

# NetXPTO - LinkPlanner

Armando Nolasco Pinto

May 22, 2018

---

# Contents

<b>1 Preface</b>	<b>7</b>
<b>2 Introduction</b>	<b>8</b>
<b>3 Simulator Structure</b>	<b>9</b>
3.1 System . . . . .	9
3.2 Blocks . . . . .	9
3.3 Signals . . . . .	9
3.4 Log File . . . . .	10
3.4.1 Introduction . . . . .	10
3.4.2 Parameters . . . . .	10
3.4.3 Output File . . . . .	11
3.4.4 Testing Log File . . . . .	12
Bibliography . . . . .	13
3.5 Input Parameters System . . . . .	13
3.5.1 Introduction . . . . .	13
3.5.2 How To Include The IPS In Your System . . . . .	14
<b>4 Development Cycle</b>	<b>17</b>
<b>5 Visualizer</b>	<b>18</b>
<b>6 Case Studies</b>	<b>19</b>
6.1 QPSK Transmitter . . . . .	19
Bibliography . . . . .	21
6.2 Optical Detection . . . . .	22
6.2.1 Theoretical Analysis . . . . .	22
6.2.2 Simulation Analysis . . . . .	35
6.2.3 Experimental Analysis . . . . .	41
6.2.4 Comparative analysis . . . . .	45
6.2.5 Known problems . . . . .	46

<i>Contents</i>	2
Bibliography . . . . .	47
6.3 BPSK Transmission System . . . . .	48
6.3.1 Theoretical Analysis . . . . .	48
6.3.2 Simulation Analysis . . . . .	49
6.3.3 Comparative Analysis . . . . .	53
Bibliography . . . . .	55
6.4 M-QAM Transmission System . . . . .	56
6.4.1 Theoretical Analysis . . . . .	57
6.4.2 Simulation Analysis . . . . .	64
6.4.3 Experimental Analysis . . . . .	89
6.4.4 DSP . . . . .	116
6.4.5 Open Issues . . . . .	116
Bibliography . . . . .	117
6.5 Kramers-Kronig Transceiver . . . . .	118
6.5.1 Theoretical Analysis . . . . .	118
6.5.2 MATLAB Simulation . . . . .	129
Bibliography . . . . .	140
6.6 DSP Laser Phase Noise Compensation . . . . .	141
6.6.1 Theoretical Analysis . . . . .	141
6.6.2 Simulation Analysis . . . . .	143
6.6.3 Simulation Results . . . . .	144
Bibliography . . . . .	148
6.7 Quantum Random Number Generator . . . . .	149
6.7.1 Theoretical Analysis . . . . .	149
6.7.2 Simulation Analysis . . . . .	151
6.7.3 Experimental Analysis . . . . .	156
6.7.4 Open Issues . . . . .	158
Bibliography . . . . .	159
6.8 BB84 with Discrete Variables . . . . .	160
6.8.1 Protocol Analysis . . . . .	160
6.8.2 Simulation Analysis . . . . .	164
6.8.3 Open Issues . . . . .	175
Bibliography . . . . .	176
6.9 Quantum Oblivious Key Distribution with Discrete Variables . . . . .	177
6.9.1 Theoretical Description . . . . .	177
6.9.2 Simulation Analysis . . . . .	188
6.9.3 Experimental Setup . . . . .	192
6.9.4 Comparative Analysis . . . . .	194
Bibliography . . . . .	195
6.10 Quantum Noise . . . . .	196
6.10.1 Theoretical Analysis . . . . .	196
6.10.2 Numerical Analysis . . . . .	197

6.10.3 Experimental Analysis . . . . .	200
Bibliography . . . . .	201
6.11 Frequency and Phase Recovery in CV-QC Systems . . . . .	202
6.11.1 Classical Frequency and Phase Recovery - State of the art . . . . .	202
6.11.2 Quantum	
Frequency Mismatch and Carrier Phase Noise Compensation - State of the art . . . . .	204
6.11.3 Novel Frequency and Phase Mismatch Compensation Algorithm . . . . .	205
6.11.4 Validation of Proposed Method . . . . .	208
6.11.5 Simulation Analysis . . . . .	245
Bibliography . . . . .	248
6.12 BPSK Transmission System . . . . .	249
6.12.1 Theoretical Analysis . . . . .	249
6.12.2 Simulation Analysis . . . . .	250
6.12.3 Comparative Analysis . . . . .	254
Bibliography . . . . .	256
6.13 Intradyne Continuous Variables QKD Transmission System . . . . .	257
6.13.1 Theoretical Analysis . . . . .	257
6.13.2 Simulation Analysis . . . . .	273
6.13.3 Experimental Analysis . . . . .	273
6.13.4 Comparative Analysis . . . . .	273
Bibliography . . . . .	274
6.14 Classical Multi-Party Computation . . . . .	274
6.14.1 Introduction . . . . .	274
6.14.2 Two-Party Computation . . . . .	275
6.14.3 Party Behavior Models . . . . .	275
6.14.4 MPC Problems and Solutions . . . . .	276
6.14.5 Garbled Circuit Protocol . . . . .	277
6.14.6 Hardware Description Languages . . . . .	277
6.14.7 TinyGarble . . . . .	277
6.14.8 ARM2GC . . . . .	278
Bibliography . . . . .	279
6.15 Quantum Multi-Party Computation . . . . .	280
6.15.1 Our Approach . . . . .	280
Bibliography . . . . .	283
6.16 Secure Multi-Party Computation . . . . .	284
6.16.1 Problem definition . . . . .	284
6.16.2 Secure Two-Party Computation . . . . .	284
Bibliography . . . . .	289

<b>7 Library</b>	<b>290</b>
7.1 ADC . . . . .	291
7.2 Add . . . . .	294
7.3 Balanced Beam Splitter . . . . .	295
7.4 Bit Error Rate . . . . .	296
Bibliography . . . . .	298
7.5 Binary Source . . . . .	299
7.6 Bit Decider . . . . .	303
7.7 Clock . . . . .	304
7.8 Clock_20171219 . . . . .	306
7.9 Coupler 2 by 2 . . . . .	309
7.10 Carrier Phase Compensation . . . . .	310
7.11 Decoder . . . . .	313
7.12 Discrete To Continuous Time . . . . .	315
7.13 DSP . . . . .	317
7.14 Electrical Signal Generator . . . . .	319
7.14.1 ContinuousWave . . . . .	319
7.15 Fork . . . . .	321
7.16 Gaussian Source . . . . .	322
7.17 MQAM Receiver . . . . .	324
7.18 IQ Modulator . . . . .	328
7.19 Local Oscillator . . . . .	330
7.20 Local Oscillator . . . . .	332
7.21 MQAM Mapper . . . . .	335
7.22 MQAM Transmitter . . . . .	338
7.23 Netxpto . . . . .	342
7.23.1 Version 20180118 . . . . .	344
7.24 Alice QKD . . . . .	345
7.25 Polarizer . . . . .	347
7.26 Probability Estimator . . . . .	348
7.27 Bob QKD . . . . .	351
7.28 Eve QKD . . . . .	352
7.29 Rotator Linear Polarizer . . . . .	353
7.30 Optical Switch . . . . .	355
7.31 Optical Hybrid . . . . .	356
7.32 Photodiode pair . . . . .	358
7.33 Photoelectron Generator . . . . .	361
Bibliography . . . . .	366
7.34 Pulse Shaper . . . . .	367
7.35 Quantizer . . . . .	369
7.36 Resample . . . . .	372
7.37 Sampler . . . . .	374

<i>Contents</i>	5
7.38 SNR of the Photoelectron Generator . . . . .	376
Bibliography . . . . .	381
7.39 Sink . . . . .	382
7.40 White Noise . . . . .	383
7.41 Ideal Amplifier . . . . .	386
<b>8 Mathlab Tools</b>	<b>388</b>
8.1 Generation of AWG Compatible Signals . . . . .	389
8.1.1 sgnToWfm.m . . . . .	389
8.1.2 sgnToWfm_20171121.m . . . . .	390
8.1.3 Loading a signal to the Tektronix AWG70002A . . . . .	392
<b>9 Algorithms</b>	<b>396</b>
9.1 Fast Fourier Transform . . . . .	397
Bibliography . . . . .	413
9.2 Overlap-Save Method . . . . .	414
Bibliography . . . . .	437
9.3 Filter . . . . .	438
Bibliography . . . . .	446
9.4 Hilbert Transform . . . . .	447
Bibliography . . . . .	451
<b>10 Code Development Guidelines</b>	<b>452</b>
<b>11 Building C++ Projects Without Visual Studio</b>	<b>453</b>
11.1 Installing Microsoft Visual C++ Build Tools . . . . .	453
11.2 Adding Path To System Variables . . . . .	453
11.3 How To Use MSBuild To Build Your Projects . . . . .	454
11.4 Known Issues . . . . .	454
11.4.1 Missing ucrtbased.dll . . . . .	454
<b>12 Git Helper</b>	<b>455</b>
12.1 Data Model . . . . .	455
12.2 Refs . . . . .	457
12.3 Tags . . . . .	457
12.4 Branch . . . . .	457
12.5 Heads . . . . .	457
12.6 Database Folders and Files . . . . .	457
12.6.1 Objects Folder . . . . .	457
12.6.2 Refs Folder . . . . .	458
12.7 Git Spaces . . . . .	458
12.8 Workspace . . . . .	459
12.9 Index . . . . .	459
12.10 Merge . . . . .	459

<i>Contents</i>	6
12.10.1 Fast-Forward Merge . . . . .	459
12.10.2 Resolve . . . . .	459
12.10.3 Recursive . . . . .	459
12.10.4 Octopus . . . . .	459
12.10.5 Ours . . . . .	459
12.10.6 Subtree . . . . .	459
12.10.7 Custom . . . . .	459
12.11 Commands . . . . .	459
12.11.1 Porcelain Commands . . . . .	459
12.11.2 Pluming Commands . . . . .	460
12.12 The Configuration Files . . . . .	461
12.13 Pack Files . . . . .	461
12.14 Applications . . . . .	461
12.14.1 Meld . . . . .	461
12.14.2 GitKraken . . . . .	461
12.15 Error Messages . . . . .	461
12.15.1 Large files detected . . . . .	461
<b>13 Simulating VHDL Programs with GHDL</b>	<b>463</b>
13.1 Adding Path To System Variables . . . . .	463
13.2 Using GHDL To Simulate VHDL Programs . . . . .	464
13.2.1 Requirements . . . . .	464
13.2.2 Option 1 . . . . .	464
13.2.3 Option 2 . . . . .	464
13.2.4 Simulation Output . . . . .	464

## **Chapter 1**

---

## **Preface**

Th

## **Chapter 2**

---

### **Introduction**

LinkPlanner is devoted to the simulation of point-to-point links.

## Chapter 3

### Simulator Structure

---

LinkPlanner is a signals open-source simulator.

The major entity is the system.

A system comprises a set of blocks.

The blocks interact with each other through signals.

#### 3.1 System

#### 3.2 Blocks

#### 3.3 Signals

List of available signals:

- Signal

##### PhotonStreamXY

A single photon is described by two amplitudes  $A_x$  and  $A_y$  and a phase difference between them,  $\delta$ . This way, the signal PhotonStreamXY is a structure with two complex numbers,  $x$  and  $y$ .

##### PhotonStreamXY\_MP

The multi-path signals are used to simulate the propagation of a quantum signal when the signal can follow multiple paths. The signal has information about all possible paths, and a measurement performed in one path immediately affects all other possible paths. From a Quantum approach, when a single photon with a certain polarization angle reaches a 50 : 50 Polarizer, it has a certain probability of follow one path or another. In order to simulate this, we have to use a signal PhotonStreamXY\_MP, which contains information about all the paths available. In this case, we have two possible paths: 0 related with horizontal and 1 related with vertical. This signal is the same in both outputs of the polarizer. The first decision is made by the detector placed on horizontal axis. Depending on that decision, the information about the other path 1 is changed according to the result of the path 0. This way, we guarantee the randomness of the process. So, signal PhotonStreamXY\_MP is a structure of two PhotonStreamXY indexed by its path.

## 3.4 Log File

### 3.4.1 Introduction

The Log File allows for a detailed analysis of a simulation. It will output a file containing the timestamp when a block is initialized, the number of samples in the buffer ready to be processed for each input signal, the signal buffer space for each output signal and the amount of time in seconds that took to run each block. Log File is enabled by default, so no change is required. If you want to turn it off, you must call the set method for the `logValue` and pass `false` as argument. This must be done before method `run()` is called, as shown in line 125 of Figure 3.1.

```

115
116  // #####
117  // ##### System Declaration and Initialization #####
118  // #####
119
120 System MainSystem{ vector<Block*> { &B1, &B2, &B3, &B4, &B5, &B6, &B7, &B8} };
121
122 // #####
123 // ##### System Run #####
124 // #####
125 MainSystem.setLogValue(false);
126 MainSystem.run();

```

Figure 3.1: Disabling Log File

### 3.4.2 Parameters

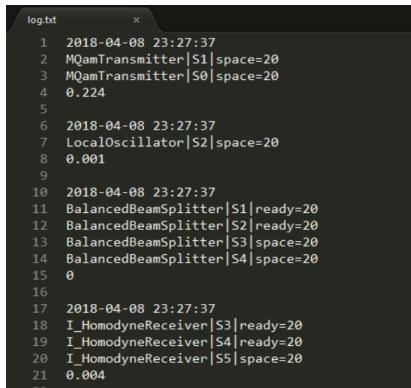
The Log File accepts two parameters: `logFileName` which correspond to the name of the output file, i.e., the file that will contain all the information listed above and `logValue` which will enable the Log File if `true` and will disable it if `false`.

Log File Parameters		
Parameter	Type	Default Value
<code>logFileName</code>	string	"log.txt"
<code>logValue</code>	bool	true

Available Set Methods		
Parameter	Type	Comments
<code>setLogFileName(string newName)</code>	void	Sets the name of the output file to the name given as argument
<code>setLogValue(bool value)</code>	void	Sets the value of <code>logValue</code> to the value given as argument

### 3.4.3 Output File

The output file will contain information about each block. From top to bottom, the output file shows the timestamp (time when the block was started), the number of samples in the buffer ready to be processed for each input signal and the signal buffer space for each output signal. This information is taken before the block has been executed. The amount of time, in seconds, that each block took to run, is also registered. Figure 3.2 shows a portion of an output file. In this example, 4 blocks have been run: MQamTransmitter, LocalOscillator, BalancedBeamSplitter and I\_HomodyneReceiver. In the case of the I\_HomodyneReceiver block we can see that the block started being ran at 23:27:37 and finished running 0.004 seconds later.



```

log.txt
1 2018-04-08 23:27:37
2 MQamTransmitter|S1|space=20
3 MQamTransmitter|S0|space=20
4 0.224
5
6 2018-04-08 23:27:37
7 LocalOscillator|S2|space=20
8 0.001
9
10 2018-04-08 23:27:37
11 BalancedBeamSplitter|S1|ready=20
12 BalancedBeamSplitter|S2|ready=20
13 BalancedBeamSplitter|S3|space=20
14 BalancedBeamSplitter|S4|space=20
15 0
16
17 2018-04-08 23:27:37
18 I_HomodyneReceiver|S3|ready=20
19 I_HomodyneReceiver|S4|ready=20
20 I_HomodyneReceiver|S5|space=20
21 0.004

```

Figure 3.2: Output File Example

Figure 3.3 shows a portion of code that consists in the declaration and initialization of the I\_HomodyneReceiver block. In line 97, we can see that the block has 2 input signals,  $S_3$  and  $S_4$ , and is assigned 1 output signal,  $S_5$ . Going back to Figure 3.2 we can observe that  $S_3$  and  $S_4$  have 20 samples ready to be processed and the buffer of  $S_5$  is empty.

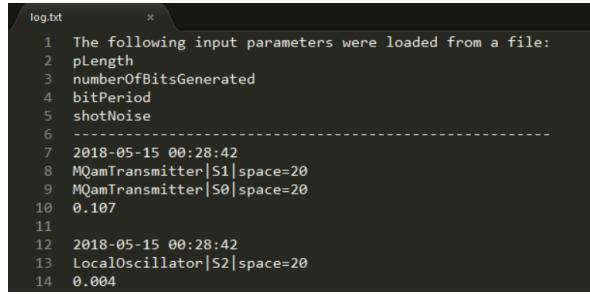
```

97  I_HomodyneReceiver B4{ vector<Signal*> {&S3, &S4}, vector<Signal*> {&S5} };
98  B4.useShotNoise(true);
99  B4.setElectricalNoiseSpectralDensity(electricalNoiseAmplitude);
100 B4.setGain(amplification);
101 B4.setResponsivity(responsivity);
102 B4.setSaveInternalSignals(true);
103

```

Figure 3.3: I-Homodyne Receiver Block Declaration

The list of the input parameters loaded from a file is presented at the top of the output file, as shown in Figure 3.4.



```
log.txt
1 The following input parameters were loaded from a file:
2 pLength
3 numberOfBitsGenerated
4 bitPeriod
5 shotNoise
6 -----
7 2018-05-15 00:28:42
8 MQamTransmitter|S1|space=20
9 MQamTransmitter|S0|space=20
10 0.107
11
12 2018-05-15 00:28:42
13 Localoscillator|S2|space=20
14 0.004
```

Figure 3.4: Four input parameters where loaded from a file

### 3.4.4 Testing Log File

In directory *doc/tex/chapter/simulator\_structure/test\_log\_file/bpsk\_system/* there is a copy of the BPSK system. You may use it to test the Log File. The main method is located in file *bpsk\_system\_sdf.cpp*

## 3.5 Input Parameters System

### 3.5.1 Introduction

With the Input Parameters System (IPS) it is possible to read the input parameters from a file.

#### Format Of The Input File

In Figure 3.5, it is possible to observe the contents of the file **input\_parameters\_0.txt** used to load the values of some of the BPSK system's input parameters. The input file must respect the following properties:

1. Input parameter values can be changed by adding a line in the following format: **paramName=newValue**, where **paramName** is the name of the input parameter and **newValue** is the value to be assigned.
2. IPS supports scientific notation. This notation works for the lower case character **e** and the upper case character **E**.
3. If an input parameters is assigned the wrong type of value, method **readSystemInputParameters()** will throw an exception.
4. Not all input parameters need to be changed.
5. The IPS supports comments in the form of the characters **//**. The comments will only be recognized if placed at the beginning of a line.

```

input_parameters_0.txt *
1 //-----BPSK PARAMETERS-----
2 //Changes the value of parameter pLength
3 pLength=5
4 //Is able to change the value of different types of parameters
5 //Of type int
6 numberOfBitsGenerated=1e3
7 //Of type double
8 bitPeriod=20e-12
9 //Of type bool
10 shotNoise=false

```

Figure 3.5: Content of file **input\_parameters\_0.txt**

### Loading Input Parameters From A File

Execute the following command in the Command Line:

```
.\some_system.exe <input_file_path> <output_directory>
```

where `some_system.exe` is the name of the executable generated after compiling the project, `<input_file_path>` is the path to the file containing the new input parameters; `<output_directory>` is the directory where the output signals will be written into.

#### 3.5.2 How To Include The IPS In Your System

In this illustrative example, the code of the BPSK System will be used. To implement the IPS the following requirements must be met:

1. Your system must include `netxpto_20180418.h` or later.
2. A class that will contain the system input parameters must be created. This class must be a derived class of `SystemInputParameters`. In this case the created class is called `BPSKInputParameters`.
3. The created class must have 2 constructors. The implementation of these constructors is the same as `BPSKInputParameters`.

```
BPSKInputParameters();
BPSKInputParameters(int argc, char*argv[]);
```

4. The created class must contain the method `initializeInputParameterMap()`. For every input parameter `addInputParameter(paramName,paramAddress)` must be called, where `paramName` is a string that represents the name of your input parameter and `paramAddress` is the address of your input parameter.

```
void initializeInputParameterMap() {
    //Add parameters
}
```

5. All signals must be instantiated using the constructor that takes as argument, the file name and the folder name, according to the type of signal.

```
Binary S0("S0.sgn", param.getOutputFolderName()) //S0 is a Binary signal
```

6. Method `main` must receive the following arguments.

```
int main(int argc, char*argv[]){...}
```

7. The `MainSystem` must be instantiated using the following line of code. The ... represent the list of blocks.

```
System MainSystem{ vector<Block*>
    {...},param.getOutputFolderName(),param.getLoadedInputParameters()};
```

The following code represents the input parameters class, **BPSKInputParameters**, and must be changed according to the system you are working on.

```
class BPSKInputParameters : public SystemInputParameters {
public:
    //INPUT PARAMETERS
    int numberOfBitsReceived{ -1 };
    int numberOfBitsGenerated{ 1000 };
    int samplesPerSymbol = 16;
    (...)

    /* Initializes default input parameters */
    BPSKInputParameters() : SystemInputParameters() {
        initializeInputParameterMap();
    }

    /* Initializes input parameters according to the program arguments */
    /* Usage: .\bpsk_system.exe <input_parameters.txt> <output_directory> */
    BPSKInputParameters(int argc, char*argv[]) : SystemInputParameters(argc,argv) {
        initializeInputParameterMap();
        readSystemInputParameters();
    }

    //Each parameter must be added to the parameter map by calling addInputParameter(string,param*)
    void initializeInputParameterMap() {
        addInputParameter("numberOfBitsReceived", &numberOfBitsReceived);
        addInputParameter("numberOfBitsGenerated", &numberOfBitsGenerated);
        addInputParameter("samplesPerSymbol", &samplesPerSymbol);
        (...)

    }
};
```

The method **main** should look similar to the following code.

```
int main(int argc, char*argv[]) {

    BPSKInputParameters param(argc, argv);

    //Signal Declaration and Initialization
    Binary S0("S0.sgn", param.getOutputFolderName());
    S0.setBufferLength(param.bufferLength);

    OpticalSignal S1("S1.sgn", param.getOutputFolderName());
    S1.setBufferLength(param.bufferLength);
    (...)

    //System Declaration and Initialization
    System MainSystem{ vector<Block*> { &B1, &B2, &B3, &B4, &B5, &B6, &B7,
        &B8},param.getOutputFolderName(),param.getLoadedInputParameters() };

    //System Run
    MainSystem.run();

    return 0;
}
```

The class **SystemInputParameters**, has the following constructors and methods available:

SystemInputParameters - Constructors	
Constructors	Comments
SystemInputParameters()	Creates an object of SystemInputParameters with the default input parameters' values
SystemInputParameters(int argc, char*argv[])	Creates an object of SystemInputParameters and loads the values according to the program arguments passed in the command line

SystemInputParameters - Methods		
Method	Type	Comments
addInputParameter(string name, int* variable)	void	Adds an input parameter whose value is of type int
addInputParameter(string name, double* variable)	void	Adds an input parameter whose value is of type double
addInputParameter(string name, bool* variable)	void	Adds an input parameter whose value is of type bool
readSystemInputParameters()	void	Reads the input parameters from a file.

## Chapter 4

## Development Cycle

---

The NetXPTO-LinkPlanner has been developed by several people using git as a version control system. The NetXPTO-LinkPlanner repository is located in the GitHub site <http://github.com/netxpto/linkplanner>. The more updated functional version of the software is in the branch master. Master should be considered a functional beta version of the software. Periodically new releases are delivered from the master branch under the branch name Release<Year><Month><Day>. The integration of the work of all people is performed by Armando Nolasco Pinto in the branch Develop. Each developer has his/her own branch with his/her name.

## Chapter 5

---

## Visualizer

visualizer

# Chapter 6

## Case Studies

### 6.1 QPSK Transmitter

---

2017-08-25, Review, Armando Nolasco Pinto

---

This system simulates a QPSK transmitter [1]. A schematic representation of this system is shown in figure 6.1.

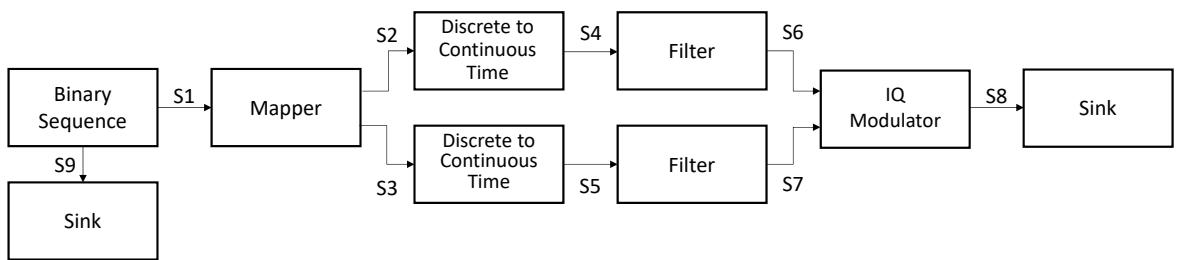


Figure 6.1: QPSK transmitter block diagram.

#### System Input Parameters

**Parameter:** *sourceMode*

**Description:** Specifies the operation mode of the binary source.

**Accepted Values:** PseudoRandom, Random, DeterministicAppendZeros, DeterministicCyclic.

**Parameter:** *patternLength*

**Description:** Specifies the pattern length used by the source in the PseudoRandom mode.

**Accepted Values:** Integer between 1 and 32.

**Parameter:** *bitStream*

**Description:** Specifies the bit stream generated by the source in the DeterministicCyclic and DeterministicAppendZeros mode.

**Accepted Values:** "XXX..", where X is 0 or 1.

**Parameter:** *bitPeriod*

**Description:** Specifies the bit period, i.e. the inverse of the bit-rate.

**Accepted Values:** Any positive real value.

**Parameter:** *iqAmplitudes*

**Description:** Specifies the IQ amplitudes.

**Accepted Values:** Any four pair of real values, for instance  $\{ \{ 1,1 \}, \{ -1,1 \}, \{ -1,-1 \}, \{ 1,-1 \} \}$ , the first value correspond to the "00", the second to the "01", the third to the "10" and the forth to the "11".

**Parameter:** *numberOfBits*

**Description:** Specifies the number of bits generated by the binary source.

**Accepted Values:** Any positive integer value.

**Parameter:** *numberOfSamplesPerSymbol*

**Description:** Specifies the number of samples per symbol.

**Accepted Values:** Any positive integer value.

**Parameter:** *rollOffFactor*

**Description:** Specifies the roll off factor in the raised-cosine filter.

**Accepted Values:** A real value between 0 and 1.

**Parameter:** *impulseResponseTimeLength*

**Description:** Specifies the impulse response window time width in symbol periods.

**Accepted Values:** Any positive integer value.

## References

- [1] Rodney Loudon. *The Quantum Theory of Light*. Oxford University Press, 2000.

## 6.2 Optical Detection

<b>Contributors</b>	:	Nelson Muga, (2017-12-21 - ...)
	:	Diamantino Silva, (2017-08-18 - 2018-02-05)
	:	Armando Pinto (2017-08-15 - ...)
<b>Goal</b>	:	Analise of various optical detection schemes.

The detection of light is a fundamental stage in every optical communication system, bridging the optical domain into the electrical domain. This section will review various theoretical, practical, and implementation aspects of the most important light detection schemes. The objective of this work is to develop numerical models for the various optical detections schemes and to validate such numerical models with experimental results.

### 6.2.1 Theoretical Analysis

<b>Contributors</b>	:	Nelson Muga (2017-12-20 - )
	:	Diamantino Silva (2017-08-18 - ...)
	:	Armando Pinto (2017-08-15 - ...)
<b>Goal</b>	:	Theoretical description of various optical detection schemes.

Receiver schemes for the detection of optical modulated signals can be roughly divided into two basic groups: Direct detection and coherent detection. In a direct detection receiver, its photodiode only responds to changes in the receiving signal optical power, and cannot extract any phase or frequency information from the optical carrier. However, using direct detection along with additional optics, e.g. an interferometer, the phase in one symbol may be compared to the phase in the previous symbol. This is often called interferometric detection. Coherent receivers use a carrier phase reference generated at the receiver, i.e. a local oscillator, and can track the phase of an optical transmitter so as to extract any phase and frequency information carried by a transmitted signal. Table 6.1 compares the three detection techniques, including maximum number of degrees of freedom and receiver sensitivities for binary and quaternary modulations.

In this subsection, we are going to calculate the signal-to-noise ratio at the input of the decision circuit for the various detection schemes under analysis. For each detection scheme a classical and a quantum description is going to be developed and a comparative analysis is going to be performed.

Table 6.1: COMPARISON OF DETECTION TECHNIQUES. SHADING DENOTES AN ADVANTAGE (from [Kahn2004] and [Kahn2006]).

Attribute	Noncoherent		Differential		Coherent
Maximum number of degrees of freedom (per polarization)	1 (magnitude)		1 (phase)		2 (two quadrature components)
Receiver sensitivity for binary	38 photons/bit (2-PAM)		20 photons/bit (2-DPSK)		18 photons/bit (2-PSK)
Receiver sensitivity for quaternary	134 photons/bit (4-PAM)		31 photons/bit (4-DPSK)		18 photons/bit (4-PSK)
	Heterodyne / Homodyne	Direct Detection	Heterodyne / Homodyne	Direct Detection	Heterodyne / Homodyne
Electrical filtering can be used to select channel (enables frequency-agile receiver)	Yes	No	Yes	No	Yes
Chromatic dispersion is linear distortion (enables effective electrical compensation)	Yes	No	Yes	No	Yes
Local oscillator laser required at receiver	Yes	No	Yes	No	Yes
Polarization control or diversity required at receiver	Yes	No	Yes	No	Yes

## Classical Description

<b>Contributors</b>	:	Nelson Muga (2017-12-20 - )
	:	Diamantino Silva (2017-08-18 - ...)
	:	Armando Pinto (2017-08-15 - ...)
<b>Goal</b>	:	Develop a classical description of various optical detection schemes.

### Direct Detection

Direct detection schemes are convincingly simple as no phase, frequency or polarization control is necessary to recover the intensity of the optical field. This section will focus only the recover of the intensity of the optical field through the calculation of the current intensity at the output of a photodiode.

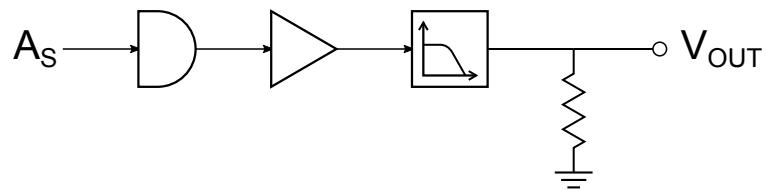


Figure 6.2: Noncoherent detection technique with direct detection implementation.

The electric field generated by an monochromatic single mode laser arriving at the input of the optical receiver can be written as

$$E_s(t) = \sqrt{P_s} \cdot A(t) \cdot e^{i(\omega_s t + \varphi_s)} \quad (6.1)$$

where  $P_s$  is the optical power,  $\omega_s$  is the carrier frequency,  $\varphi_s$  is the initial phase, and  $A = a(t)e^{i\varphi(t)}$  is the modulated envelop amplitude, with  $a(t)$  and  $\varphi(t)$  standing for the modulated amplitude and phase, respectively.

The output current of the photodiode,  $I_{DD}(t)$  can be written as

$$I_{DD}(t) = R \cdot E_s(t) \cdot E_s^*(t) + i_{sh} + i_{th} \quad (6.2)$$

$$= R \cdot a^2(t) \cdot P_s + i_{sh} + i_{th}, \quad (6.3)$$

where  $R$  represents the responsivity of the photodiode, which is equal to

$$R = \eta \frac{2\pi q}{h\omega_s}, \quad (6.4)$$

with  $q = 1.6 \cdot 10^{-19}$  C being the charge of the electron,  $h\omega_s/2\pi$  is the energy per photon with  $h = 6.63 \cdot 10^{-34}$  J·s being the Planck constant, and  $\eta$  is the fundamental quantum efficiency of the photodiode that corresponds to the averaged number of electrons generated per photon (Ref.Agrawal2013).

The two last terms on the right hand side of equation (6.3),  $i_{sh}$  and  $i_{th}$ , represent the current fluctuation related to the shot noise and thermal noise effects, respectively. From the classical interpretation, the shot noise is induced by the optical-to-electrical conversion process. Mathematically,  $i_{sh}(t)$  is a stationary random process with Poisson statistics, whose autocorrelation function is related with to the spectral density (Ref.Agrawal2012)

$$\langle i_{sh}(t)i_{sh}(t + \tau) \rangle = \int_{-\infty}^{+\infty} S_{sh}(f) e^{i2\pi f \tau} df \quad (6.5)$$

where  $\langle \cdot \rangle$  stands for the ensemble average over fluctuations. The two-sided spectral density (notice that the integral in (6.5) includes both negative and positive frequencies) of the shot noise is a constant term and can be written as

$$S_{sh}(f) = qI_p \quad (6.6)$$

where  $I_p$  is the average output current of the photodiode presented in equation (6.2), i.e.,

$$I_p(t) = I_{DD}(t) - (i_{sh} + i_{th}). \quad (6.7)$$

The shot noise variance can be calculated by setting  $\tau = 0$

$$\sigma_{sh}^2 = \langle i_{sh}(t)^2 \rangle = \int_{-\infty}^{+\infty} S_{sh}(f) df = 2qI_p B \quad (6.8)$$

where  $B$  is the photodetector bandwidth. The thermal noise term represented in (6.3) is related with the random thermal motion of electrons in the load resistor in the front end of the receiver, see Fig. 6.2. Mathematically, this noise contribution is modeled as a stationary Gaussian random process with a two-sided spectral density approximately constant (ref.Agrawal), given by

$$S_{th}(f) = 2k_B T / R_L \quad (6.9)$$

where  $k_B$  is the Boltzmann constant,  $T$  is the absolute temperature and  $R_L$  is the load resistor. Considering a autocorrelation function given by (6.5), with the subscripts  $sh$  replaced by  $th$ , and setting  $\tau = 0$ , the noise variance becomes

$$\sigma_{th}^2 = \langle i_{th}(t)^2 \rangle = \int_{-\infty}^{+\infty} S_{th}(f) df = \frac{4k_B T}{R} B. \quad (6.10)$$

It is worth noticing that: a) de photodiode bandwidth appears on the expressions of both noise terms; b) in contrast with the shot noise case, the variance of the thermal noise does not depend on the average photocurrent  $I_p$ .

Since the shot and thermal noise are independent random processes with approximately Gaussian statistics, the total variance of the current fluctuations,  $\Delta i = i_{sh} + i_{th}$ , can be written as the sum of the individual variances presented in (6.11) and (6.10)

$$\sigma_{\Delta I_{DD}}^2 = \langle \Delta I_{DD}^2 \rangle = \left( 2qI_p + \frac{4k_B T}{R} \right) B \quad (6.11)$$

Once obtained the variance of the photocurrent one can easily compute the signal-to-noise-ratio (SNR) of the optical receiver, which is a key parameter on the performance assessment of the device. The SNR of a electrical signal is defined as the ratio of the average signal power over the noise power, which leads to

$$SNR = \frac{I_p^2}{\sigma_{\Delta I_{DD}}^2} \quad (6.12)$$

Using (6.7) and (6.11) into the previous equation, one obtains the SNR as a function of the optical field impinging the photodiode

$$SNR = \frac{R^2 (a^2(t) \cdot P_s)^2}{(2qI_p + 4k_B T / R) B} \quad (6.13)$$

—VV—VV—VV—VV—VV—VV—

We will consider that the detector has a bandwidth  $B$ , greater than the signal  $A(t)$ , but much smaller than  $2\omega_0$ . The calculation of power incident in the photodiode is given by the expected value of the square of the amplitude during a time interval  $\Delta t = 2\pi/\omega$

Measurable optical power, assuming that the detector bandwidth,  $B$ , is greater than the signal,  $A(t)$ , bandwidth but much small than  $2\omega_0$

$$\begin{aligned} P(t) &= \overline{E_R^2(t)} \\ &= \overline{|A(t)|^2} + \overline{|A(t)|^2 \cos(-2\omega t + 2\theta(t))} \\ &= |A(t)|^2 \quad W \end{aligned} \quad (6.14)$$

To simplify calculations, the electric field can be expressed the complex notation

$$E(t) = A(t)e^{-i\omega_0 t} \quad (6.15)$$

The physically measurable quantities are obtained by taking the real part of the complex wave. Using this notation, the beam power,  $P(t)$ , is obtained by multiplying the electric field's conjugate by itself

$$\begin{aligned} P(t) &= E^*(t)E(t) \\ &= |A(t)|^2 \end{aligned} \quad (6.16)$$

$$i(t) = \eta q \frac{P(t)}{\hbar\omega_0} \quad (6.17)$$

in which  $\eta$  is the photodiode's responsivity,  $q$  is the unit charge and  $P(t)/\hbar\omega_0$  is the number of removed electrons.

which will use to express the second moment of the photocurrent as

$$\langle i^2(t) \rangle = \eta^2 q^2 \frac{\langle |A(t)|^4 \rangle}{\hbar^2 \omega_0^2} \quad (6.18)$$

Assuming a fase modulation, in which the amplitude is constant, the signal is simplified to

$$A(t) = |A|e^{i\theta} \quad (6.19)$$

Therefore, the current becomes constant

$$i(t) = I_0 = \eta q \frac{A_s^2}{\hbar\omega_0} \quad (6.20)$$

and it's second moment becomes simply

$$\langle i^2(t) \rangle = I_0^2 \quad (6.21)$$

## Shot noise in photodiodes

$$\langle i_n^2(t) \rangle = 2qBI_0 \quad (6.22)$$

The signal to noise ratio is obtained by the relation between the second moment of the signal to the second moment of the noise

$$\begin{aligned} \frac{S}{N} &= \frac{\langle i^2(t) \rangle}{\langle i_n^2(t) \rangle} \\ &= \frac{I_0}{2qB} \\ &= \eta \frac{|A|^2}{\hbar\omega_0 B} \end{aligned} \quad (6.23)$$

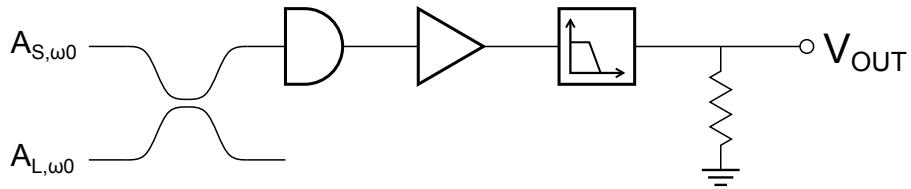
*Homodyne Detection*

Figure 6.3: Homodyne detection.

The homodyne detection scheme uses an auxiliary local oscillator, which is combined in a beamsplitter with the signal beam. After this step, it is similar to the direct detection. As we will see this has some implications in the phase detection???

Given a splitter with intensity transmission  $\epsilon$ , the resulting field incident to the photodetector is [1]

$$E(t) = \sqrt{\epsilon}E_S(t) + \sqrt{1-\epsilon}E_{LO}(t) \quad (6.24)$$

in which  $E_{LO} = e^{i\omega_0 t}$ . Given a local oscillator with a much larger power than the signal, then, the incident power in the photodiode is

$$P(t) = \eta \left[ (1-\epsilon)P_{LO}(t) + 2\sqrt{\epsilon(1-\epsilon)}\text{Re}[E_S(t)E_{LO}^*(t)] \right] \quad (6.25)$$

$$= \eta \left[ (1-\epsilon)P_{LO}(t) + 2\sqrt{\epsilon(1-\epsilon)}|E_S(t)||E_{LO}(t)|\cos(\phi) \right] \quad (6.26)$$

$$(6.27)$$

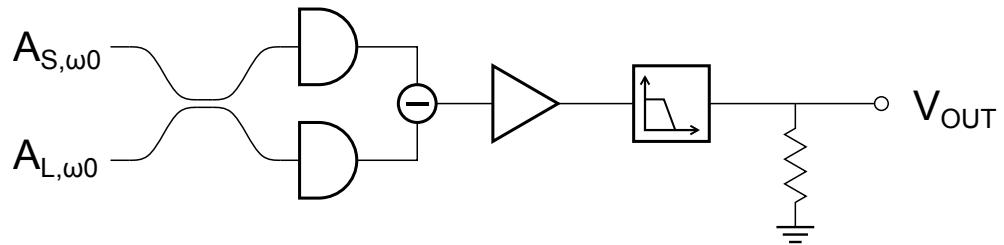
*Balanced Homodyne Detection*

Figure 6.4: Balanced homodyne detection.

"In a balanced homodyne detector (BHD), the signal to be measured is mixed with a local oscillator (LO) at a beam splitter. The interference signals from the two output ports of the beam splitter are sent to two photodiodes followed by a subtraction operation, and then, amplification may be applied. The output of a BHD can be made to be proportional to either the amplitude quadrature or the phase quadrature of the input signal depending on the relative phase between the signal and the LO".

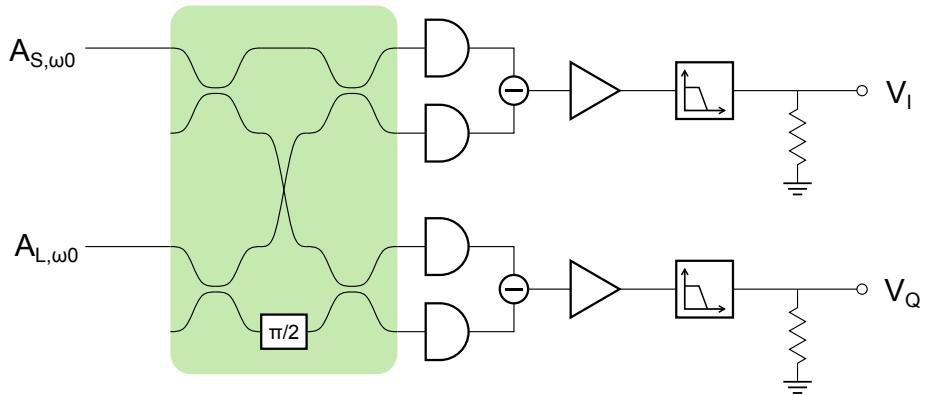
*IQ Homodyne Balanced Detection*

Figure 6.5: IQ balanced homodyne detection.

*Semiclassical model*

*Quantum model*

**Heterodyne Detection**

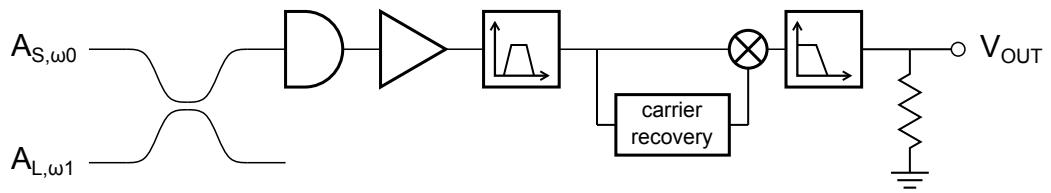


Figure 6.6: Heterodyne detection.

In contrast with the homodyne detection, in which the frequency of the signal carrier is equal to the frequency of the local oscillator, in the heterodyne detection, these frequencies are different.

Because of this, the inference will result in a new signal with an intermediate frequency at...

**Balanced Heterodyne Detection**

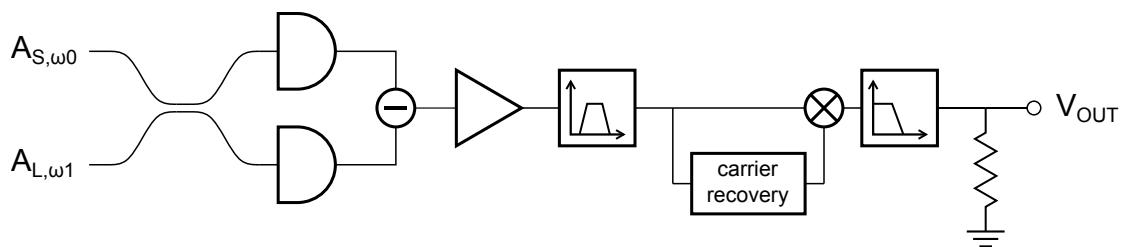


Figure 6.7: Balanced heterodyne detection.

*Semiclassical model*

### Quantum model

#### IQ Heterodyne Balanced Detection

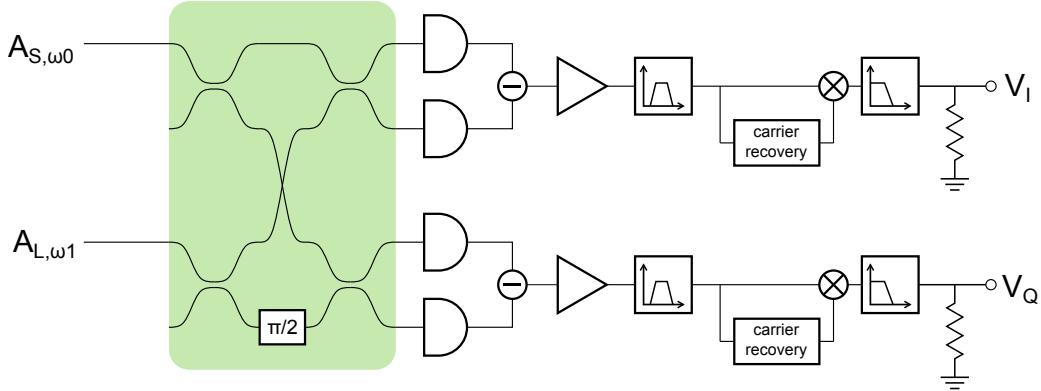


Figure 6.8: IQ balanced heterodyne detection.

#### Thermal noise

Thermal noise is generated by electrons in response to temperature. Its contribution to the resulting current can be described by the following equation [2]

$$\langle (\Delta i_T)^2 \rangle = 4K_B T_0 B / R_L \quad (6.28)$$

in which  $K_B$  is Boltzmann's constant,  $T_0$  is the absolute temperature,  $B$  is the bandwidth and  $R_L$  is the receiver load impedance. The  $B$  value is imposed by default or chosen when the measurements are made, but the  $R_L$  value is dependent in the internal setup of the various components of the detection system. Nevertheless, for simulation purposes, we can just introduce an experimental value.

#### Quantum Description

<b>Contributors</b>	:	Diamantino Silva (2017-08-18 - ...)
	:	Armando Pinto (2017-08-15 - ...)
<b>Goal</b>	:	Develop a quantum description of various optical detection schemes, and compare with the classical description.

We start by defining number states  $|n\rangle$  (or Fock states), which correspond to states with perfectly fixed number of photons [3]. Associated to those states are two operators, the creation  $\hat{a}^\dagger$  and annihilation  $\hat{a}$  operators, which in a simple way, remove or add one photon from a given number state [2]. Their action is defined as

$$\hat{a}|n\rangle = \sqrt{n}|n-1\rangle \quad (6.29), \quad \hat{a}^\dagger|n\rangle = \sqrt{n+1}|n+1\rangle \quad (6.30), \quad \hat{n}|n\rangle = n|n\rangle \quad (6.31)$$

in which  $\hat{n} = \hat{a}^\dagger\hat{a}$  is the number operator. Therefore, number states are eigenvectors of the number operator.

Coherent states have properties that closely resemble classical electromagnetic waves, and are generated by single-mode lasers well above the threshold. [3] We can define them, using number states in the following manner

$$|\alpha\rangle = e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle \quad (6.32)$$

in which the complex number  $\alpha$  is the sole parameter that characterizes it. In fact, if we calculate the expected number of photons with  $\langle\alpha|\hat{n}|\alpha\rangle$  we will obtain  $|\alpha|^2$ . The coherent state is an eigenstate of the annihilation operator,  $\hat{a}|\alpha\rangle = \alpha|\alpha\rangle$ .

Using the creation and annihilation operators, we can define two quadrature operators [3]

$$\hat{X} = \frac{1}{2} (\hat{a}^\dagger + \hat{a}) \quad (6.33), \quad \hat{Y} = \frac{i}{2} (\hat{a}^\dagger - \hat{a}) \quad (6.34)$$

The expected value of these two operators, using a coherent state  $|\alpha\rangle$  are

$$\langle\hat{X}\rangle = \text{Re}(\alpha) \quad (6.35), \quad \langle\hat{Y}\rangle = \text{Im}(\alpha) \quad (6.36)$$

We see that the expected value of these operators give us the real and imaginary part of  $\alpha$ . Now, we can obtain the uncertainty of these operators, using:

$$\text{Var}(\hat{X}) = \langle\hat{X}^2\rangle - \langle\hat{X}\rangle^2 \quad (6.37)$$

For each of these quadrature operators the variance will be

$$\text{Var}(\hat{X}) = \text{Var}(\hat{Y}) = \frac{1}{4} \quad (6.38)$$

This result shows us that for both quadratures, the variance of measurement is the same and independent of the value of  $\alpha$ .

### Homodyne detection

The measurement of a quadrature of an input signal (S) is made by using the balanced homodyne detection technique, which measures the phase difference between the input signal and a local oscillator (LO). The measurement of quadrature are made relative to a reference phase of the LO, such that if the measurement is made in-phase with this reference,

the value will be proportional to the  $\hat{X}$  quadrature of the signal. If the phase of the LO is has an offset of  $\pi/2$  relative to the reference, the output will be proportional to the  $\hat{Y}$  quadrature of the signal.

Experimentally, the balanced homodyne detection requires a local oscillator with the same frequency as the input signal, but with a much larger amplitude. These two signals are combined using a 50:50 beam splitter, from were two beams emerge, which are then converted to currents using photodides. Finally, the two currents are subtracted, resulting in an output current proportional to a quadrature of the input signal [2].

A phase of the local oscillator can be defined as the reference phase. A phase offset equal to 0 or  $\pi/2$  will give an output proportional to the signal's in-phase component or to the quadrature component, respectively. Therefore, the  $\hat{X}$  operator will correspond to the in-phase component and  $\hat{Y}$  operator correspond to quadrature component

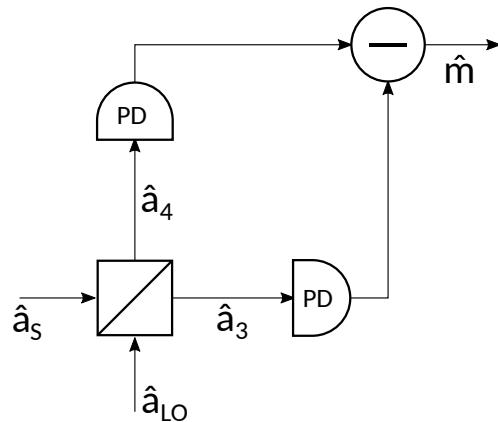


Figure 6.9: Balanced homodyne detection.

In the lab and in our simulations, a more complex system is used, the double balanced homodyne detection, which allows the simultaneous measurement of the  $\hat{X}$  and  $\hat{Y}$  components. The signal is divided in two beam with half the power of the original. One of the beams is used in a balanced homodyne detection with a local oscillator. The other beam is used in another balanced homodyne detection, but using a local oscillator with a phase difference  $\pi/2$  relative to the first one.

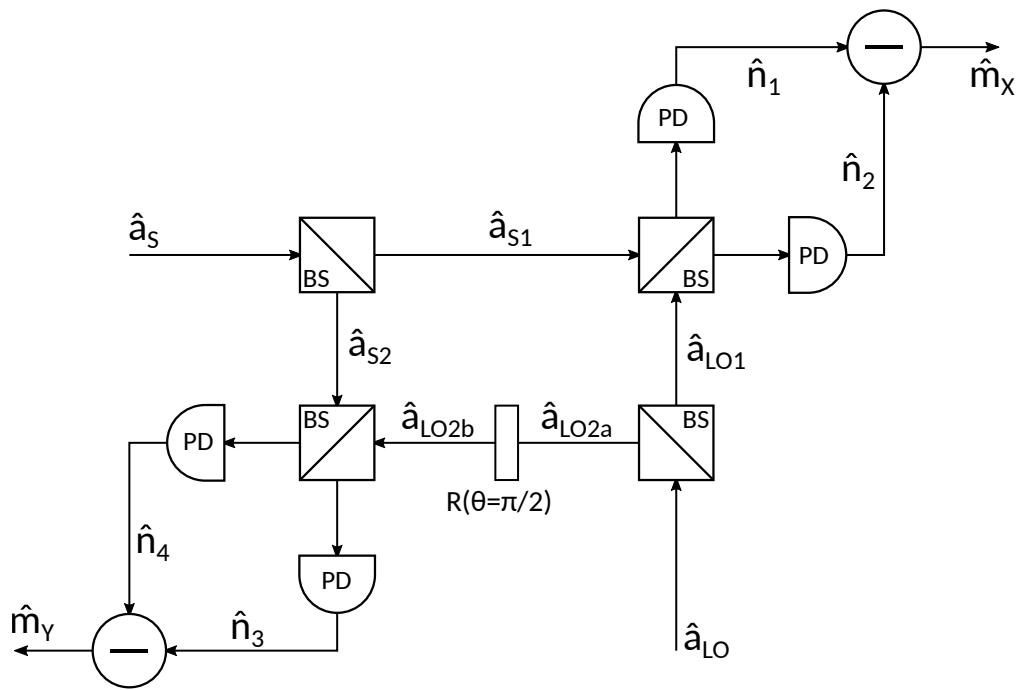


Figure 6.10: Balanced double homodyne detection.

---

## Bibliography

- [1] Joseph M. Kahn, *Modulation and Detection Techniques for Optical Communication Systems*, OSA/COTA - Coherent Optical Technologies and Applications Topical Meeting, BC, Canada, 2006.
- [2] Joseph M. Kahn, and Keang-Po Ho, Spectral Efficiency Limits and Modulation/Detection Techniques for DWDM Systems, *IEEE Journal Of Selected Topics In Quantum Electronics*, Vol. 10, No. 2, 2004.

### Noise sources in homodyne detection

The detection of light using photodiodes is subjected to various sources of noise. One of these sources is the electrical field itself. The interaction of the signal with the vacuum field adds quantum noise to the detection. Another source of noise comes from the detection system, such as photodiodes and other electrical circuits, originating various kinds of noise, such as thermal noise, dark noise and amplifier noise [4]. In the following sections, we will focus on two noise sources, quantum noise and thermal noise.

### Quantum Noise

In order to grasp this effect, the quantum mechanical description of balanced homodyne detection will be used, employing quantum operators to describe the effect of each component in the system (fig. ??). We start with the operators  $\hat{a}_S$  and  $\hat{a}_{LO}$  corresponding to the annihilation operator for the signal and local oscillator, which are the inputs in a beam divisor. The outputs will be  $\hat{a}_3$  and  $\hat{a}_4$ . Using a balanced beam splitter, we can write the output as

$$\hat{a}_3 = \frac{1}{\sqrt{2}} (\hat{a}_S + \hat{a}_{LO}) \quad (6.39), \quad \hat{a}_4 = \frac{1}{\sqrt{2}} (\hat{a}_S - \hat{a}_{LO}) \quad (6.40)$$

The final output of a homodyne measurement will be proportional to the difference between the photocurrents in arm 3 and 4. Then

$$I_{34} = I_3 - I_4 \sim \langle \hat{n}_3 - \hat{n}_4 \rangle \quad (6.41)$$

We can define an operator that describes the difference of number of photons in arm 3 and arm 4:

$$\hat{m} = \hat{a}_3^\dagger \hat{a}_3 - \hat{a}_4^\dagger \hat{a}_4 \quad (6.42)$$

If we assume that the local oscillator produces the the coherent state  $|\beta\rangle$ , then the expected value of this measurement will be

$$\langle m \rangle = 2|\alpha||\beta| \cos(\theta_\alpha - \theta_\beta) \quad (6.43), \quad \text{Var}(m) = |\alpha|^2 + |\beta|^2 \quad (6.44)$$

The local oscillator normally has a greater power than the signal , then  $|\alpha| \ll |\beta|$ . If we use as unit,  $2|\beta|$ , then these two quantities can be simplified to

$$\langle m \rangle = |\alpha| \cos(\theta_\alpha - \theta_\beta) \quad (6.45), \quad \text{Var}(m) \approx \frac{1}{4} \quad (6.46)$$

[4]

Has we have seen previously, in order to measure two quadratures simultaneously, we can use double balanced homodyne detection. For each quadrature, the input signal now has half the power, so  $|\alpha| \rightarrow |\alpha/\sqrt{2}|$ . If we use a local oscillator that produces states  $|\beta\rangle$ , then we can divide it in two beams in state  $|\beta/\sqrt{2}\rangle$  and  $|i\beta/\sqrt{2}\rangle$  which will be used in each homodyne detection. In this setting, the expected values for each quadrature,  $X$  and  $Y$ , (in normalized values of  $\sqrt{2}|\beta|$ ) are

$$\langle m_X \rangle = \left| \frac{\alpha}{\sqrt{2}} \right| \cos(\theta_\alpha - \theta_\beta) \quad (6.47), \quad \text{Var}(m_X) \approx \frac{1}{4} \quad (6.48)$$

$$\langle m_Y \rangle = \left| \frac{\alpha}{\sqrt{2}} \right| \sin(\theta_\alpha - \theta_\beta) \quad (6.49), \quad \text{Var}(m_Y) \approx \frac{1}{4} \quad (6.50)$$

Therefore the measurement of each quadrature will have half the amplitude, but the same variance.

### 6.2.2 Simulation Analysis

<b>Contributors</b>	:	Diamantino Silva, (2017-08-18 - ...)
<b>Goal</b>	:	Simulation of various optical detection schemes.
<b>Directory</b>	:	sdf/optical_detection

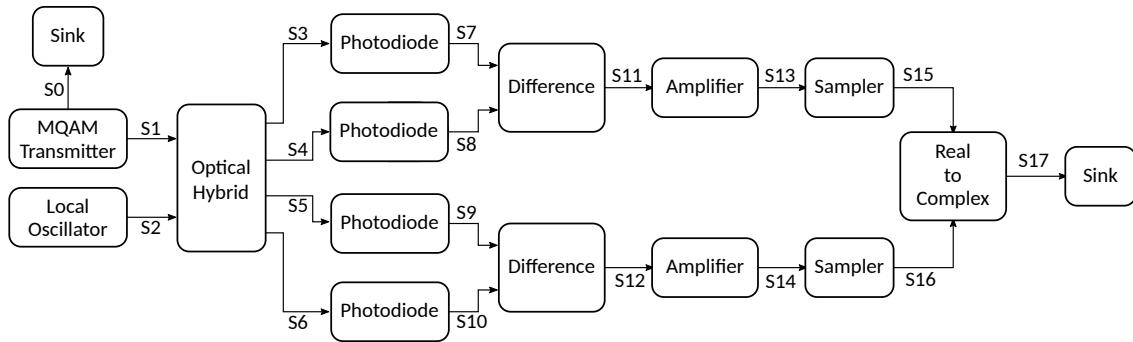


Figure 6.11: Overview of the simulated optical system.

List of signals used in the simulation:

Signal name	Signal type	Status
S0	Binary	check
S1	OpticalSignal	check
S2	OpticalSignal	check
S3	OpticalSignal	check
S4	OpticalSignal	check
S5	OpticalSignal	check
S6	OpticalSignal	check
S7	TimeContinuousAmplitudeContinuousReal	check
S8	TimeContinuousAmplitudeContinuousReal	check
S9	TimeContinuousAmplitudeContinuousReal	check
S10	TimeContinuousAmplitudeContinuousReal	check
S11	TimeContinuousAmplitudeContinuousReal	check
S12	TimeContinuousAmplitudeContinuousReal	check
S13	TimeContinuousAmplitudeContinuousReal	check
S14	TimeContinuousAmplitudeContinuousReal	check
S15	TimeDiscreteAmplitudeContinuousReal	check
S16	TimeDiscreteAmplitudeContinuousReal	check
S17	OpticalSignal	check

This system takes into account the following input parameters:

System Parameters	Default value	Description
localOscillatorPower1	$2.0505 \times 10^{-8} \text{W}$	Sets the optical power, in units of W, of the local oscillator inside the MQAM
localOscillatorPower2	$2.0505 \times 10^{-8} \text{W}$	Sets the optical power, in units of W, of the local oscillator used for Bob's measurements
localOscillatorPhase	0 rad	Sets the initial phase of the local oscillator used in the detection
responsivity	1 A/W	Sets the responsivity of the photodiodes used in the homodyne detectors
iqAmplitudeValues	$\{\{1, 1\}, \{-1, 1\}, \{-1, -1\}, \{1, -1\}\}$	Sets the amplitude of the states used in the MQAM

The simulation setup is represented in figure 6.11. The starting point is the MQAM, which generates random states from the constellation given by the variable iqAmplitudeValues. The output from the generator is received in the Optical Hybrid where it is mixed with a local oscillator, outputting two optical signal pairs. Each pair is converted to currents by two photodiodes, and the same currents are subtracted from each other, originating another current proportional to one of the quadratures of the input state. The other pair suffers the same process, but the resulting subtraction current will be proportional to another quadrature, dephased by  $\pi/2$  relative to the other quadrature.

## Required files

### Header Files

File	Description	Status
netxpto.h	Generic purpose simulator definitions.	check
m_qam_transmitter.h	Outputs a QPSK modulated optical signal.	check
local_oscillator.h	Generates continuous coherent signal.	check
optical_hybrid.h	Mixes the two input signals into four outputs.	check
photodiode.h	Converts an optical signal to a current.	check
difference.h	Outputs the difference between two input signals.	check
ideal_amplifier.h	Performs a perfect amplification of the input signal	check
sampler.h	Samples the input signal.	check
real_to_complex.h	Combines two real input signals into a complex signal	check
sink.h	Closes any unused signals.	check

### Source Files

File	Description	Status
netxpto.cpp	Generic purpose simulator definitions.	check
m_qam_transmitter.cpp	Outputs a QPSK modulated optical signal.	check
local_oscillator.cpp	Generates continuous coherent signal.	check
optical_hybrid.cpp	Mixes the two input signals into four outputs.	check
photodiode.h	Converts an optical signal to a current.	check
difference.h	Ouputs the difference between two input signals.	check
ideal_amplifier.h	Performs a perfect amplification of the input sinal	check
sampler.cpp	Samples the input signal.	check
real_to_complex.cpp	Combines two real input signals into a complex signal	check
sink.cpp	Empties the signal buffer.	check

## Simulation Results

To test the simulated implementation, a series of states  $\{|\phi_i\rangle\}$  were generated and detected, resulting in a series of measurements  $\{(x_i, y_i)\}$ . The simulation result is presented in figure 6.12:

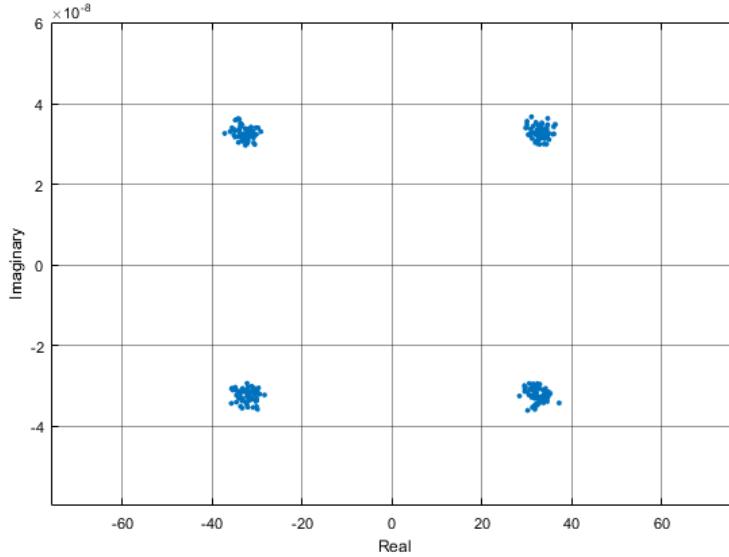


Figure 6.12: Simulation of a constelation of 4 states ( $n = 100$ )

We see that the measurements made groups in certain regions. Each of this groups is centered in the expected value  $(\langle X \rangle, \langle Y \rangle)$  of one the generated states. Also, they show some variance, which was tested for various expected number of photons,  $\langle n \rangle$ , resulting in figure 6.13:

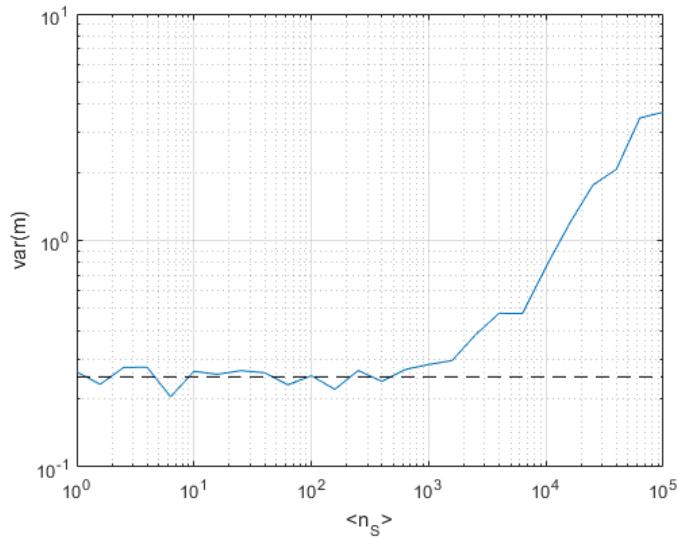


Figure 6.13: Simulation of the variance of  $m$ .  
Local oscillator expected number of photons:  $10^4$

It was expected that the variance should independent of the input's signal number of photons. Plot 6.13 shows that for low values of  $n_S$ , the simulation is in accordance with the theoretical prevision, with  $\text{Var}(X) = \text{Var}(Y) = \frac{1}{4}$ . For large values of  $n_S$ , when the number of photons is about the same has the local oscillator, the quantum noise variance starts to grow proportionally to  $n_S$ , in accordance with the non approximated calculation of quantum noise (eq. ??).

#### Noise Variance with LO power Simulation

The following plot shows the behavior of current noise variance  $\langle(\Delta i)^2\rangle$  with local oscillator power,  $P_{LO}$ :

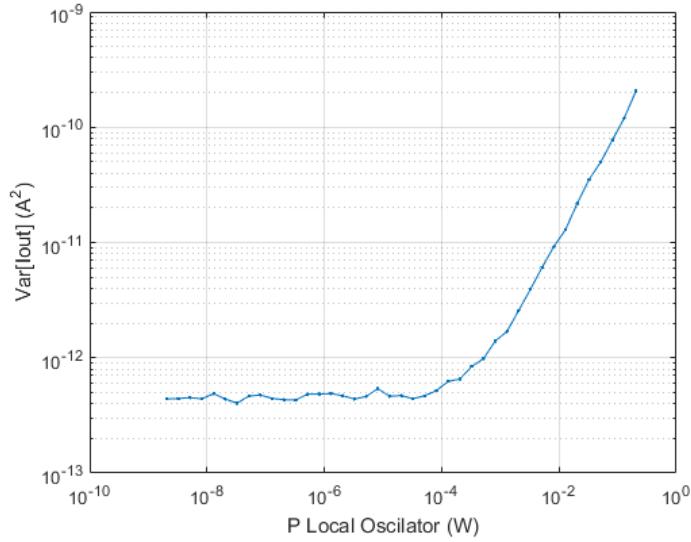


Figure 6.14: Output current variance in function of LO power.

We see that for low LO power, the dominant noise is the thermal contribution, but for higher power, quantum noise dominates, growing proportionally to  $P_{LO}$ . This in accordance with equation 6.45

### 6.2.3 Experimental Analysis

In this section, we will test experimentally the setups discussed in section 6.2.1. This comparison between the theoretical and experimental results will require a high degree of precision from the devices used in these setups. Therefore, the correct characterization of these devices must be the starting point of this experimental phase. One of the most fundamental components is the photodetector, which performs the signal's conversion from the optical domain into the electrical domain.

#### Thorlabs detector

The detector used in the laboratory is the Thorlabs PDB 450C. This detector consists of two well-matched photodiodes and a transimpedance amplifier that generates an output voltage (RF OUTPUT) proportional to the difference between the photocurrents of the photodiodes. Additionally, the unit has two monitor outputs (MONITOR+ and MONITOR-) to observe the optical input power level on each photodiode separately. [5]

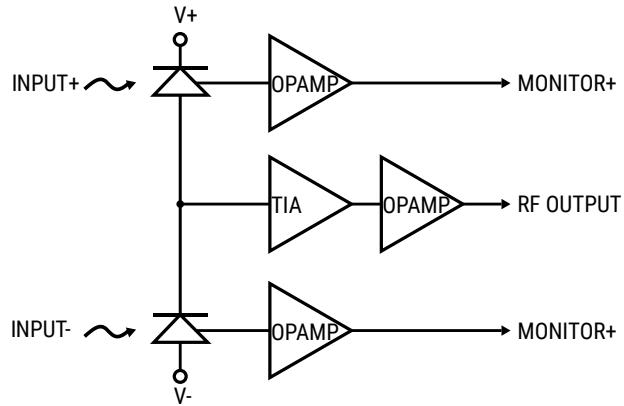


Figure 6.15: Functional block diagram of the PDB 450C Thorlabs detector [5]

Figure 6.15 shows a functional block diagram of the photodetector, in which the TIA (transimpedance amplifier) is an opamp-based current to voltage conversion amplifier. In contrast to the photodiode's non-linear voltage response to incident light, its current response is linear. To take advantage of this linear response, the TIA is used to perform the conversion of the difference of currents between the two photodiodes into a voltage proportional to that same difference. Various of those parameters can be readily extracted from the device's manual, which are presented in the following tables and plots.

Parameter	Value
Max Responsivity	1.0 A/W

Table 6.2: Thorlabs PDB450C PIN parameters

Parameter	Value
Bandwidth (-3dB)	1 MHz
Voltage Gain	10 V/mW @ peak responsivity
Voltage Noise (RMS)	<180 $\mu$ V (RMS)

Table 6.3: Thorlabs PDB450C MONITOR +/- output parameters

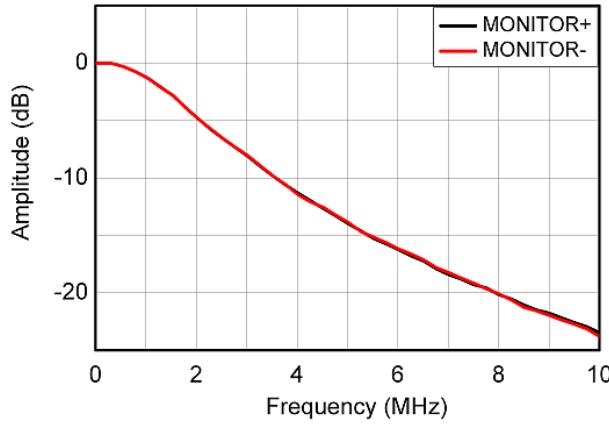


Figure 6.16: MONITOR +/- output response with frequency. [5]

The parameters of the RF OUTPUT have 5 values each, corresponding to the 5 gain settings of the TIA.

Parameter	Values						
Bandwidth (-3dB)	150	45	4	0.3	0.1	0.05	MHz
Transimpedance Gain	$10^3$	$10^4$	$10^5$	$10^6$	$10^7$	$10^8$	V/A
Conversion Gain	$10^3$	$10^4$	$10^5$	$10^6$	$10^7$	$10^8$	V/W
Overall Output Voltage Noise (RMS)	0.50	0.80	1.0	1.1	2.0	3.0	mV

Table 6.4: Thorlabs PDB450C RF OUTPUT parameters

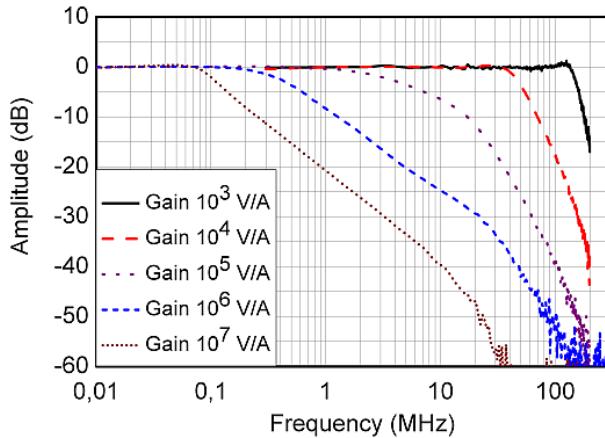


Figure 6.17: RF output response with frequency, for various gain values. [5]

### Data analysis

For each configuration of amplitude and frequency of the input signal, the photodetector output voltage is collected by the Digital Oscilloscope during a time interval and saved in a

data file. The data consists on a sequence of pulses and it's analysis will be focused on the samples with equal phase. The steps will be the following

1. For each phase value, the average and variance of the samples with the same phase is calculated;
2. The representative amplitude and variance for the present configuration will be the obtained from the middle of the maximum plateau.

To confirm the theoretical results obtained in section 6.2.1, two experimental setups will be created. In the first experiment, we will study quantum noise in the single homodyne detection. The experimental setup will be based on the paper [6]. In the second experiment, we will study quantum noise in the double homodyne detection setup, which will be basically an extension of the single homodyne detection setup.

### Single homodyne detection

To keep the experiment simple and avoid extra sources of noise, we will avoid using black boxes which have complicated inner workings, having a preference in using simple components, as shown in fig. 6.18:

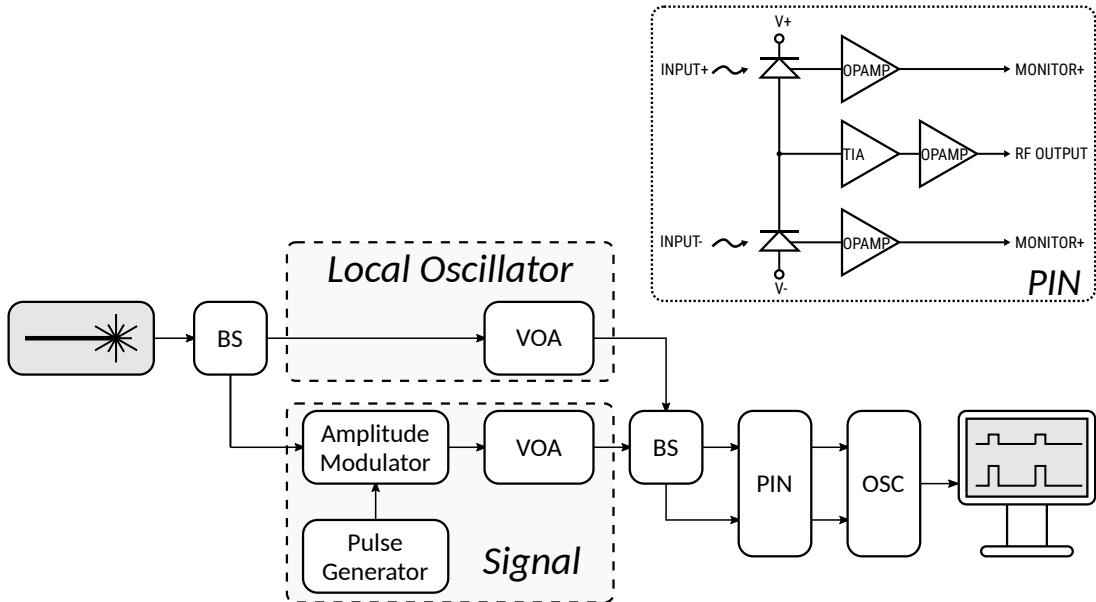


Figure 6.18: Experimental setup

Material list

Device	Description
Local Oscillator	Yenista OSICS Band C/AG
BS	Beam Splitter
Pulse Generator	HP 8116A Pulse Generator
Amplitude Modulator	Mach Zehnder SDL OC 48
VOA	Eigenlicht Power Meter 420
VOA	Thorlabs VOA 45-APC
PIN	Thorlabs PDB 450C
ADC	Picoscope 6403D

A single laser is splitted and used as the source for the signal (S) and the local oscillator (LO). The signal beam is pulsed and highly attenuated. The local oscillator is also attenuated, but not pulsed. The signal and local oscillator interfere in a Beam Splitter originating two beams which are then converted to voltages in the PIN. These voltages are read in the Digital Oscilator (OSC) and collect in the computer. In the post processing phase, the quantum noise is measured by applying a difference between the two beams and measuring it's variance.

The second stage of the experiment will be very similar to the first one, in which the signal and local oscillator branches will be divided. One of the new branches of the local oscillator will suffer a phase delay of  $\pi/2$ , in order to measure the quadrature component of the incoming signal.

#### 6.2.4 Comparative analysis

Given the theoretical, simulated and experimental frameworks, we will now compare the results obtained by each of them.

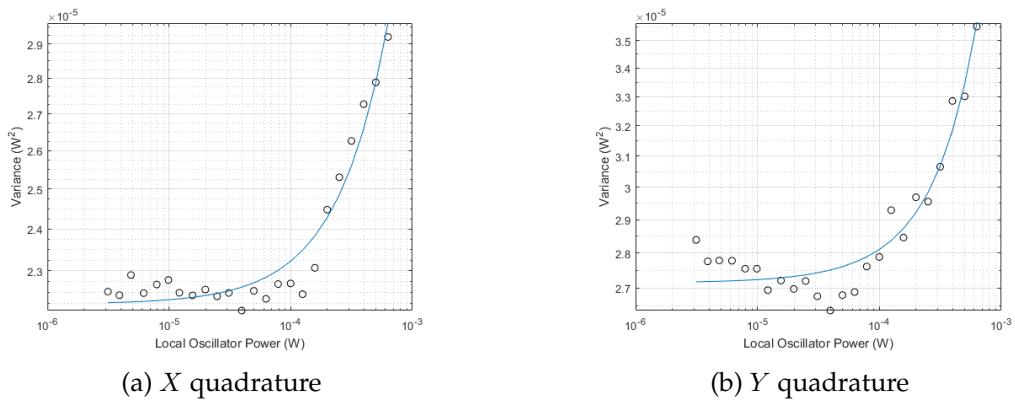


Figure 6.19: Noise variance dependency with local oscillator power for two different quadratures. Experimental vs fitted data.

Figures 6.19a and 6.19b show measurements of total noise for two different quadratures.

For low power of LO, the noise variance fluctuates around a constant value. For high power of LO, ( $P_{LO} > 10^{-4}W$ ), the variance of noise shows an increasing trend roughly proportional to  $P_{LO}^2$ . The polynomial fittings confirm this trend, showing a degree 2 coefficient much larger than the degree 1 coefficient

$$\text{Var}_X = 2.22 \times 10^{-5} + 9.6 \times 10^{-3} P_{LO} + 3.40 P_{LO}^2 \quad (6.51)$$

$$\text{Var}_Y = 2.71 \times 10^{-5} + 8.9 \times 10^{-3} P_{LO} + 7.25 P_{LO}^2 \quad (6.52)$$

The expected growth should be proportional to  $P_{LO}$ , but the RIN noise, originated by the electric apparatus, which grows quadratically with the power, is dominating the noise amplitude for large  $P_{LO}$ .

We see that both the simulation and experimental data display a similar behaviour, but the quadratic growth of noise for large  $P_{LO}$  was not predicted in the simulations.

### 6.2.5 Known problems

## References

- [1] J Shapiro. "Quantum noise and excess noise in optical homodyne and heterodyne receivers". In: *IEEE journal of quantum electronics* 21.3 (1985), pp. 237–250.
- [2] Mark Fox. *Quantum Optics: An Introduction*. Oxford University Press, 2006.
- [3] Rodney Loudon. *The Quantum Theory of Light*. Oxford University Press, 2000.
- [4] Hans-A. Bachor and Timothy C. Ralph. *A Guide to Experiments in Quantum Optics*. Wiley-VCH, 2004.
- [5] Thorlabs. *Thorlabs Balance Amplified Photodetectors: PDB4xx Series Operation Manual*. 2014.
- [6] Yue-Meng Chi et al. "A balanced homodyne detector for high-rate Gaussian-modulated coherent-state quantum key distribution". In: *New Journal of Physics* 13.1 (2011), p. 013003.

### 6.3 BPSK Transmission System

<b>Student Name</b>	:	André Mourato (2018/01/28 - 2018/02/27) Daniel Pereira (2017/09/01 - 2017/11/16)
<b>Goal</b>	:	Estimate the BER in a Binary Phase Shift Keying optical transmission system with additive white Gaussian noise. Comparison with theoretical results.
<b>Directory</b>	:	sdf/bpsk_system

Binary Phase Shift Keying (BPSK) is the simplest form of Phase Shift Keying (PSK), in which binary information is encoded into a two state constellation with the states being separated by a phase shift of  $\pi$  (see Figure 6.235).

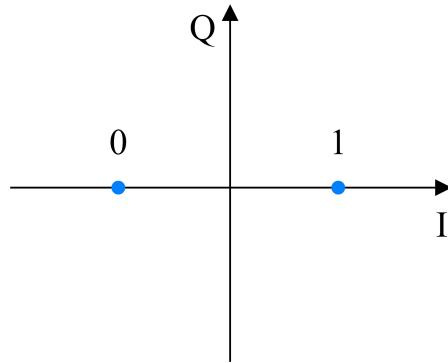


Figure 6.20: BPSK symbol constellation.

White noise is a random signal with equal intensity at all frequencies, having a constant power spectral density. White noise is said to be Gaussian (WGN) if its samples follow a normal distribution with zero mean and a certain variance  $\sigma^2$ . For WGN its spectral density equals its variance. For the purpose of this work, additive WGN is used to model thermal noise at the receivers.

The purpose of this system is to simulate BPSK transmission in back-to-back configuration with additive WGN at the receiver and to perform an accurate estimation of the BER and validate the estimation using theoretical values.

#### 6.3.1 Theoretical Analysis

The output of the system with added gaussian noise follows a normal distribution, whose first probabilistic moment can be readily obtained by knowledge of the optical power of the received signal and local oscillator,

$$m_i = 2\sqrt{P_L P_S} G_{ele} \cos(\Delta\theta_i), \quad (6.53)$$

where  $P_L$  and  $P_S$  are the optical powers, in watts, of the local oscillator and signal, respectively,  $G_{ele}$  is the gain of the trans-impedance amplifier in the coherent receiver and

$\Delta\theta_i$  is the phase difference between the local oscillator and the signal, for BPSK this takes the values  $\pi$  and 0, in which case (6.238) can be reduced to,

$$m_i = (-1)^{i+1} 2 \sqrt{P_L P_S} G_{ele}, \quad i = 0, 1. \quad (6.54)$$

The second moment is directly chosen by inputting the spectral density of the noise  $\sigma^2$ , and thus is known *a priori*.

Both probabilist moments being known, the probability distribution of measurement results is given by a simple normal distribution,

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m_i)^2}{2\sigma^2}}. \quad (6.55)$$

The BER is calculated in the following manner,

$$BER = \frac{1}{2} \int_0^{+\infty} f(x|\Delta\theta = \pi) dx + \frac{1}{2} \int_{-\infty}^0 f(x|\Delta\theta = 0) dx, \quad (6.56)$$

given the symmetry of the system, this can be simplified to,

$$BER = \int_0^{+\infty} f(x|\Delta\theta = \pi) dx = \frac{1}{2} \operatorname{erfc} \left( \frac{-m_0}{\sqrt{2}\sigma} \right) \quad (6.57)$$

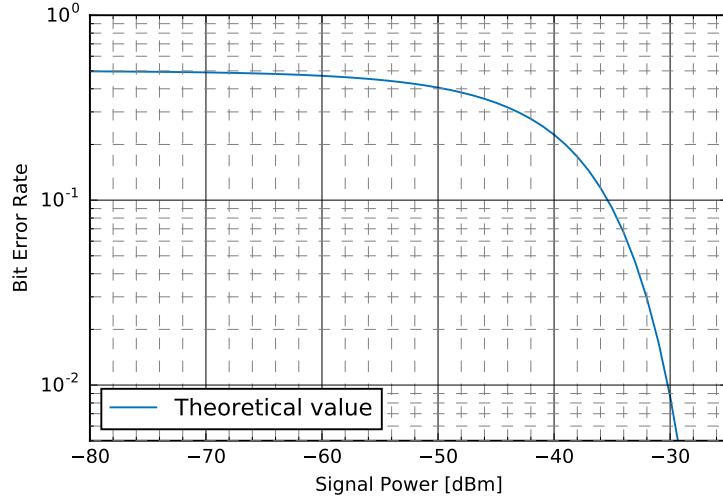


Figure 6.21: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm.

### 6.3.2 Simulation Analysis

A diagram of the system being simulated is presented in the Figure 6.237. A random binary sequence is generated and encoded in an optical signal using BPSK modulation. The decoding of the optical signal is accomplished by an homodyne receiver, which combines the

signal with a local oscillator. The received binary signal is compared with the transmitted binary signal in order to estimate the Bit Error Rate (BER). The simulation is repeated for multiple signal power levels. Each corresponding BER is recorded and plotted against the expectation value.

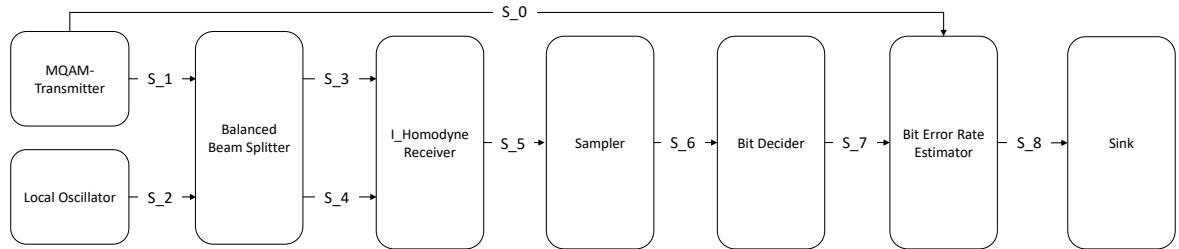


Figure 6.22: Overview of the BPSK system being simulated.

## Required files

Header Files		
File	Comments	Status
add_20171116.h		✓
balanced_beam_splitter_20180124.h		✓
binary_source_20180118.h		✓
bit_decider_20170818.h		✓
bit_error_rate_20171810.h		✓
discrete_to_continuous_time_20180118.h		✓
i_homodyne_receiver_20180124.h		✓
ideal_amplifier_20180118.h		✓
iq_modulator_20180130.h		✓
local_oscillator_20180130.h		✓
m_qam_mapper_20180118.h		✓
m_qam_transmitter_20180118.h		✓
netxpto_20180418.h		✓
photodiode_old_20180118.h		✓
pulse_shaper_20180118.h		✓
sampler_20171116.h		✓
sink_20180118.h		✓
super_block_interface_20180118.h		✓
ti_amplifier_20180102.h		✓
white_noise_20180118.h		✓

Source Files		
File	Comments	Status
add_20171116.cpp		✓
balanced_beam_splitter_20180124.cpp		✓
binary_source_20180118.cpp		✓
bit_decider_20170818.cpp		✓
bit_error_rate_20171810.cpp		✓
discrete_to_continuous_time_20180118.cpp		✓
i_homodyne_receiver_20180124.cpp		✓
ideal_amplifier_20180118.cpp		✓
iq_modulator_20180130.cpp		✓
local_oscillator_20180130.cpp		✓
m_qam_mapper_20180118.cpp		✓
m_qam_transmitter_20180118.cpp		✓
netxpto_20180418.cpp		✓
photodiode_old_20180118.cpp		✓
pulse_shaper_20180118.cpp		✓
sampler_20171116.cpp		✓
sink_20180118.cpp		✓
super_block_interface_20180118.cpp		✓
ti_amplifier_20180102.cpp		✓
white_noise_20180118.cpp		✓

### System Input Parameters

This system takes into account the following input parameters:

System Input Parameters		
Parameter	Default Value	Comments
numberOfBitsReceived	-1	
numberOfBitsGenerated	1000	
samplesPerSymbol	16	
pLength	5	
bitPeriod	$20 \times 10^{-12}$	
rollOffFactor	0.3	
signalOutputPower_dBm	-20	
localOscillatorPower_dBm	0	
localOscillatorPhase	0	
iqAmplitudesValues	$\{ \{-1, 0\}, \{1, 0\} \}$	
transferMatrix	$\{ \{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}\} \}$	
responsivity	1	
amplification	$10^6$	
electricalNoiseAmplitude	$5 \times 10^{-4}\sqrt{2}$	
samplesToSkip	$8 \times \text{samplesPerSymbol}$	
bufferLength	20	
shotNoise	false	

## Inputs

This system takes no inputs.

## Outputs

This system outputs the following objects:

System Output Signals	
Signal	Associated File
Initial Binary String ( $S_0$ )	S0.sgn
Optical Signal with coded Binary String ( $S_1$ )	S1.sgn
Local Oscillator Optical Signal ( $S_2$ )	S2.sgn
Beam Splitter Outputs ( $S_3, S_4$ )	S3.sgn & S4.sgn
Homodyne Receiver Electrical Output ( $S_5$ )	S5.sgn
Sampler Output ( $S_6$ )	S6.sgn
Decoded Binary String ( $S_7$ )	S7.sgn
BER Result String ( $S_8$ )	S8.sgn
Report	Associated File
Bit Error Rate Report	BER.txt

## Bit Error Rate - Simulation Results

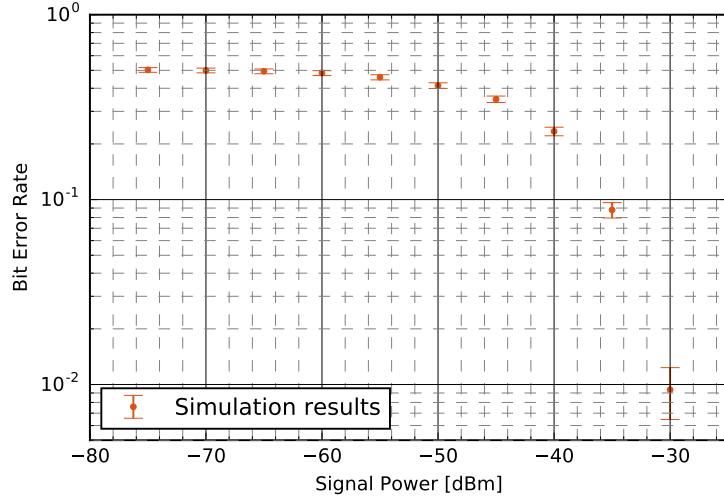


Figure 6.23: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm.

### 6.3.3 Comparative Analysis

The following results show the dependence of the error rate with the signal power assuming a constant Local Oscillator power of 0 dBm, the signal power was evaluated at levels between -70 and -25 dBm, in steps of 5 dBm between each. The simulation results are presented in orange with the computed lower and upper bounds, while the expected value, obtained from (6.242), is presented as a full blue line. A close agreement is observed between the simulation results and the expected value. The noise spectral density was set at  $5 \times 10^{-4}\sqrt{2} \text{ V}^2$  [1].

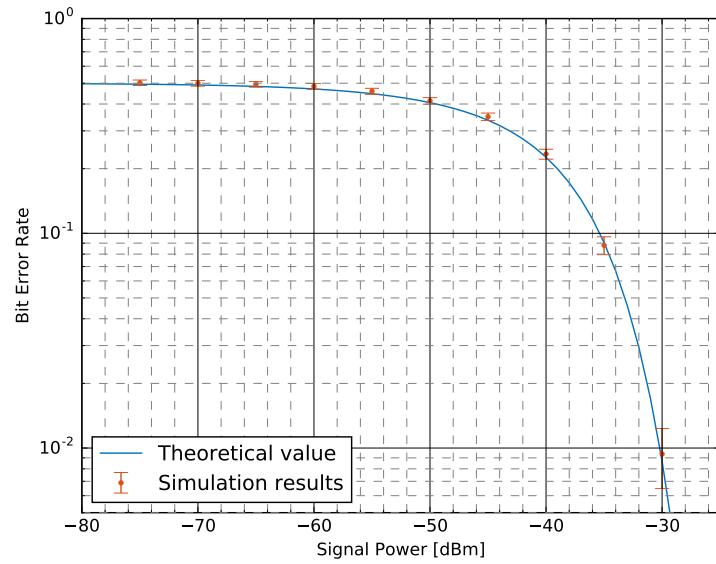


Figure 6.24: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm. Theoretical values are presented as a full blue line while the simulated results are presented as a errorbar plot in orange, with the upper and lower bound computed in accordance with the method described in 7.4

## References

- [1] Thorlabs. *Thorlabs Balance Amplified Photodetectors: PDB4xx Series Operation Manual*. 2014.

## 6.4 M-QAM Transmission System

<b>Student Name</b>	Andoni Santos (2018/01/03 - ) Ana Luisa Carvalho (2017/04/01 - 2017/12/31)
<b>Goal</b>	: M-QAM system implementation with BER measurement and comparison with theoretical and experimental values.
<b>Directory</b>	: sdf/m_qam_system

The goal of this project is to simulate a Quadrature Amplitude Modulation transmission system with  $M$  points in the constellation diagram (M-QAM) and to perform a Bit Error Rate (BER) measurement that can be compared with theoretical and experimental values.

M-QAM systems can encode  $\log_2 M$  bits per symbol which means they can transmit higher data rates keeping the same bandwidth when compared, for example, to PSK systems. However, because the states are closer together, these systems require a higher signal-to-noise ratio. The Bit Error Rate (BER) is a measurement of how a bit stream is altered by a transmission system due to noise (among other factors). To study this effect we introduced Additive White Gaussian Noise (AWGN) to model thermal noise at the receiver.

For  $M = 4$  the M-QAM system can be reduced to a Quadrature Phase Shift Keying system (QPSK) system that uses four equispaced points in the constellation diagram (see figure 6.25).

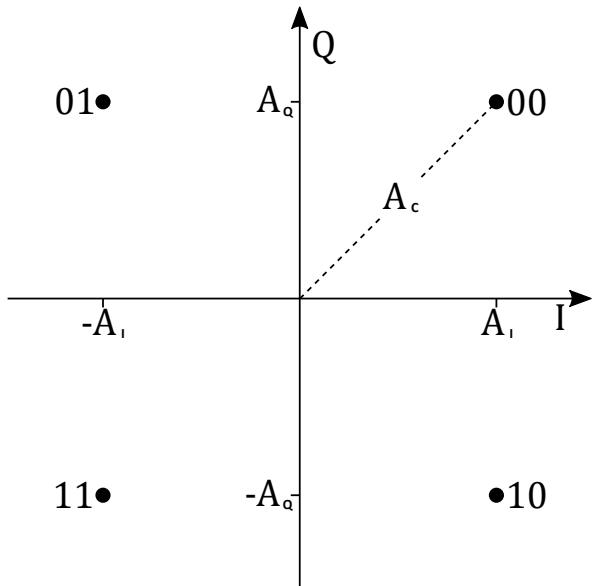


Figure 6.25: 4-QAM constellation points.

### 6.4.1 Theoretical Analysis

M-QAM is a modulation scheme that takes advantage of two sinusoidal carriers with a phase difference of  $\pi/2$ . The resultant output consists of a signal with both amplitude and phase variations. The two carriers, referred to as I (In-phase) and Q (Quadrature), can be represented as

$$I(t) = a(t) \cos(\phi t) \quad (6.58)$$

$$Q(t) = a(t) \sin(\phi t) \quad (6.59)$$

which means that any sinusoidal wave can be decomposed in its I and Q components:

$$A_c \cos(\omega t + \phi) = A (\cos(\omega t) \cos(\phi) - \sin(\omega t) \sin(\phi)) \quad (6.60)$$

$$= A_I \cos(\omega t) - A_Q \sin(\omega t), \quad (6.61)$$

where we have used the expression for the cosine of a sum and the definitions of  $A_I$  and  $A_Q$ .

For the particular case of  $M = 4$ , it can be considered that  $A = A_I = A_Q$ . If it is assumed there is no crosstalk or interference between the Q and I components, the signal can be treated as a pair of independent BPSK systems, one for the in-phase component and another for the quadrature component. Using Gray coding, adjacent symbols differ by only one bit. As such, an error in a single component leads to a single bit error. This means that the probability of an error in bit detection is independent among components, as there is no crosstalk or interference. That being the case, the bit error rate can be calculated for the BPSK case.

Let the  $s(t)$  be the signal to be sampled at a given instant  $t$  for either the in-phase or quadrature component,  $a(t)$  be the component corresponding to the transmitted signal with peak amplitude equal to  $A$ , depending on whether the transmitted signal was 1 or 0, and  $n(t)$  the component associated with the Gaussian white noise with spectral density  $G_n(f) = n_0/2$ , such that:

$$s(t) = a(t) + n(t) \quad (6.62)$$

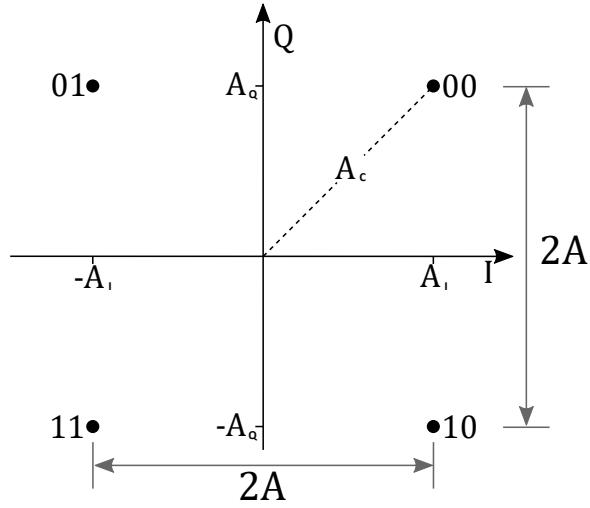


Figure 6.26: The relation between  $A$  and the distance between constellation points.

In this case, assuming the absence of inter-symbol interference,  $s(t)$  at the time of sampling will be a Gaussian random variable with average value of  $\pm A$ , depending on what signal was transmitted, and variance equal to  $N$ , the noise power. In this case, using the constellation from Figure 6.26, with a decision boundary halfway between  $A$  and  $-A$ , an error occurs in two situations: when the a 0 is transmitted but a 1 is identified, or a 1 is transmitted and a 0 is identified. These errors happen when the perturbation in the signal due to noise is enough to push the value beyond the decision boundary. This is illustrated in Figures 6.27 and 6.28, where the probability of error is shown by the colored area under the curve [1].

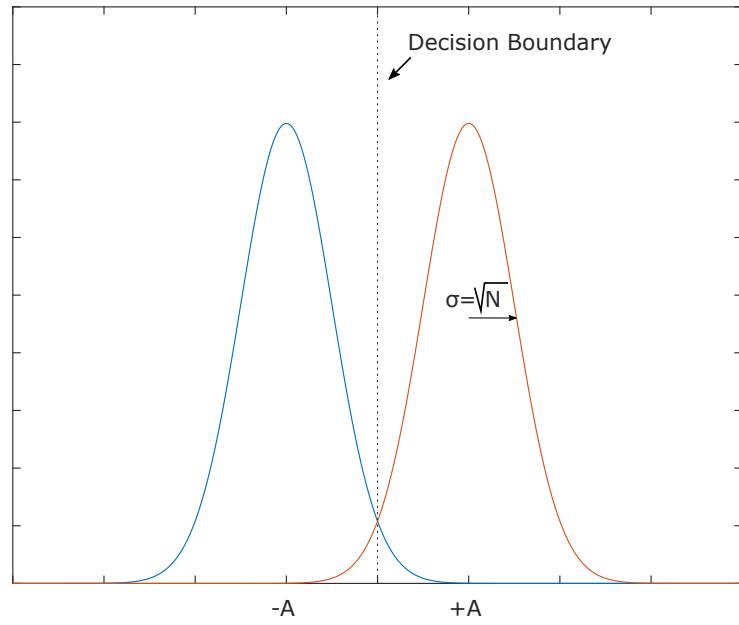


Figure 6.27: Probability density functions for  $s(t) = a(t) + n(t)$ , with  $a(t) = \pm A$  and  $n(t)$  as a Gaussian variable of zero mean and variance  $N$ .

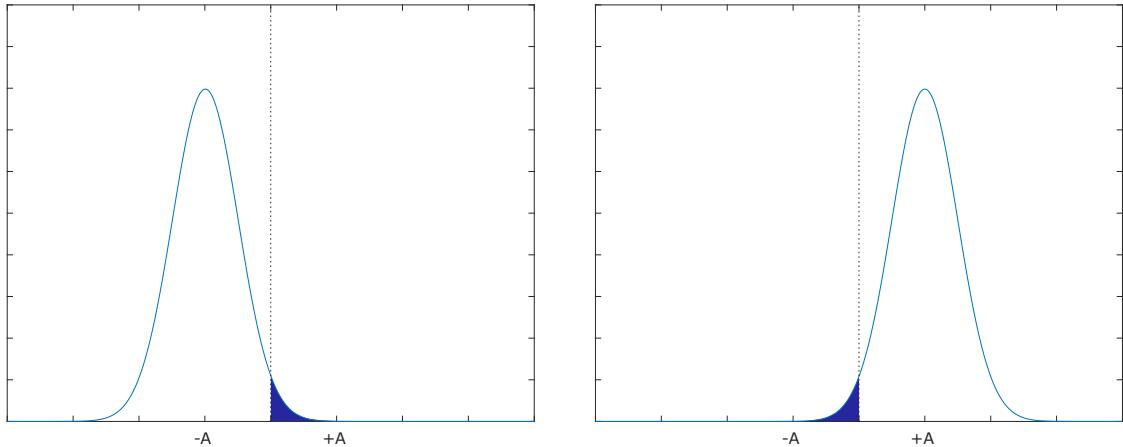


Figure 6.28: The area below the curves represents the probability of error for each transmitted bit.

The probability of bit error can be expressed as:

$$P_{be} = P_0 P_{e0} + P_1 P_{e1} \quad (6.63)$$

With equal probability for both bits, and considering the constellation's symmetry

$$P_{be} = Q\left(\frac{A}{\sqrt{N}}\right) = \frac{1}{2}\operatorname{erfc}\left(\frac{A}{\sqrt{2N}}\right) \quad (6.64)$$

with

$$A = K_{ele}\sqrt{P_L P_S} \quad (6.65)$$

$$N = n_0 B \quad (6.66)$$

where  $P_L$  is the local oscillator power,  $P_S$  is the average optical power of the laser source on which the signal is modulated,  $K_{ele}$  is the transimpedance amplifier's gain,  $n_0/2$  is the noise spectral density and  $B$  is the channel bandwidth.  $A$  is the magnitude of the signal at sampling time and  $\sqrt{N}$  is the RMS noise. Figure 6.26 shows the relation between the constellation points and  $A$ .

The symbol error rate depends on both bits being correctly detected. This probability is:

$$P_C = (1 - P_{be})^2 \quad (6.67)$$

From this, the probability of symbol error is:

$$\begin{aligned} P_s &= 1 - P_C \\ &= 1 - \left(1 - Q\left(\frac{A}{\sqrt{2N}}\right)\right)^2 \\ &= 2Q\left(\frac{A}{\sqrt{2N}}\right) \left[1 - \frac{1}{2}Q\left(\frac{A}{\sqrt{2N}}\right)\right] \end{aligned} \quad (6.68)$$

The peak power signal-to-noise ratio, defined as the ratio between the instantaneous peak signal power to mean noise power, is given by [1]:

$$\text{SNR} = \frac{A^2}{N} \quad (6.69)$$

$N$ , the mean noise power, is the variance of the noise signal. The square root of this value is the peak signal-to-rms-noise ratio, and it is the single parameter on which the probability of errors depends.

$$\text{SNR}_{RMS} = \frac{A}{\sqrt{N}} \quad (6.70)$$

Here,  $A$  is the peak signal amplitude and  $\sqrt{N}$  is the RMS noise in the absence of a signal. Here it is assumed that the electric filter's bandwidth is large enough for the signal not to be affected, thus limiting only the noise power.

It's worth noting that optical noise is being ignored in this analysis. The conditions considered here assume only the presence of electrical noise appearing at the receiver, as shown in figure 6.29. In these conditions, the noise is considered to be independent from the local oscillator. If there was a noise source in the optical domain, this would not be so.

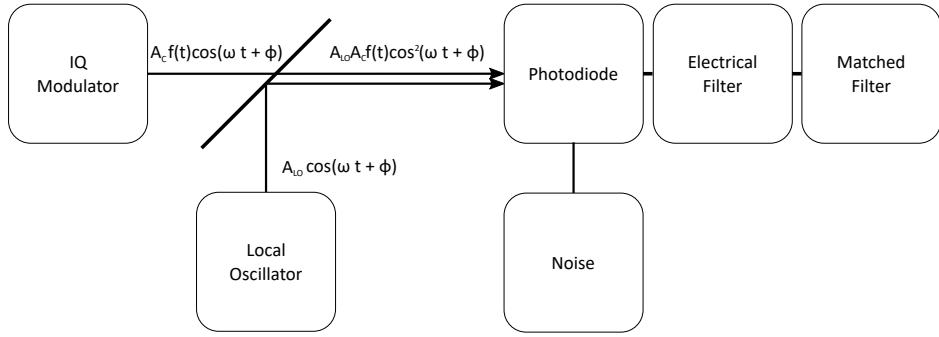


Figure 6.29: Local oscillator and receiver filters diagram.

It is possible to decrease the SNR and, consequently, the error rate reducing the filter's bandwidth and finding an optimum filter. This filter is called a matched filter. The resulting signal will still have Gaussian noise, but the signal-to-noise ratio will be greatly improved. This can be achieved by using a root-raised cosine filter at the pulse shaper and a similar one at the receiver, before the sampler. Inter-symbol interference will still be null as it is equivalent to a raised cosine filter, where half the filtering is done on the transmitter side (while pulse-shaping) and the other half is done on the receiver side, before sampling.

Considering a pulse with composition similar to the signal described by Equation 6.62, let  $a(t) = A_p p(t)$ , where  $A_p$  is the peak amplitude of the signal and  $p(t)$  is the shape of the pulse, and  $n(t)$  be AWGN of spectral density  $G_n(f) = \frac{n_0}{2}$ . Also, let  $H(f)$  be the frequency domain response of the filter at the receiver. The filter is similar to the shape of the pulse, but reversed in time and shifted by a time delay, such that it maximizes the SNR [2].

The energy contained in the pulse that enters the receiver filter depends on the pulse amplitude and on its shape, and it is given by:

$$E_p = \int_{-\infty}^{\infty} |A(f)|^2 df = A_p^2 \int_{-\infty}^{\infty} |P(f)|^2 df \quad (6.71)$$

The amplitude of the signal component after the receiver filter  $H(f)$ , at a given sampling time  $t = t_0 + t_d$ , is:

$$A_o = \mathfrak{F}^{-1} [H(f)A(f)] \big|_{t=t_0+t_d} = A_p \int_{-\infty}^{+\infty} H(f)P(f)e^{+j\omega t_d} df \quad (6.72)$$

Similarly, the noise power at the receiver filter output is given by:

$$N_o = \int_{-\infty}^{+\infty} |H(f)|^2 G_n(f) df = \frac{n_0}{2} \int_{-\infty}^{+\infty} |H(f)|^2 df \quad (6.73)$$

This means that the peak signal power to mean noise power ratio at the filter output is given by:

$$\begin{aligned}\frac{A_o^2}{N_o} &= A_p^2 \frac{\left| \int_{-\infty}^{+\infty} |H(f)P(f)e^{j\omega t_d} df \right|^2}{\int_{-\infty}^{+\infty} |H(f)|^2 G_n(f) df} \\ &= A_p^2 \frac{\left| \int_{-\infty}^{+\infty} |H(f)P(f)e^{j\omega t_d} df \right|^2}{\frac{n_0}{2} \int_{-\infty}^{+\infty} |H(f)|^2 df}\end{aligned}\quad (6.74)$$

It can be shown that a matched filter maximizes the ratio above, so that it becomes [2] :

$$\frac{A_o^2}{N_o} = \frac{A_p^2}{n_0/2} \int_{-\infty}^{+\infty} |P(f)|^2 df = \frac{A_p^2}{n_0/2} \int_{-\infty}^{+\infty} p(t)^2 dt \quad (6.75)$$

Substituting from equation 6.71:

$$\frac{A_o^2}{N_o} = \frac{2E_p}{n_0} \quad (6.76)$$

This shows that the SNR and, therefore, the error probability are dependent on the energy of each symbol and the noise spectral density. However, even though the relation does not directly depend on the pulse shape, the energy of each symbol still depends on the pulse shape. The energy of the symbol results from an integration over the symbol period.

The BER expression for matched filtering becomes:

$$P_{be} = Q \left( \sqrt{\frac{2E_p}{n_0}} \right) = \frac{1}{2} \operatorname{erfc} \left( \sqrt{\frac{E_p}{n_0}} \right) \quad (6.77)$$

To exemplify with a simple case, let the pulse shape be rectangular with period  $T_s$ , such as:

$$a(t) = \begin{cases} A & 0 \leq t \leq T_s \\ 0 & t > T_s \end{cases}$$

The energy of a pulse at the receptor input will be given by:

$$E_p = \int_0^{T_s} a(t)^2 \cos^2(\omega_0 t) dt \quad (6.78)$$

If  $\omega_0$  is such that  $\omega_0 T_s \gg 1$ , as it typically is when modulating optical signals,  $\cos^2(\omega_0 t) \doteq 1/2$ . Then:

$$E_p \doteq \frac{1}{2} \int_0^{T_s} a(t)^2 dt = \frac{1}{2} \int_0^{T_s} A^2 p(t)^2 dt = \frac{1}{2} A^2 \int_0^{T_s} p(t)^2 dt = \frac{A^2 T_s}{2} \quad (6.79)$$

Substituting from equations 6.66 and 6.79 into Equation 6.77, the BER expression becomes:

$$P_{be} = \frac{1}{2} \operatorname{erfc} \left( \sqrt{\frac{A^2 T_s / 2}{N/B}} \right) = \frac{1}{2} \operatorname{erfc} \left( \frac{A}{\sqrt{2N}} \sqrt{T_s B} \right) = \frac{1}{2} \operatorname{erfc} \left( \frac{A}{\sqrt{2N}} \sqrt{\frac{B}{R_s}} \right) \quad (6.80)$$

Here,  $R_s$  is the symbol rate, defined as  $1/T_s$ . Note that  $N$  here is the noise power at the input of the matched filter, as is the bandwidth  $B$  and the peak amplitude  $A$ . The expressions for the Bit-Error-Rate in the cases with and without matched filtering:

Without matched filter	With Matched Filter
$\frac{1}{2}\operatorname{erfc}\left(\frac{A}{\sqrt{2N}}\right)$	$\frac{1}{2}\operatorname{erfc}\left(\frac{A}{\sqrt{2N}}\sqrt{\frac{B}{R_s}}\right)$

Table 6.5: Comparison between the BER equations for the cases with and without matched filtering.

Figure 6.30 show the curves for both these equations, with the parameters listed in Table 6.6.

Curve Parameters	
Parameter	Default Value
bandwidth	64 GHz
symbolRate	4 GBd, 16GBd
outputOpticalPower	Variable
localOscillatorPower	0 dBm
responsivity	1
amplification	$10^3$
noisePower	$10^{-6} \text{ V}^2$
confidence	0.95
theoreticalSNR (before matched filter)	0 dB

Table 6.6: Parameters for the theoretical curves in Figure 6.30.

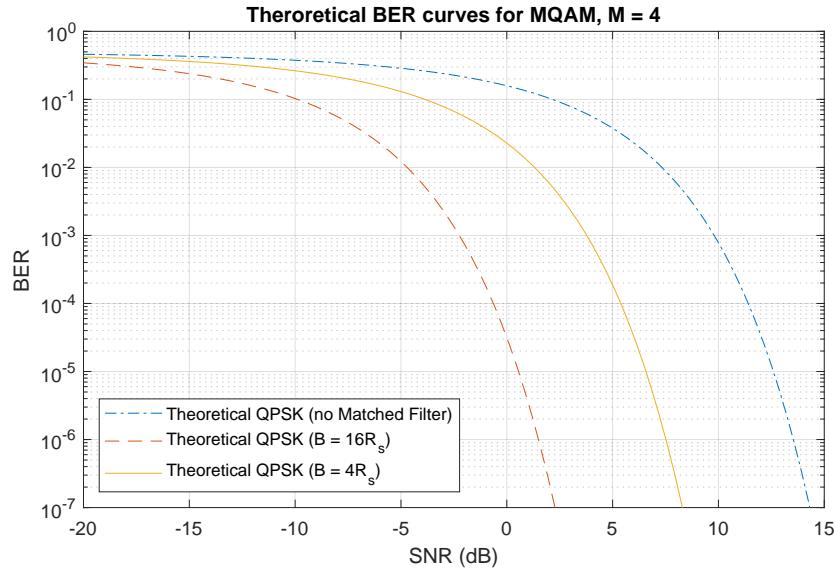


Figure 6.30: QPSK theoretical BER values as a function of the output optical power in dBm.

Figure 6.30 shows the two theoretical curves for QPSK. The blue one is for QPSK without matched filtering and the red one is using root-raised-cosine for matched filtering, which provides optimum transmission. As the use of root-raised-cosine for matched filtering maximizes the signal-to-noise ratio to its optimal value, it allows achieving the same BER with much lower optical power. On the blue curve, on the other hand, the sampling is affected by the full effect of the random noise, as there is no filtering at the receiver. Because of this, a higher optical power is required to achieve a similar BER.

It's worth noting that these equations are only valid for  $M=4$ , as in that case the system is similar to QPSK with a 4 point constellation. For  $M > 4$  a different approach is required.

The use of matched filtering should allow transmission with a lower SNR to achieve the same results as a similar system, even when using a shape with no inter-symbol interference. This is due to the second filter reducing the noise impact before detection, while not causing inter-symbol interference or degradation of the signal data.

#### 6.4.2 Simulation Analysis

The M-QAM transmission system is a complex block of code that simulates the modulation, transmission and demodulation of an optical signal using M-QAM modulation. It is composed of four blocks: a transmitter, a receiver, a sink and a block that performs a Bit Error Rate (BER) measurement. The schematic representation of the system is presented in Figures 6.31 to 6.33.

**Current state:** The system currently being implemented is a QPSK system ( $M=4$ ).

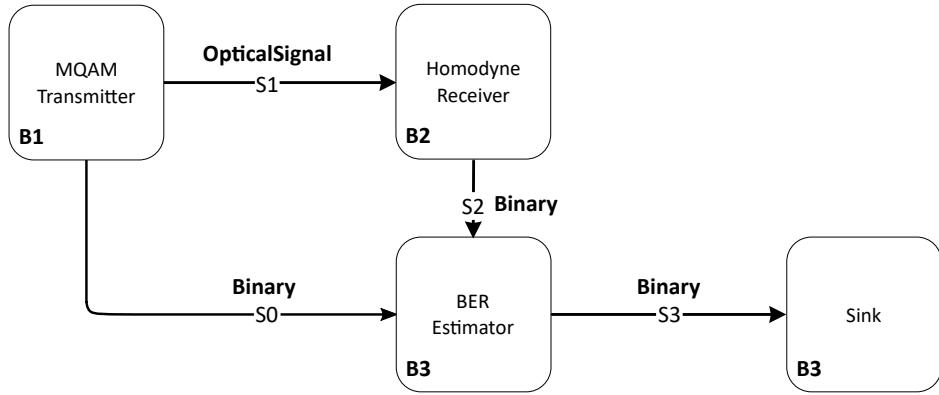


Figure 6.31: Schematic representation of the MQAM system.

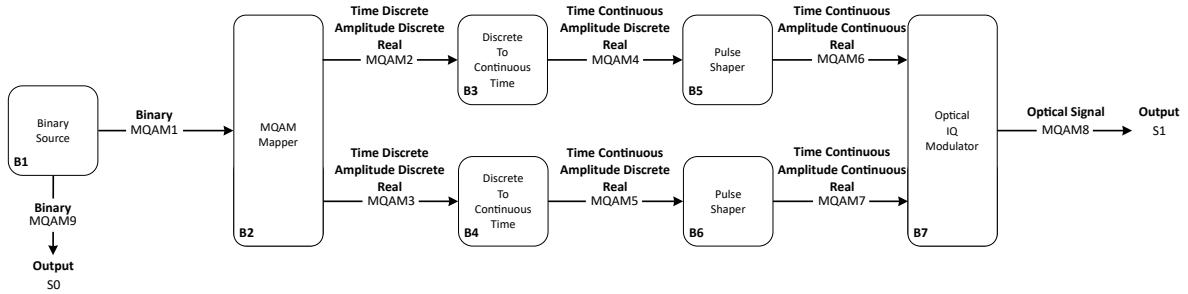


Figure 6.32: Schematic representation of the MQAM Transmitter block.

**Future work:** Extend this block to include other values of M.

### Functional description

A complete description of the M-QAM transmitter and M-QAM homodyne receiver blocks can be found in the *Library* chapter of this document as well as a detailed description of the independent blocks that compose these blocks. The M-QAM transmitter generates one or two optical signals by encoding a binary string using M-QAM modulation. It also outputs a binary signal that is used to perform the BER measurement. The M-QAM homodyne receiver accepts one input optical signal and outputs a binary signal. It performs the M-QAM demodulation of the input signal by combining the optical signal with a local oscillator. The demodulated optical signal is compared to the binary one produced by the transmitter in order to estimate the Bit Error Rate (BER). The files used are summarized in tables 6.7 and 6.8. These include all blocks and sub-blocks used and allow for the full operation of the M-QAM system.

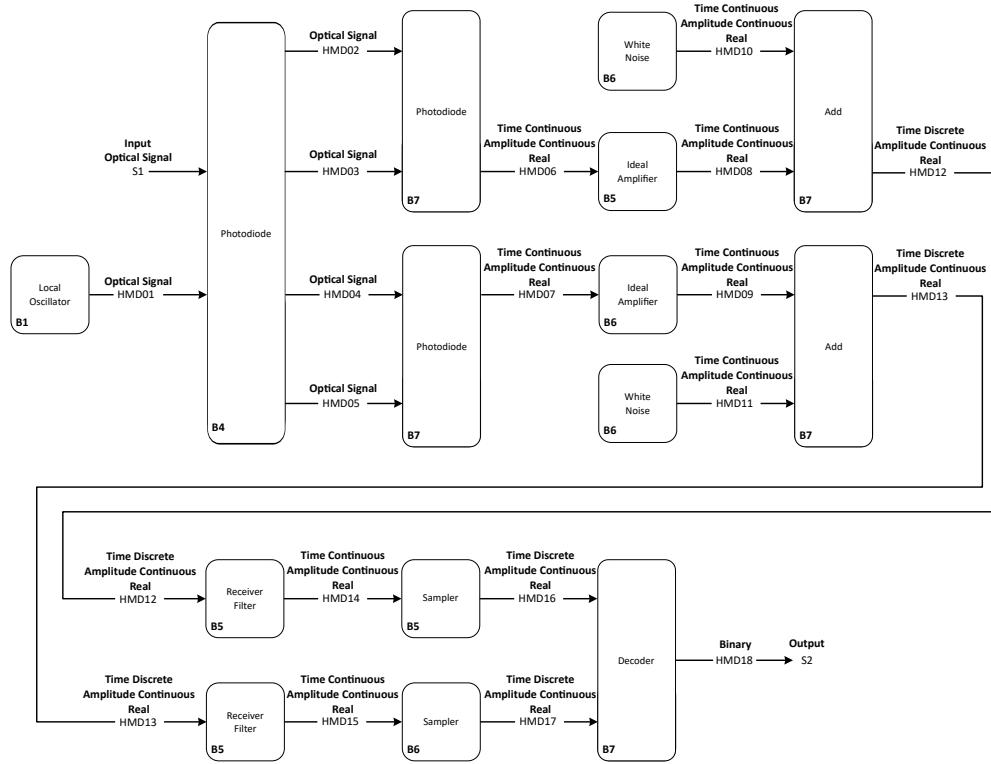


Figure 6.33: Schematic representation of the Homodyne Receiver block.

## Required files

Source Files		
File	Comments	Status
add_20171116.cpp		✓
binary_source_20180118.cpp		✓
bit_error_rate_20171810.cpp		✓
decoder_20180118.cpp		✓
discrete_to_continuous_time_20180118.cpp		✓
homodyne_receiver_20171203.cpp	1	✓
ideal_amplifier_20180118.cpp		✓
iq_modulator_20180118.cpp		✓
local_oscillator_20180118.cpp		✓
m_qam_mapper_20180118.cpp		✓
m_qam_system.cpp	2	✓
m_qam_transmitter_20180118.cpp		✓
netxpto_20180118.cpp	2	✓
optical_hybrid_20180118.cpp		✓
photodiode_old_20180118.cpp		✓
pulse_shaper_20180118.cpp		✓
sampler_20171119.cpp		✓
sink_20180118.cpp		✓
super_block_interface_20180118.cpp	2	✓
white_noise_20180118.cpp		✓

Table 6.7: <sup>1</sup> The library entry is under a different name, *m\_qam\_receiver*;

<sup>2</sup> No library entry as it is a main or general purpose file, not a specific block.

Header Files		
File	Comments	Status
add_20171116.h		✓
binary_source_20180118.h		✓
bit_error_rate_20171810.h		✓
decoder_20180118.h		✓
discrete_to_continuous_time_20180118.h		✓
homodyne_receiver_20171203.h	<sup>1</sup>	✓
ideal_amplifier_20180118.h		✓
iq_modulator_20180118.h		✓
local_oscillator_20180118.h		✓
m_qam_mapper_20180118.h		✓
m_qam_transmitter_20180118.h		✓
netxpto_20180118.h	<sup>2</sup>	✓
optical_hybrid_20180118.h		✓
photodiode_old_20180118.h		✓
pulse_shaper_20180118.h		✓
sampler_20171119.h		✓
sink_20180118.h		✓
super_block_interface_20180118.h	<sup>2</sup>	✓
white_noise_20180118.h		✓

Table 6.8: <sup>1</sup> The library entry is under a different name, *m\_qam\_receiver*

<sup>2</sup> No library entry as it is a main or general purpose file, not a specific block.

## Input Parameters

Table 6.9: Input parameters

Parameter	Type	Description
numberOfBitsGenerated	t_integer	Determines the number of bits to be generated by the binary source
samplesPerSymbol	t_integer	Number of samples per symbol
prbsPatternLength	int	Determines the length of the pseudorandom sequence pattern (used only when the binary source is operated in <i>PseudoRandom</i> mode)
bitPeriod	t_real	Temporal interval occupied by one bit
rollOffFactor_shp	t_real	Parameter of the pulse shaper filter
rollOffFactor_out	t_real	Parameter of the output filter
shaperFilter	enum	Type of filter used in Pulse Shaper
outputFilter	enum	Type of filter used in output filter
seedType	enum	Method of seeding noise generators
seedArray	array<int,2>	Seeds to initialize noise generators
signalOutputPower_dBm	t_real	Determines the power of the output optical signal in dBm
numberOfBitsReceived	int	Determines when the simulation should stop. If -1 then it only stops when there is no more bits to be sent
iqAmplitudeValues	vector<t_iqValues>	Determines the constellation used to encode the signal in IQ space
symbolPeriod	double	Given by bitPeriod / samplesPerSymbol
localOscillatorPower_dBm	t_real	Power of the local oscillator
responsivity	t_real	Responsivity of the photodiodes (1 corresponds to having all optical power transformed into electrical current)
amplification	t_real	Amplification provided by the ideal amplifier
noisePower	t_real	Average power (and variance) of the white noise
samplesToSkip	t_integer	Number of samples to be skipped by the <i>sampler</i> block
confidence	t_real	Determines the confidence limits for the BER estimation
midReportSize	t_integer	
bufferLength	t_integer	Corresponds to the number of samples that can be processed in each run of the system

## Simulation results

In this section we show results obtained through the simulations. The following sections show the eye diagrams of the signal obtained at three different points in the system: the optical signal S1, the signals after amplifying the electrical signal and adding the Gaussian white noise HMD12 and HMD13, and the signal after the filter in the receiver. These eye diagrams will be shown for a variety of system configurations, with varying noise and filters, displaying the advantages of using matched filtering with an optimal filter.

### Without Noise

These section presents the mentioned eye diagrams in configurations without any noise added to the signal. This allows studying the effects of inter-symbol interference caused only by the filters used at the pulse-shaper and at the receiver,

Figure 6.34 shows the eye diagrams for the S1 optical signals for two different values of the output optical power. These were obtained using a raised-cosine filter as a pulse shaper, with a roll-off factor of 0.9. It can be seen that no inter-symbol interference is present.

Curve Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-120 dBm (left), -60 dBm (right)
shaperFilter	RaisedCosine
rollOffFactor	0.9

### Raised-Cosine Signal (roll-off=0.9) At the Transmitter Output

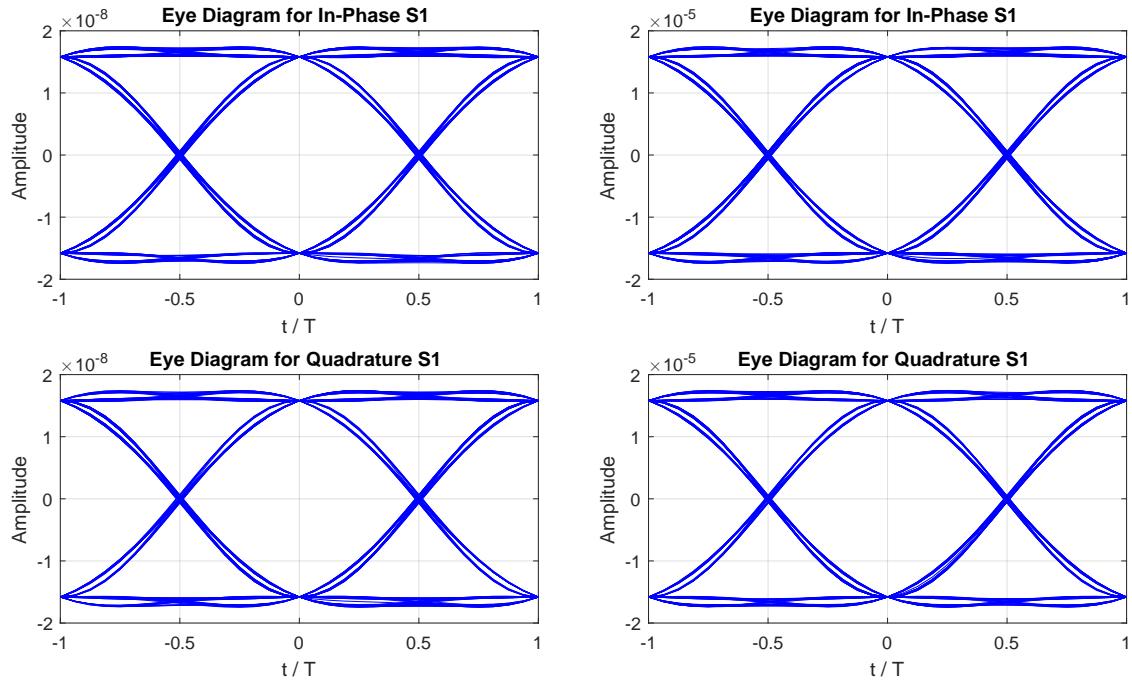


Figure 6.34

As mentioned previously, using matched filters is highly beneficial, as it allows achieving much lower error rates with the same optical power.

Figure 6.35 shows the eye diagrams of the signal at the three mentioned points, for a system without any added white noise, while using matched filtering. The filter used in this case is a raised cosine filter with a roll-off factor of 0.9. Although this is the ideal filter to use for pulse shaping, as it does not cause inter-symbol-interference, it can be seen that when used twice its results are less than ideal, as shown in the eye diagram captured after the second filter.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RaisedCosine
outputFilter	RaisedCosine
rollOffFactor	0.9
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	0 V <sup>2</sup>

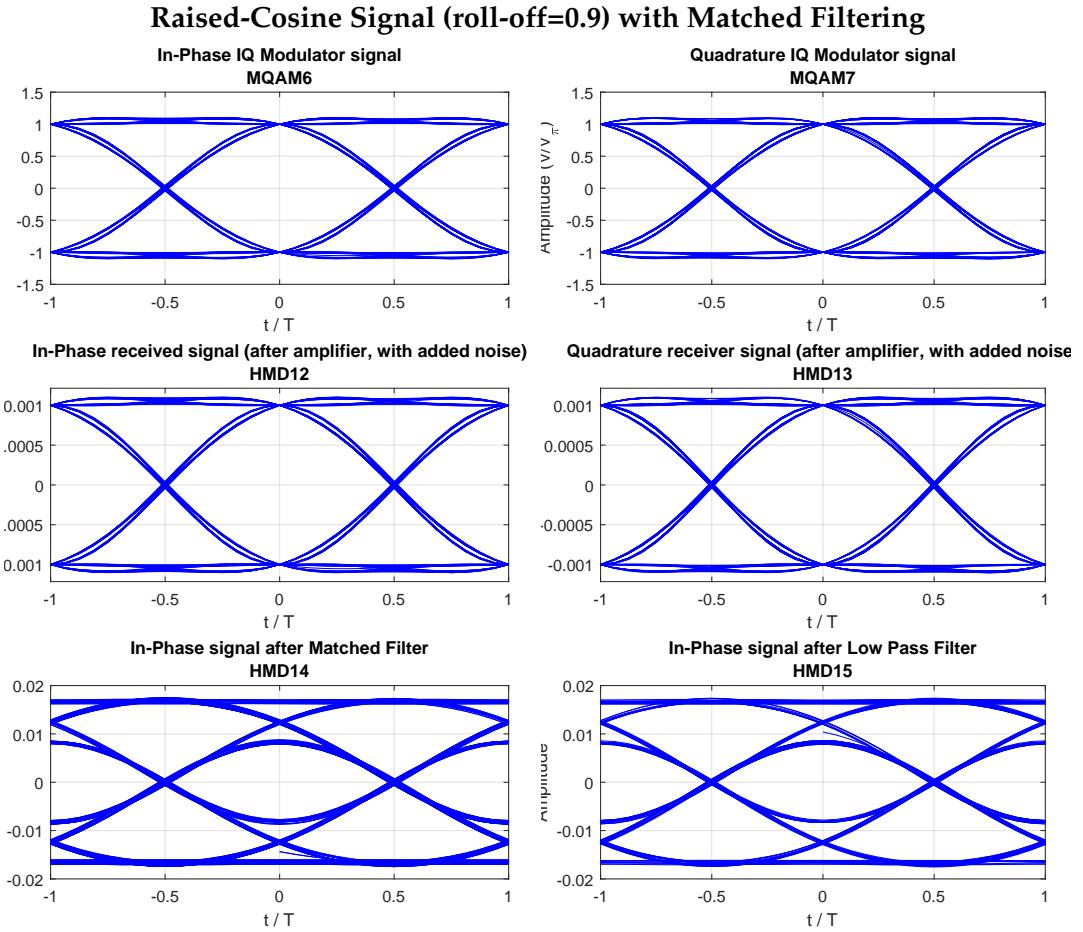


Figure 6.35: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

The optimum solution to achieving no inter-symbol interference while using matched

filtering is to use a root-raised-cosine to do the pulse-shaping at the transmitter and the filtering at the receiver. The corresponding output of applying twice a root-raised-cosine is exactly the same as using a raised-cosine once. As such, the end result suffers from no inter-symbol interference while reaping the benefits of optimum matched filtering. Figure 6.36 shows the eye diagrams when using root-raised-cosine filter both in the transmitter's pulse-shaper and at the receiver's filter. The roll-off factor used in both was 0.9. It can be seen that the shape of the eye diagram is equal to that of Figure 6.34, which uses a single raised cosine filter at the pulse-shaper. Thus, it shows no sign of inter-symbol interference.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.9
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	0 V <sup>2</sup>

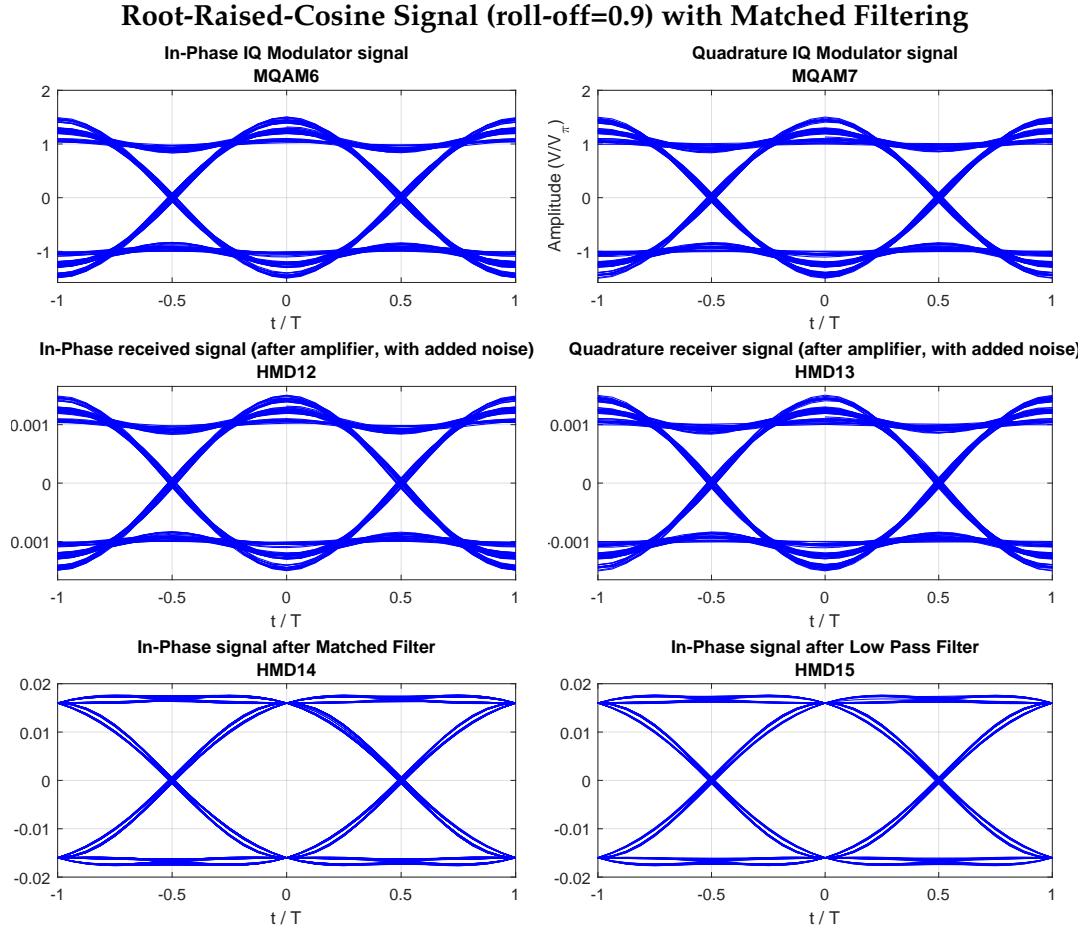


Figure 6.36: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

Figures 6.38 and 6.38 show a similar comparison between matched filtering using raised-cosine or root-raised-cosine filters, but with a roll-off factor of 0.3. Again, it can be seen that the final shape of the eye diagram when using the root-raised-cosine for matched filtering is the same as the shape of the optical signal S1 when using a raised-cosine-filter.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RaisedCosine
outputFilter	RaisedCosine
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	0 V <sup>2</sup>

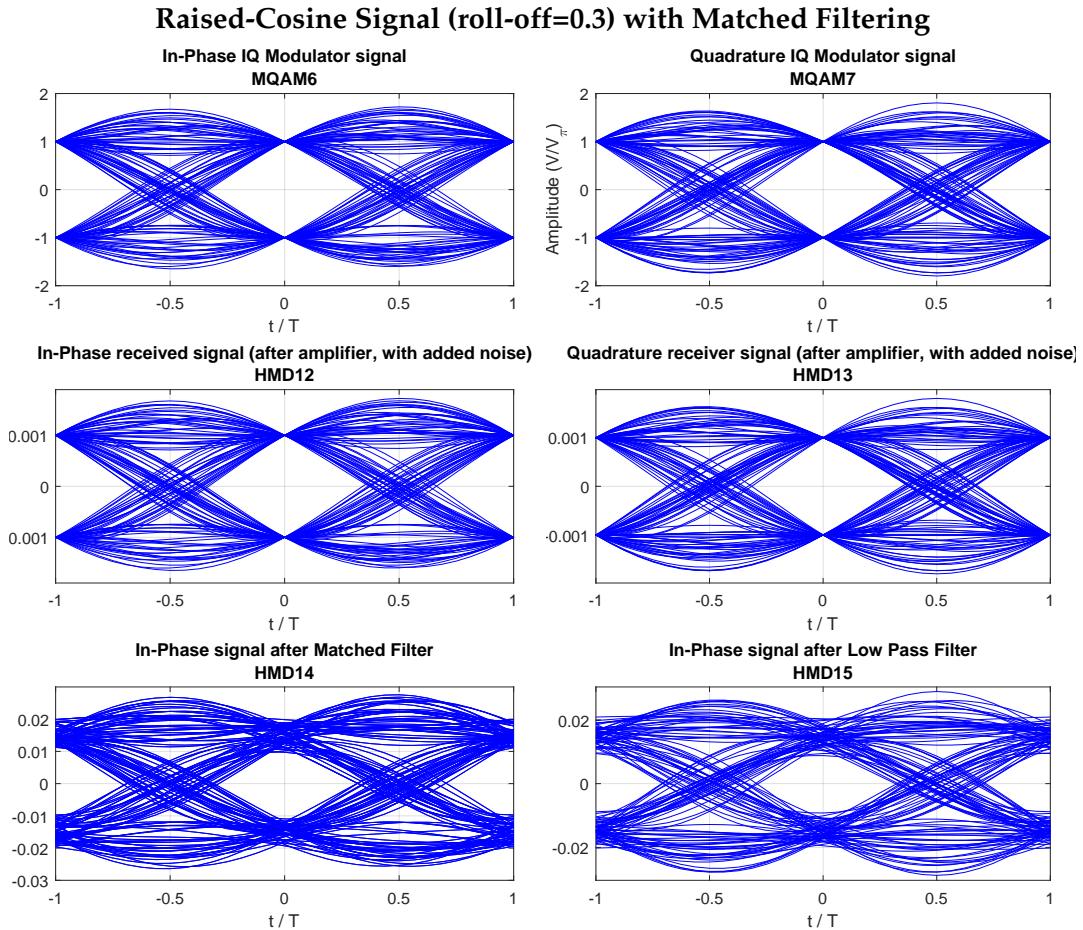


Figure 6.37: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	0 V <sup>2</sup>

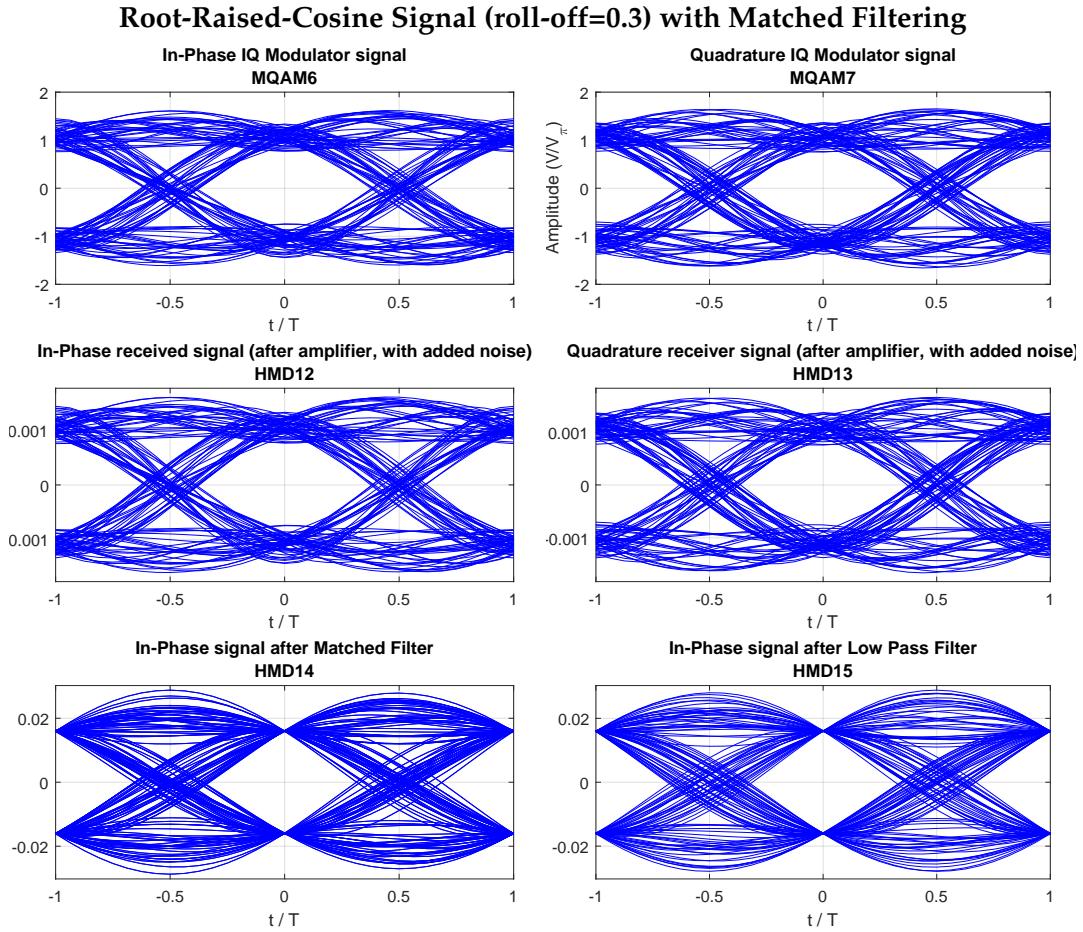


Figure 6.38: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

Thus, it can be concluded that, in order to avoid inter-symbol interference, the filters

used should be raised-cosine or root-raised-cosine, if not using a filter at the receiver or if using matched filtering, respectively. As such, from now on only these configurations will be used.

### With Noise (High SNR)

In this section and the following one, a comparison will be presented between not using a filter on the receiver and using matched filtering. This comparison will be made for signals affected by added white gaussian noise, where the noise is added to the signal after the amplifier stage and before the signal passes through the filter on the receiver.

For the first case, where no filter is present at the receiver, a raised-cosine filter will be used at the pulse shaper. For matched filtering, a root-raised-cosine filter will be used at the pulse-shaper and the receiver.

Figures 6.39-6.40 show the eye diagrams for both these cases. The optical power used was  $-45 \text{ dBm}$ , the noise power was set at  $10^{-6} \text{ V}^2$ , and the roll-off factor was set to 0.9 in both cases. In both cases, it is still possible to visibly see the approximate shape of the signal after noise is added, even without matched filtering. However, it can be seen that the output signal in the case with matched filtering is much less affected by noise, as the root-raised-cosine at the receiver is rather effective at filtering the noise.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9 \text{ s}$
samplesPerSymbol	16
outputOpticalPower	$-45 \text{ dBm}$
shaperFilter	RaisedCosine
outputFilter	
rollOffFactor	0.9
localOscillatorPower	0 $\text{dBm}$
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6} \text{ V}^2$
theoreticalSNR	15 $\text{dB}$
numericalSNR	13.9182 $\text{dB}$
numericalSNR Upper Bound (95% confidence)	13.9624 $\text{dB}$
numericalSNR Lower Bound (95% confidence)	13.8737 $\text{dB}$

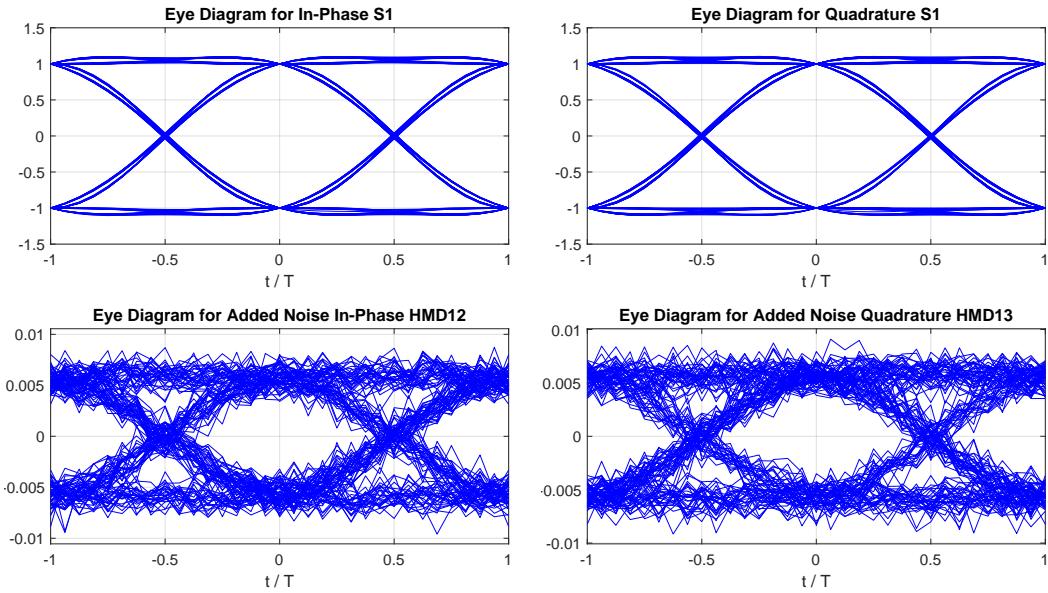
**Raised-Cosine Signal (roll-off=0.9) with Added Noise, SNR = 15 dB**


Figure 6.39: Obtained at two different points in the system: optical output of transmitter on the top and the noisy signal at the bottom.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-45 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.9
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	15 dB
numericalSNR	14.9902 dB
numericalSNR Upper Bound (95% confidence)	15.0408 dB
numericalSNR Lower Bound (95% confidence)	14.9391 dB

**Root-Raised-Cosine Signal (roll-off=0.9) with Added Noise and Matched Filtering,  
SNR = 15**

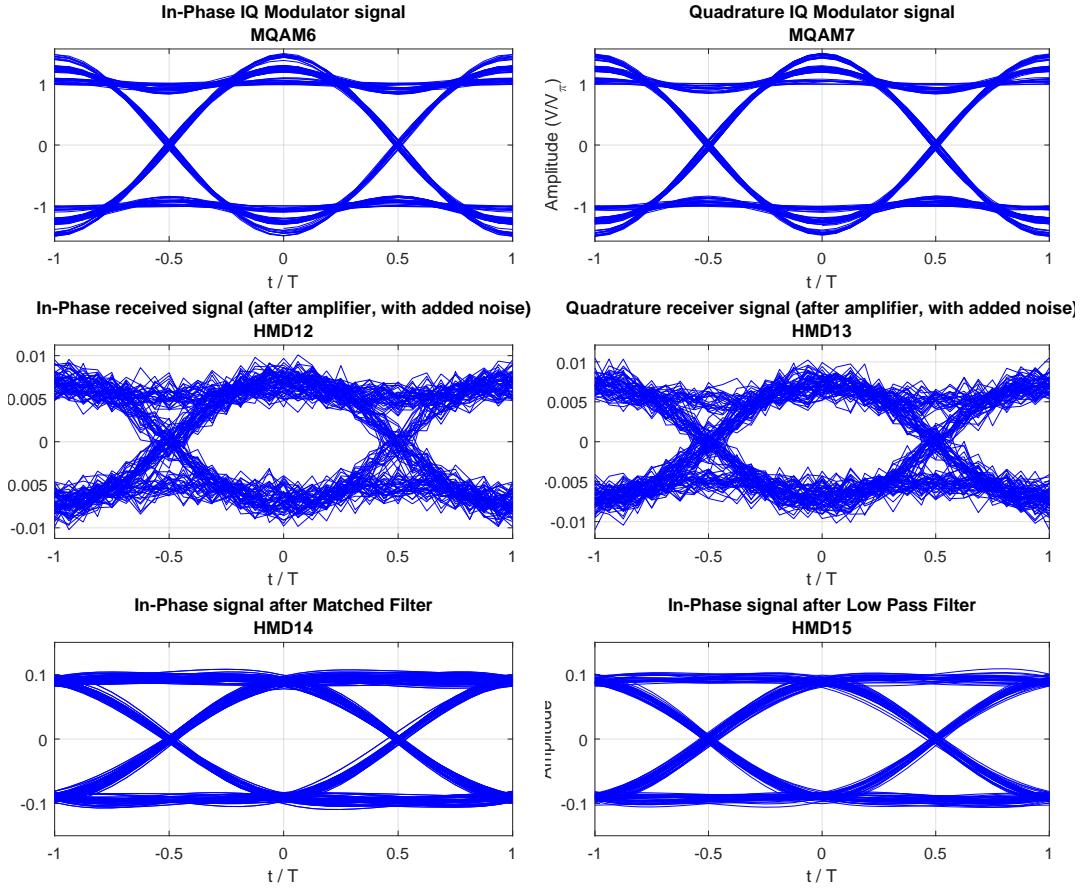


Figure 6.40: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

Figures 6.41 and 6.42 show the cases described above, but with a roll-off factor of 0.3. It can be seen that the case is in all aspects similar to the one presented above.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-45 dBm
shaperFilter	RaisedCosine
outputFilter	
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	15 dB
numericalSNR	14.6827 dB
numericalSNR Upper Bound (95% confidence)	14.7321 dB
numericalSNR Lower Bound (95% confidence)	14.6327 dB

**Raised-Cosine Signal (roll-off=0.3) with Added Noise, SNR = 15 dB**

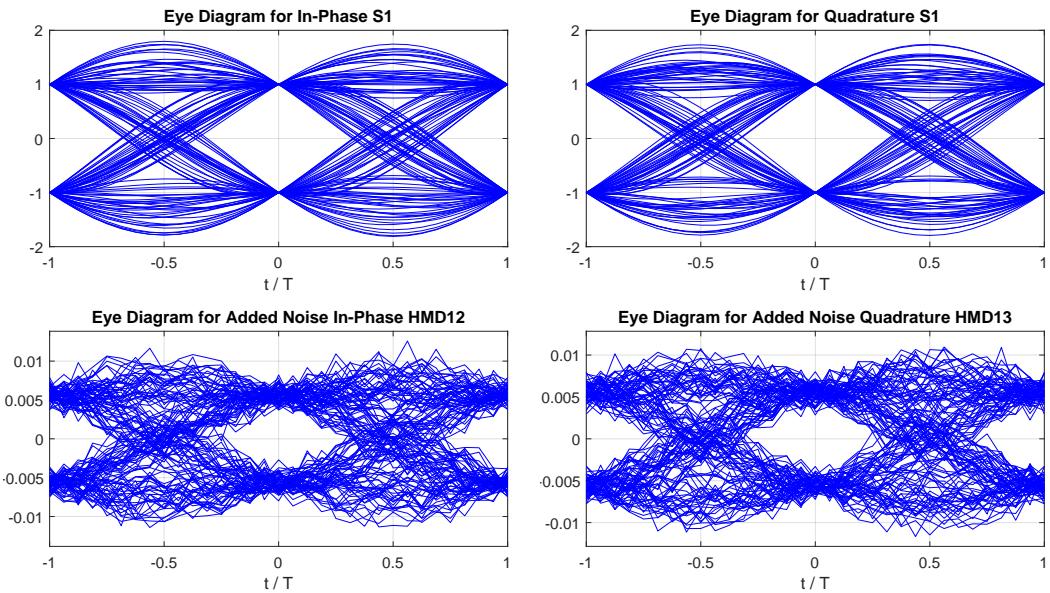


Figure 6.41: Obtained at two different points in the system: optical output of transmitter on the top and the noisy signal at the bottom.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-45 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	15 dB
numericalSNR	14.9696 dB
numericalSNR Upper Bound (95% confidence)	15.006 dB
numericalSNR Lower Bound (95% confidence)	14.9329 dB

**Root-Raised-Cosine Signal (roll-off=0.3) with Added Noise and Matched Filtering, SNR = 15 dB**

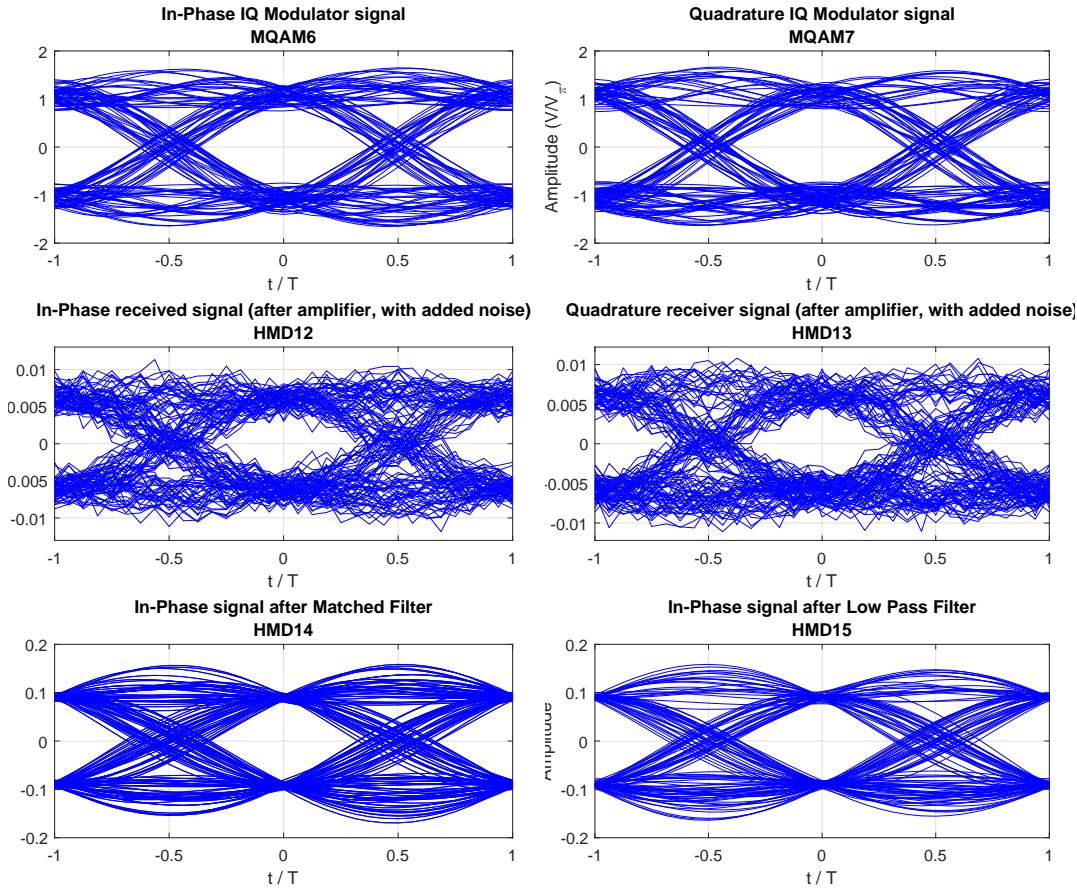


Figure 6.42: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

### Signals with AWGN and low SNR

Figures 6.45-6.43 show eye diagrams similar to the previous section, but with a lower optical power ( $-60 \text{ dBm}$ ), comparable to the average noise power ( $10^{-6} \text{ V}^2$ ).

Figures 6.43 and 6.44 show the diagrams obtained without matched filtering and with matched filtering, respectively, both using a roll-off factor of 0.9.

In this example the effects of matched filtering is even more obvious, as without it the signal visually appears to be random noise.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RaisedCosine
outputFilter	
rollOffFactor	0.9
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	0 dB
numericalSNR	-1.07702 dB
numericalSNR Upper Bound (95% confidence)	-1.0038 dB
numericalSNR Lower Bound (95% confidence)	-1.1515 dB

**Raised-Cosine Signal (roll-off=0.9) with Added Noise, SNR = 0 dB**

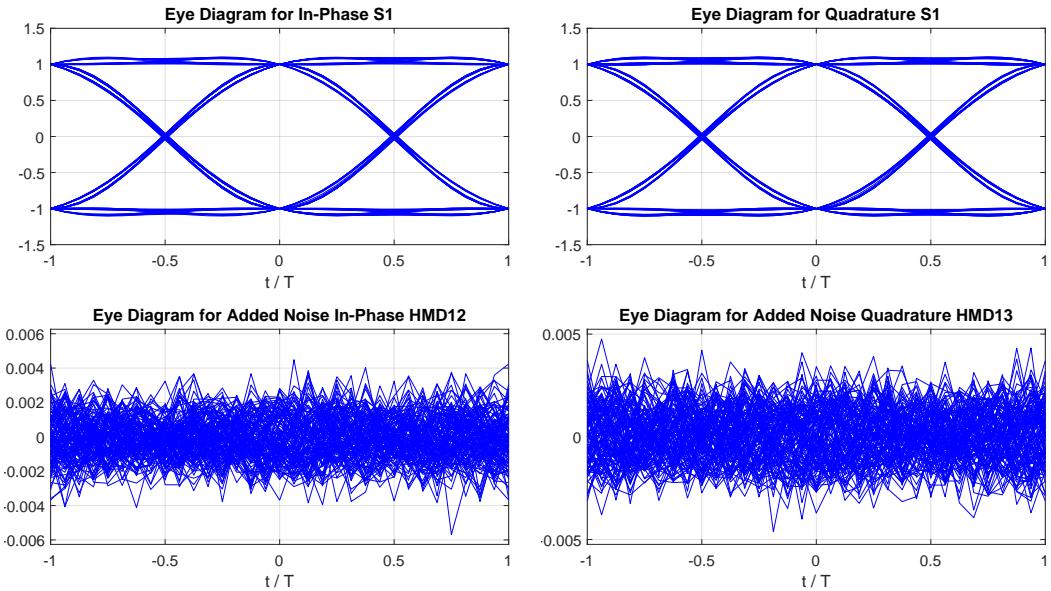


Figure 6.43: Obtained at two different points in the system: optical output of transmitter on the top and the noisy signal at the bottom.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.9
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	0 dB
numericalSNR	0.0290907 dB
numericalSNR Upper Bound (95% confidence)	0.120973 dB
numericalSNR Lower Bound (95% confidence)	-0.064778 dB

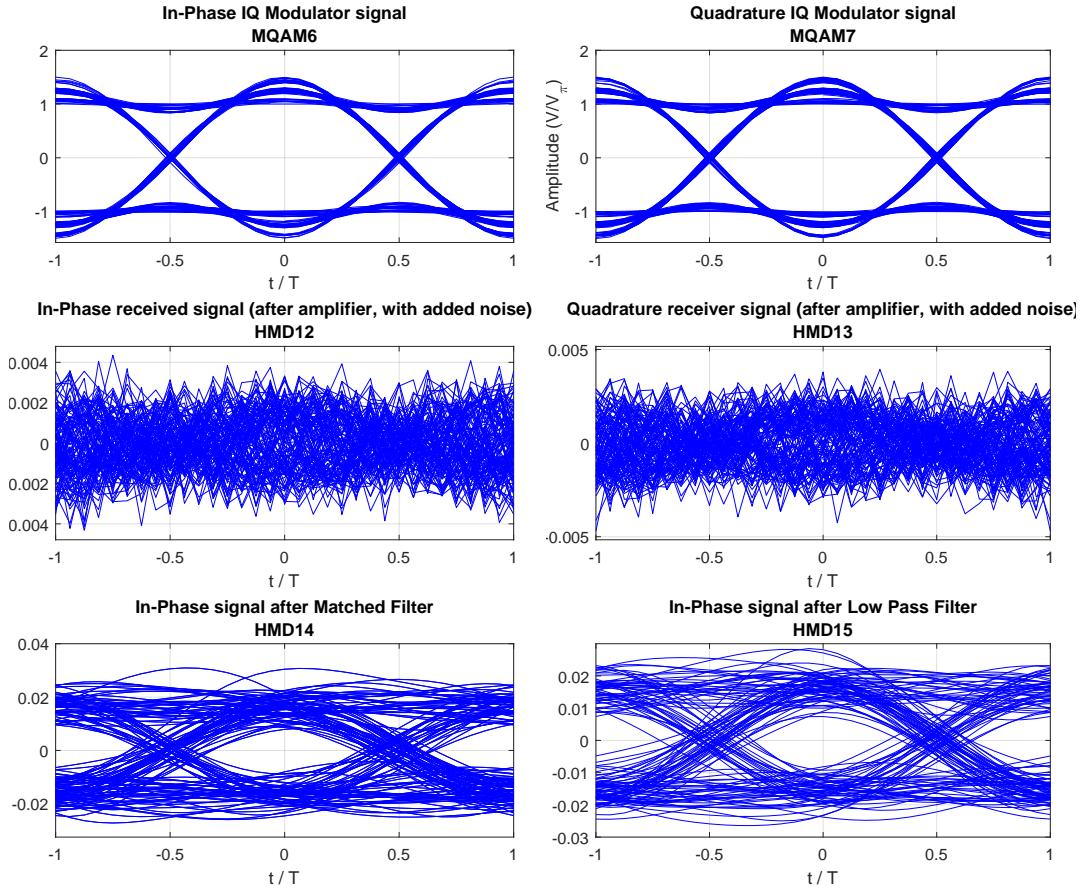
**Root-Raised-Cosine Signal (roll-off=0.9) with Added and Matched Filtering, SNR = 0 dB**

Figure 6.44: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

Figures 6.46 and 6.44 show the same case but using a roll-off factor of 0.3.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RaisedCosine
outputFilter	
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	0 dB
numericalSNR	-0.306456 dB
numericalSNR Upper Bound (95% confidence)	-0.233164 dB
numericalSNR Lower Bound (95% confidence)	-0.381006 dB

**Raised-Cosine Signal (roll-off=0.3) with Added Noise, SNR = 0 dB**

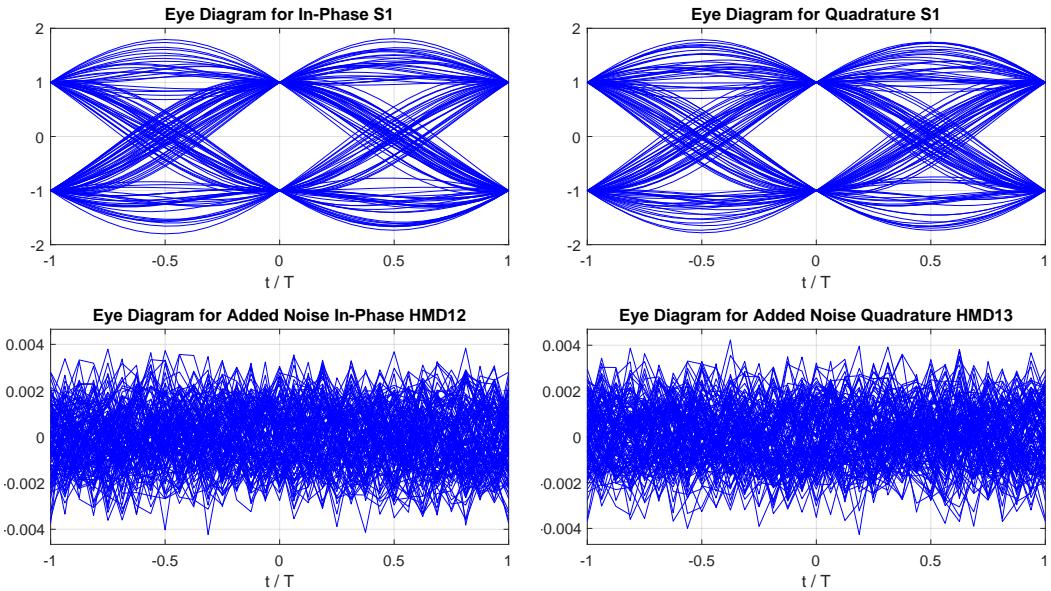


Figure 6.45: Obtained at two different points in the system: optical output of transmitter on the top and the noisy signal at the bottom.

Simulation Parameters	
Parameter	Default Value
bitPeriod	$1/32 \times 10^9$ s
samplesPerSymbol	16
outputOpticalPower	-60 dBm
shaperFilter	RootRaisedCosine
outputFilter	RootRaisedCosine
rollOffFactor	0.3
localOscillatorPower	0 dBm
localOscillatorPhase	0
responsivity	1
amplification	$10^3$
noisePower	$10^{-6}$ V <sup>2</sup>
theoreticalSNR	0 dB
numericalSNR	0.0368278 dB
numericalSNR Upper Bound (95% confidence)	0.09153 dB
numericalSNR Lower Bound (95% confidence)	-0.0185722 dB

**Root-Raised-Cosine Signal (roll-off=0.3) with Added Noise and Matched Filtering,  
SNR = 0 dB**

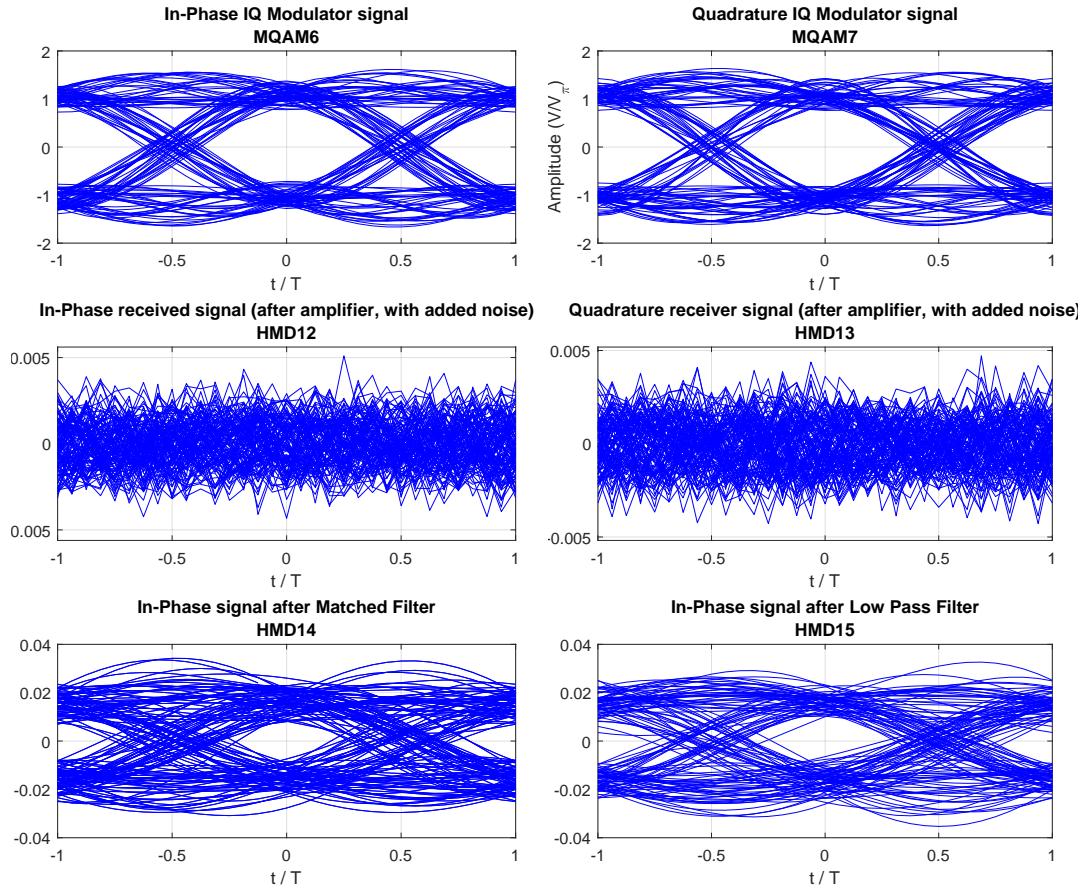


Figure 6.46: Obtained at three different points in the system: optical output of transmitter on the top; the amplified signal at the middle; and after the receiver filter.

### BER Curves

The simulated results show agreement with the theoretical curves, as can be seen in Figure 6.47.

Simulation and Curve Parameters	
Parameter	Default Value
numberOfBitsGenerated	40000
samplingRate	64 GHz
symbolRate	4 GHz
shaperFilter	Red Curve: RootRaisedCosine; Blue Curve: RaisedCosine
receiverFilter	Red Curve: RootRaisedCosine; Blue Curve: no filter
rollOff	0.3
outputOpticalPower	Variable
localOscillatorPower	0 dBm
responsivity	1
amplification	$10^3$
noisePower	$10^{-6} \text{ V}^2$
confidence	0.95

Table 6.10: Parameters for the theoretical curves and simulation results shown in Figure 6.47.

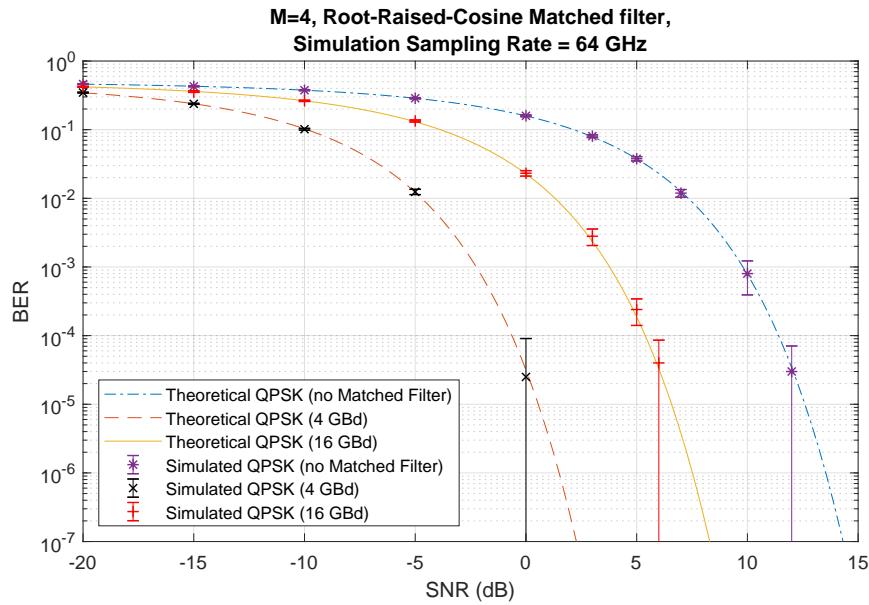


Figure 6.47: QPSK theoretical BER values as a function of the output optical power in dBm.

In this section we show the simulation results and compared them with the theoretical predictions for an M-QAM system with  $M = 4$ . Figure 6.47 shows the variation of the BER with the optical power of the signal, using 40000 bits produced by a random number generator. The noise power was set at  $10^{-6} \text{ V}^2$ , the local oscillator at 0 dBm and the amplification at the transimpedance amplifier was set at  $10^3$ . The red and blue lines represents the theoretical curve, with and without matched filters, respectively. The the red

and purple points represent the simulated values for the same situation with the respective confidence margins. The simulation agrees closely with the theoretical values.

## Conclusions

The use of a root-raised-cosine filters for shaping and filtering the signal provides the best results, due to reducing noise while creating no inter-symbol interference. The experimental BER curves agree with the theoretical values and show the advantages of matched filtering.

### 6.4.3 Experimental Analysis

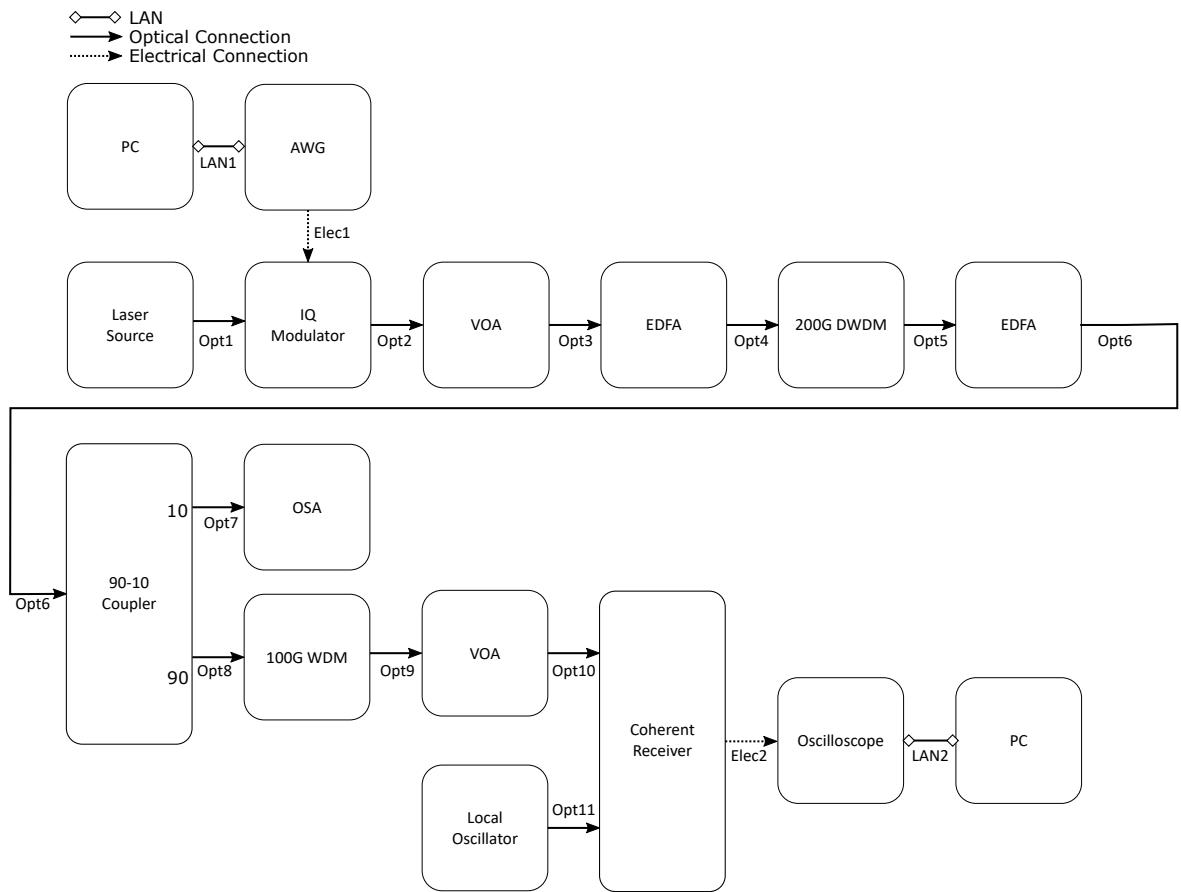


Figure 6.48: Experimental setup

The setup shown in Figure 6.48 was used to obtain experimental results to compare with the theory and validate the simulation. The list of devices used for the setup is available in Table 6.11. Tables 6.12 to 6.16 show the devices' specifications.

Device	Model	Description
Laser Source	Yenista OSICS Band C/AG TLS Laser	Optical laser source for modulate the signal
IQ Modulator	22.5GHz IQ Modulator with automatic Bias Controller	
AWG	Keysight M8195A	
VOA		USB-Controlled Variable Optical Attenuator
EDFA	Constelex Hydra-C-17-17 EDFA	
200G DWDM		
EDFA	Constelex Hydra-C-17-17 EDFA	
90/10 Coupler		
OSA	Apex Technologies AP-2043B	
100G WDM		
VOA		
Local Oscillator	Emcore CRTND3U02D ECL Laser	
Coherent Receiver	Picometrix CR-100D	
Oscilloscope	Tektronix DPO77002SX-R3	

Table 6.11: Devices in experimental setup.

Tables 6.12 to 6.16 list the relevant specifications for the devices used in the experimental setup.

<b>Tektronix DPO77002SX-R3 Oscilloscope, TekConnect channels</b>	
Analog channels bandwidth	33 GHz
Sample rate per channel	100 GS/s
Rise time (typical)	
10% to 90%	13 ps
20% to 80%	9 ps
Vertical noise, BWE on, max sample rate (typical)	0.71% of full scale 0.56% of full scale @ 0 V offset (500 mV <sub>FS</sub> )
Timing resolution	10 ps
Sensitivity range	62.5 mV <sub>FS</sub> to 6 V <sub>FS</sub>
Vertical resolution	8 bits (11 bits with averaging)
Effective number of bits	5.0 bits @ 500 mV <sub>FS</sub> , 100 GS/s
Channel to channel Isolation (DC to 33 GHz)	60 dB

Table 6.12

<b>Yenista OSICS TLS-AG Wide C-band</b>	
Frequency Range (Wavelength Range)	196.275 - 191.125 THz (1527.41 - 1568.57 nm)
Output Power	20 mW (+13 dBm)
Power Range (typ.)	+6 to +13.6 dBm
Relative Frequency (Wavelength)	±0.5 GHz (± 4 pm)
Accuracy (typ.)	
Absolute Frequency (Wavelength)	±1.5 GHz (± 12 pm)
Accuracy (typ.)	
Frequency Setting Resolution	Down to 1 MHz
Instantaneous Linewidth (FWHM)	< 100 kHz
Power Stability	±0.03 dB
Absolute Output Power Deviation	±0.2 dB
Accross Tuning Range	
Side Mode Suppression Ratio	60 dB
Relative Intensity Noise	-145 dB/Hz

Table 6.13

<b>Local Oscillator Laser</b>	
Optical Output Power Adjustment Range	+7 - +13.5 dBm
Optical Output Power Adjustment Range - high power variant	7 - 15.5 dBm
Short term power variation	0.05 dB
Optical Output Power Step Size	0.01 dB
Operating Frequency (Wavelength) Range	191.5 - 196.25 THz (1527.6 - 1565.5 nm)
Fine Tune Frequency Resolution (typ.)	1 MHz
Fine Tune Frequency Range	-6 - +6GHz
Frequency Accuracy EOL	±2.5 GHz
Frequency Accuracy EOL (25 GHz spacing variant)	±1.5 GHz
Instantaneous Linewidth (FWHM)	100 kHz
Relative Intensity Noise (+13dBm output)	-145 dB/Hz
Relative Intensity Noise (+7dBm output)	-140 dB/Hz
Side Mode Suppression Ratio (typ.)	55 dB
Back reflection	-14 dB
Optical Isolation	30 dB
SSER (typ.)	55 dB
Polarization Extinction Ratio	20 dB

Table 6.14

<b>Balanced Receiver</b>		
Wavelength Range		1525 - 1570 nm
Bit Rate (max)		32 Gb/s
Signal Input Power		-6 dBm
Polarization Extinction Ratio		18 dB
LO input Power		16 dBm
I/Q relative phase in Mixer		
	minimum	85 deg
	typical	90 deg
	maximum	95 deg
I/Q phase stability in mixer (over lifetime)		-2 - +2 deg
PBS mixer excess loss		3.1 dB
Optical input return loss		-27 dB
Bandwidth (-3 dB elec.)		18.5 - 28 GHz
Low frequency cutoff		100 kHz
Group delay variation		
	0.1 - 15 GHz	-3 - +3 ps
	15 - 25 GHz	-9 - +9 ps
CMMR (signal input)		
	DC	-17 dB
	22GHz	-16 dB
CMMR (LO input)		
	DC	-12 dB
	22GHz	-10 dB
Linearity		5%
Temporal Skew		
	P/N outputs	3 ps
	Across all four channels	10 ps
Photodiode dark current (25°C)		100 nA
Photodiode responsivity @1550 nm		0.64 A/W
Effective responsivity (both inputs)		0.05 A/W

Table 6.15

Constelex Hydra-C EDFA	
Input wavelength range	1530-1565 nm
Saturated output power	13 -21 dBm
Input power	-20 - +3 dBm
Small signal gain (Pin = -20dBm)	>28 dB
Noise Figure (Pin = -10dBm @1555 nm)	<4 dB

Table 6.16

Keysight M8195A AWG	
Sample Rate	65 GSa/s
Analog Bandwidth (typ.)	25 GHz
Vertical Resolution	8 bits
Amplitude (single ended)	75 mV <sub>PP</sub> to 1 V <sub>PP</sub>
Amplitude Resolution	200 $\mu$ V
Intrinsic Random Jitter	< 200 fs
Rise time (typical)	
20% to 80%	18 ps

Table 6.17

22.5 GHz IQ Modulator with automatic Bias Controller	
Wavelength Range	1520 - 1580 nm
Electro Optical Bandwidth (min.)	22 GHz
Electro Optical Bandwidth (typ.)	25 GHz
Optical Return Loss	30 dB
Electrical Input High Frequency 3 dB point (typ.)	40 GHz
Electrical Input High Frequency 3 dB point (max.)	65 kHz
Electrical input Gain Ripple (typ.)	$\pm 1$ dB
Gain delay ripple (max.)	$\pm 50$ ps

Table 6.18

Figures 6.51 to 6.52 show the BER curves obtained resorting to the OptDSP libraries for signal processing. The post-processing is done offline and several processing stages, including signal synchronization, frequency offset correction, phase offset correction and adaptive equalization. The SNR used to plot the experimental data points in Figure 6.51 is estimated offline, based on spectral analysis of the waveforms obtained in the oscilloscope. The SNR in Figure 6.52 is also calculated offline, but through a different method more appropriate to the estimation of SNR from constellation points, commonly known as  $M_2M_4$ , or the moments method [3].

The experimental results found here were obtained for two different symbol rates, 4 GBd and 16 GBd.

#### 4 GBd

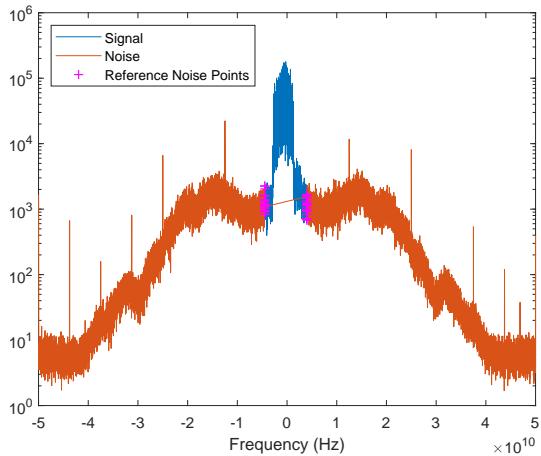


Figure 6.49  
16 GBd

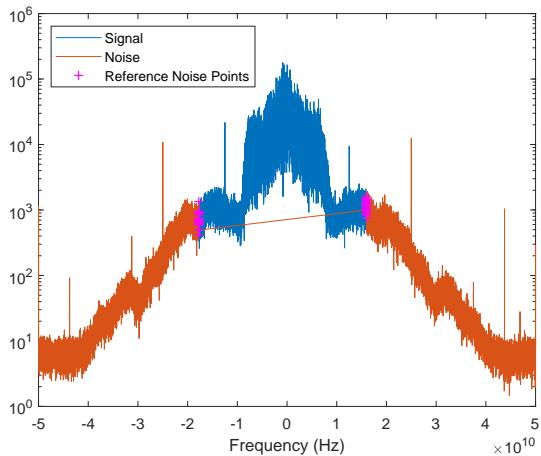


Figure 6.50

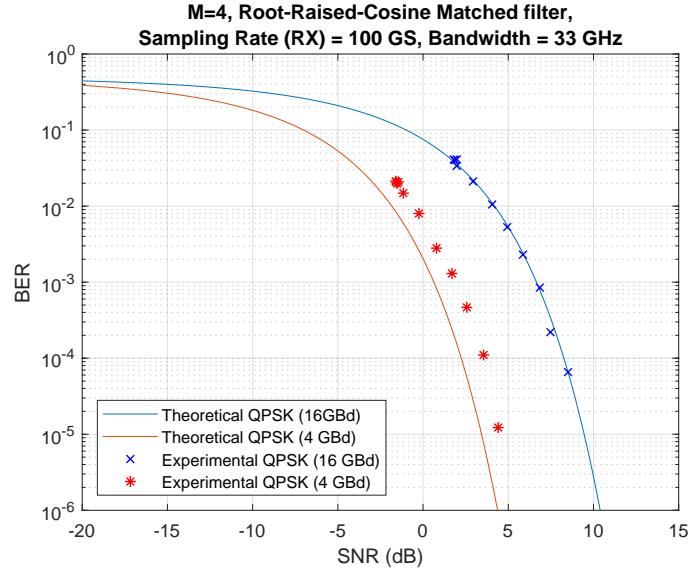


Figure 6.51: Comparison of theoretical BER curves against obtained results. SNR values are estimated from the raw waveforms obtained at the oscilloscope, through spectral analysis of the acquired signal.

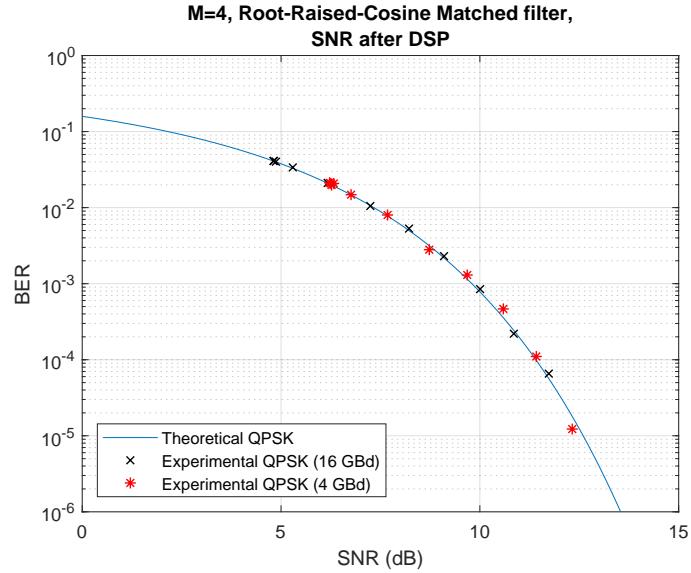


Figure 6.52: SNR after DSP vs measured BER. Estimated SNR present in the final constellation was calculated through the moments method [3].

## 16 GBd Signal

Figure 6.53 shows the optical spectrum of a 16 GBd signal. The estimated SNR is 12.9755 dB. The signal power was calculated by integrating the area of the signal, subtracting the estimated noise that shares the same bandwidth, and dividing by the estimated signal bandwidth. The noise for the SNR was calculated from the remainder of the spectrum, and divided by the full frequency interval of the spectrum.

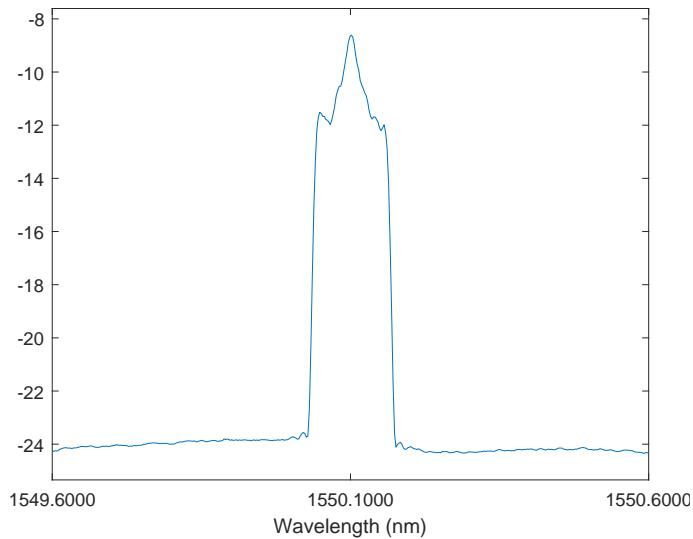


Figure 6.53: Optical spectrum of the transmitted signal, obtained through the OSA, at the point shown in Figure 6.48.

Figures 6.54 to 6.61 show the eye diagrams and the spectrum of the signals at every stage of the DSP process. The process consists of a series of several consecutive steps toward improving the signal. It starts by applying a series of front-end corrections, including the removal of the DC component, Gram-Schmidt Orthonormalization and deskew of the components. After that, the signal is put through a low pass filter, typically a matched filter, and decimated from the 100 GHz to 32 GHz, so it has only two samples per symbol. The most important steps are what follows, the adaptive equalization, frequency and phase corrections. Adaptive equalization is done in two stages, the first after the matched filter, using a constant modulus algorithm, and the second after the first stage of carrier-phase estimation, using an LMS algorithm. Frequency mismatch compensation is done after the first stage of adaptive equalization, by analysis of the spectral peak after demodulation of the signal. It is followed by carrier-phase estimation, using the Viterbi-Viterbi algorithm. After a second stage of adaptive equalization, the signal is downsampled to the symbol rate, so that it only contains the symbols.

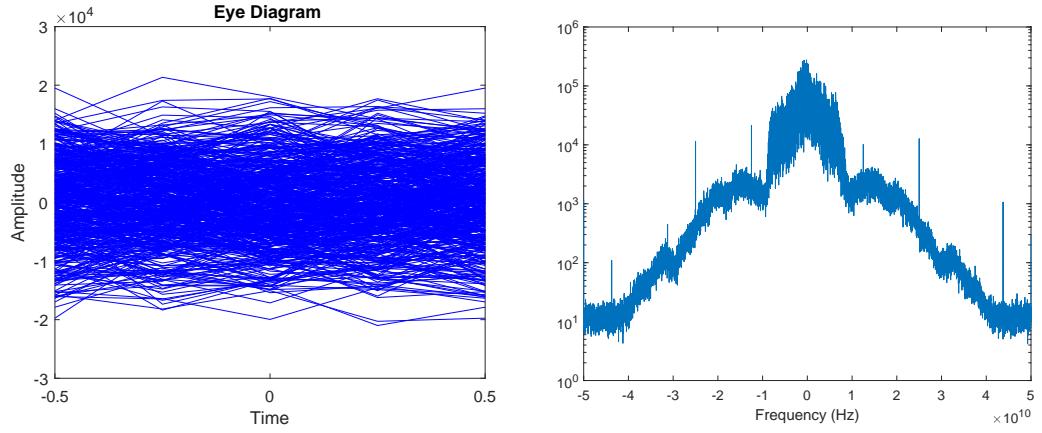


Figure 6.54: Eye Diagram and spectrum of the original signal obtained at the oscilloscope. Initial estimated SNR before any signal processing is 7.94 dB.

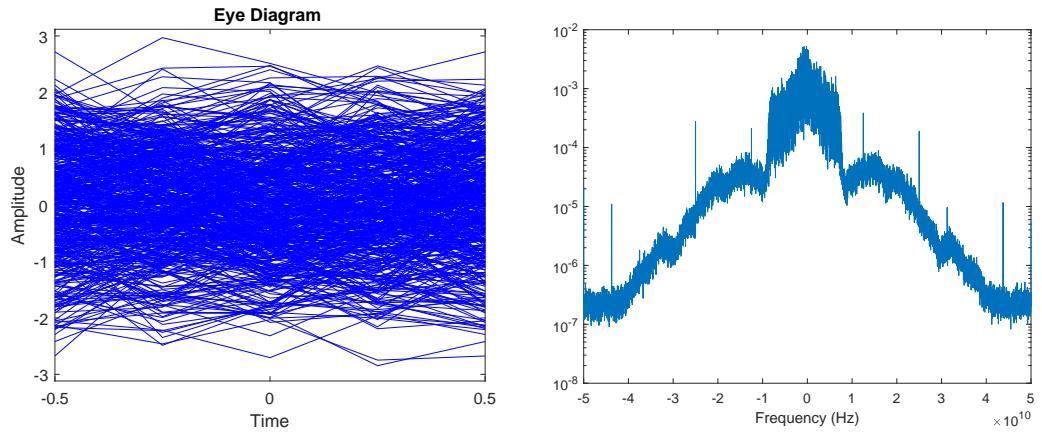


Figure 6.55: Eye Diagram and spectrum after front end corrections, including removal of DC component, deskew and Gram-Schmidt Orthonormalization.

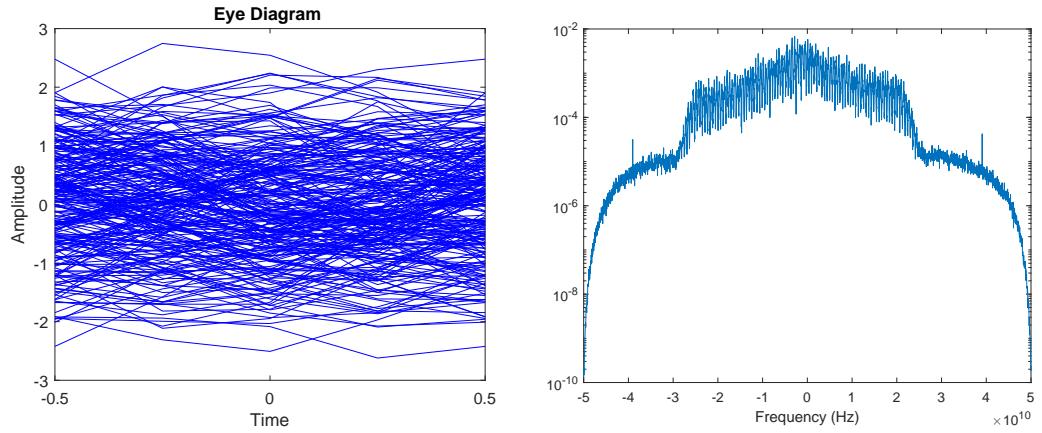


Figure 6.56: Eye Diagram and spectrum after applying the matched filter and downsampling to  $2S_R$ .

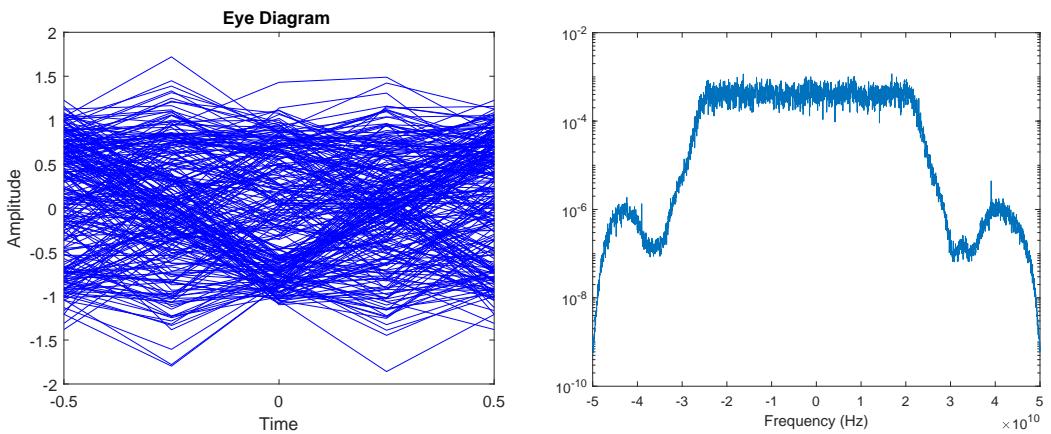


Figure 6.57: Eye Diagram and spectrum after the first stage of adaptive equalization, using a MIMO 2x2 CMA equalizer.

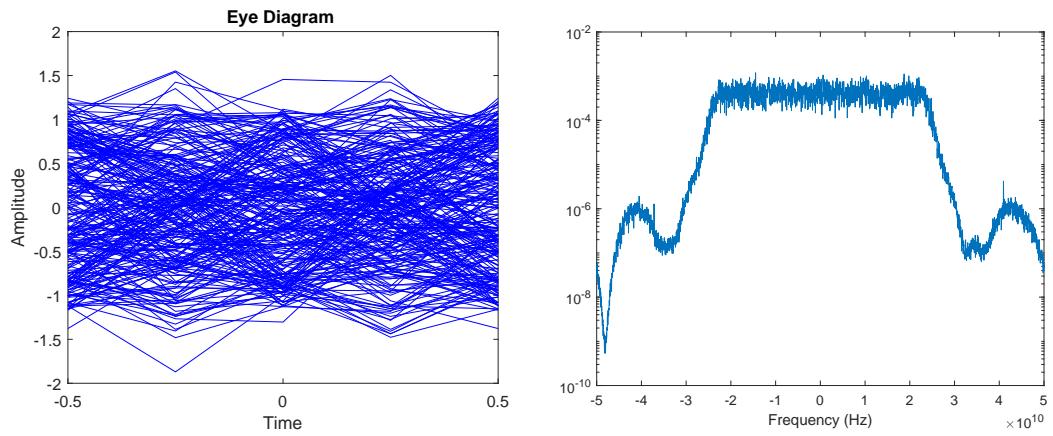


Figure 6.58: Eye Diagram and spectrum after frequency offset correction.

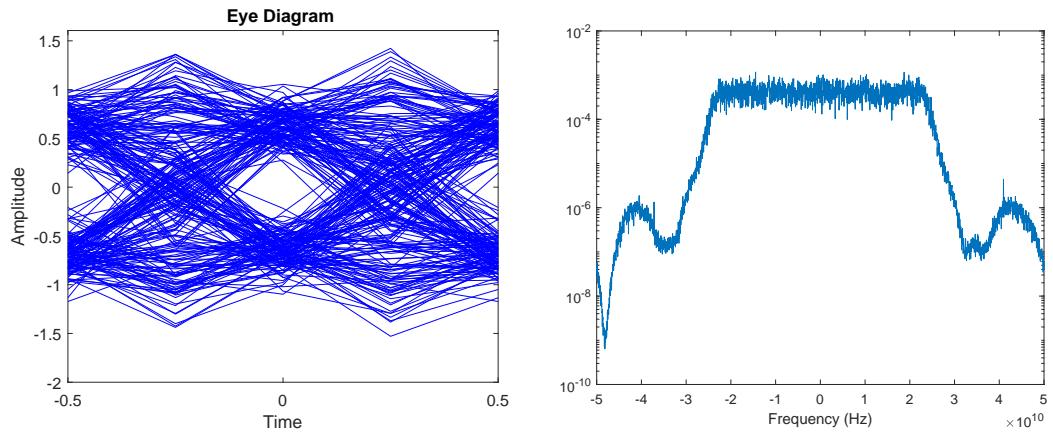


Figure 6.59: Eye Diagram and spectrum after the first stage of carrier-phase estimation.

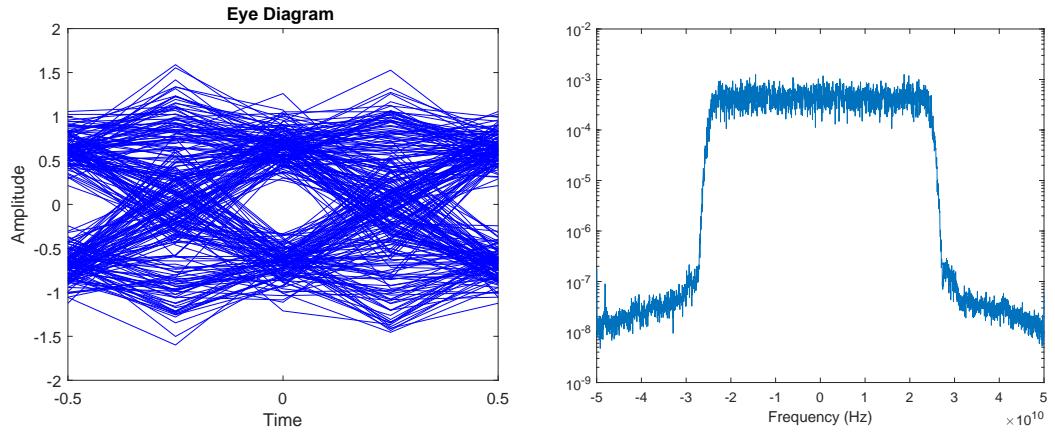


Figure 6.60: Eye Diagram and spectrum after the second stage of adaptive equalization, with a MIMO 4x4 LMS equalizer.

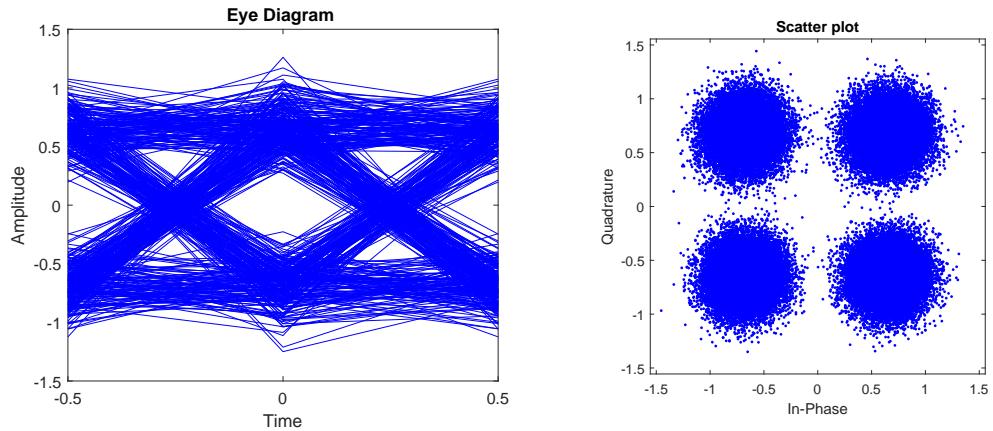


Figure 6.61: Eye Diagram after downsampling to 1 sample per symbol, and final constellation obtained after processing. The BER in this case is  $6.5641 \times 10^{-5}$ .

#### 4 GBd Signal

Figure 6.62 shows the optical spectrum of a 4 GBd signal. The estimated SNR is 13.0930 dB. The signal power for the SNR was calculated by integrating the area of the signal, subtracting the estimated noise that shares the same bandwidth, and dividing by the estimated signal

bandwidth. The noise for the SNR was calculated by integrating the remainder of the spectrum, interpolating the signal area, and dividing by the full frequency interval of the spectrum.

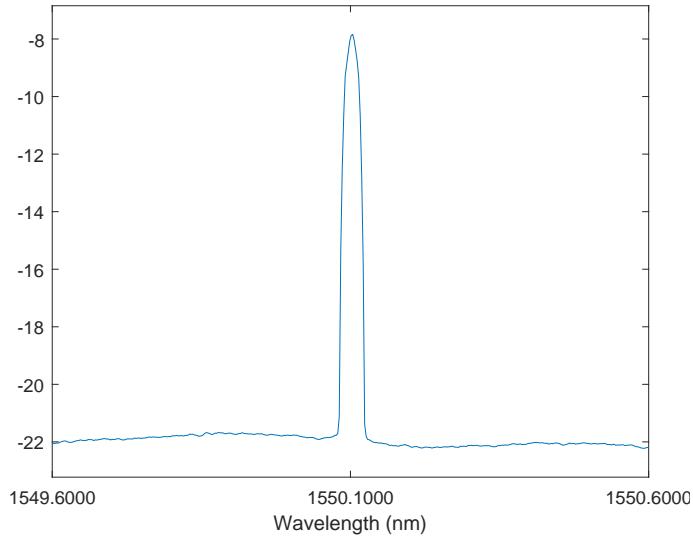


Figure 6.62: Optical spectrum of the transmitted signal, obtained through the OSA, at the point shown in Figure 6.48.

Figures 6.84 to 6.70 show the eye diagrams and the spectrum of the signals at every stage of the DSP process. The process is similar to the described for the 16 GBd signal. However, some differences can be seen, particularly on the signal spectrum before the corrections. A particularly relevant difference is the stronger deviation from the theoretical curve in this case (for ber curve using the SNR calculated at the oscilloscope input), as can be seen in Figure 6.51. This can be due to how the SNR is calculated, as it is assumed the signal spectrum is centered in at zero in the frequency domain, and this assumption is not true, as can be seen from the spectrum examples below, particularly in the spectrum of Figure 6.86. It is also worth noting that in the input signal the noise is not white. Therefore, the reference points chosen to calculate the noise that shares the frequencies of the signal strongly affect the SNR measurement.

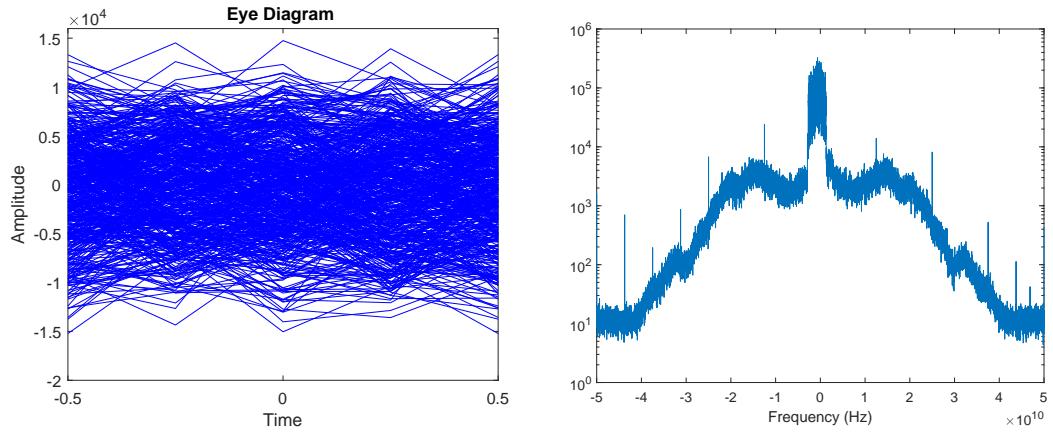


Figure 6.63: Eye Diagram and spectrum of the original signal obtained at the oscilloscope. Initial estimated SNR before any signal processing is 4.26 dB.

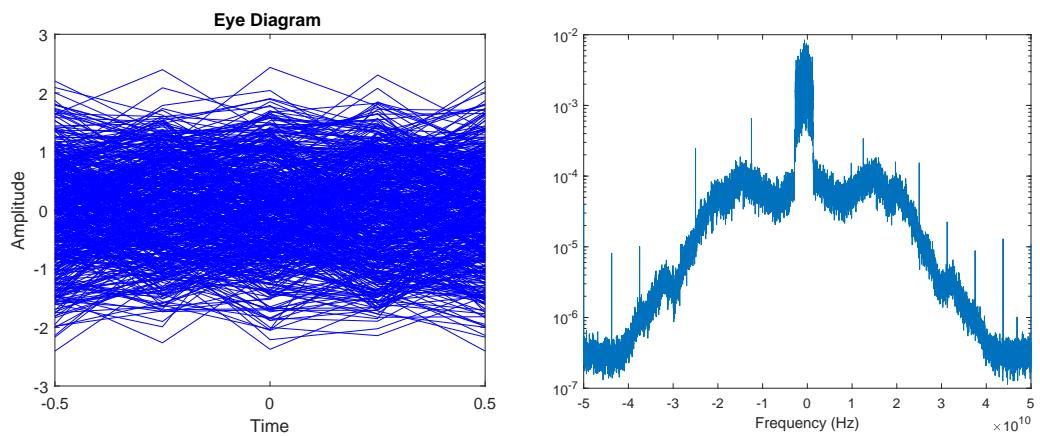


Figure 6.64: Eye Diagram and spectrum after front end corrections, including removal of DC component, deskew and Gram-Schmidt Orthonormalization.

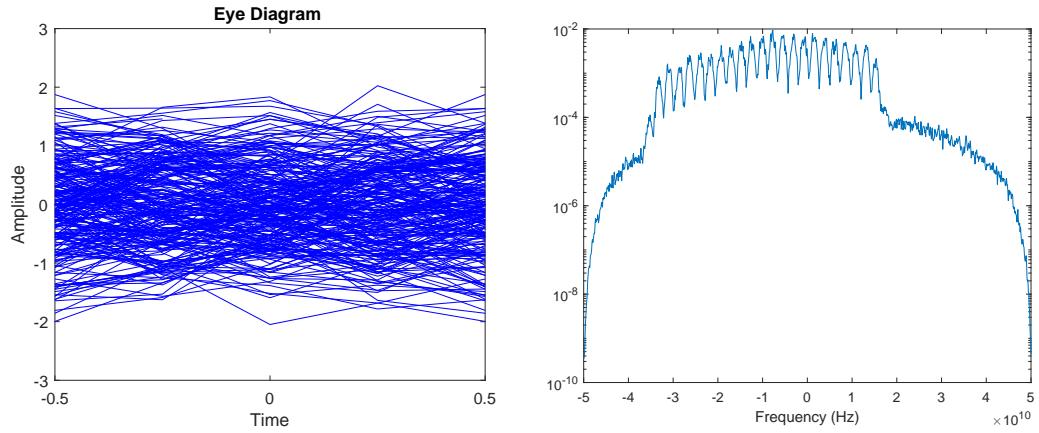


Figure 6.65: Eye Diagram and spectrum after applying the matched filter and downsampling to  $2S_R$ .

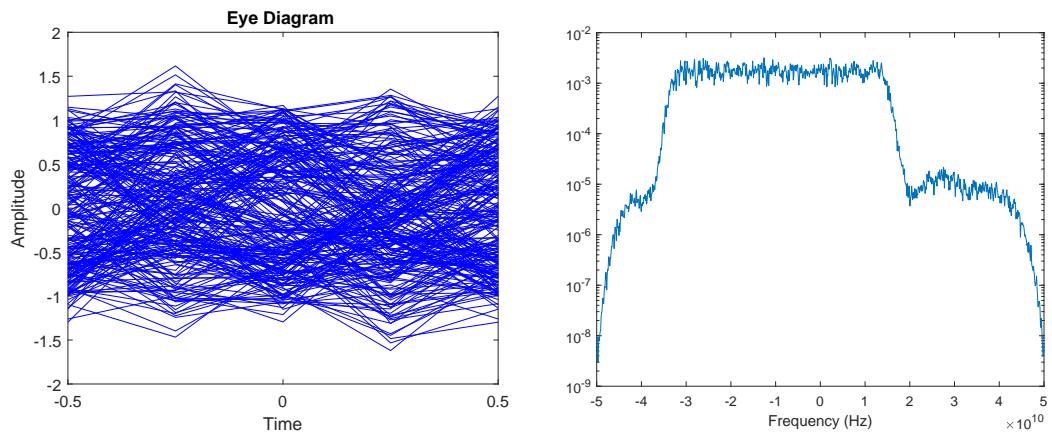


Figure 6.66: Eye Diagram and spectrum after the first stage of adaptive equalization, using a MIMO 2x2 CMA equalizer.

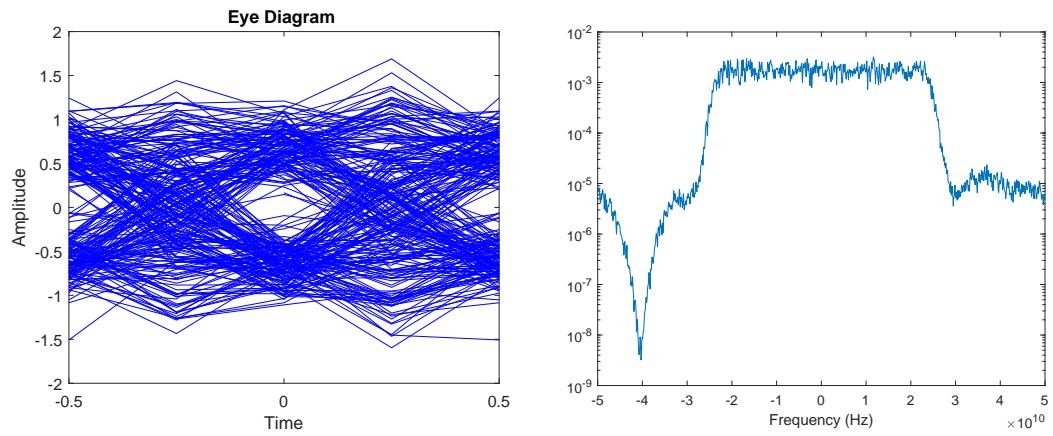


Figure 6.67: Eye Diagram and spectrum after frequency offset correction.

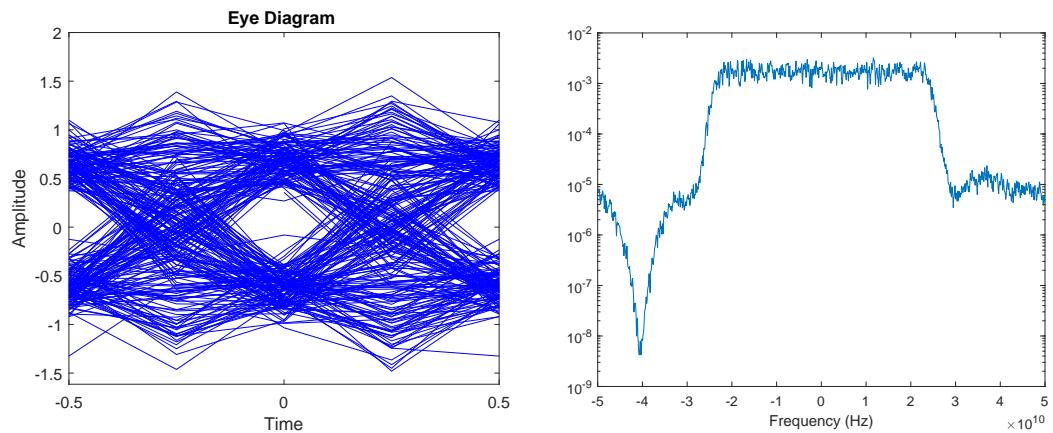


Figure 6.68: Eye Diagram and spectrum after the first stage of carrier-phase estimation.

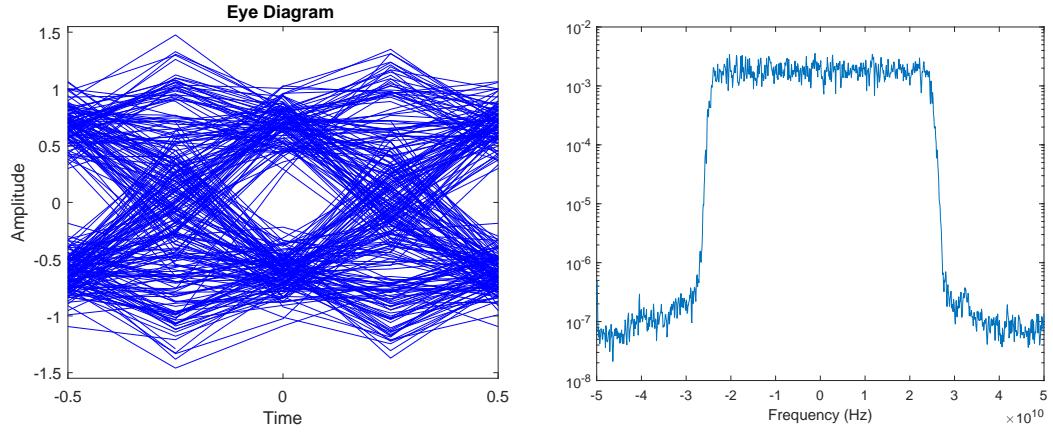


Figure 6.69: Eye Diagram and spectrum after the second stage of adaptive equalization, with a MIMO 4x4 LMS equalizer.

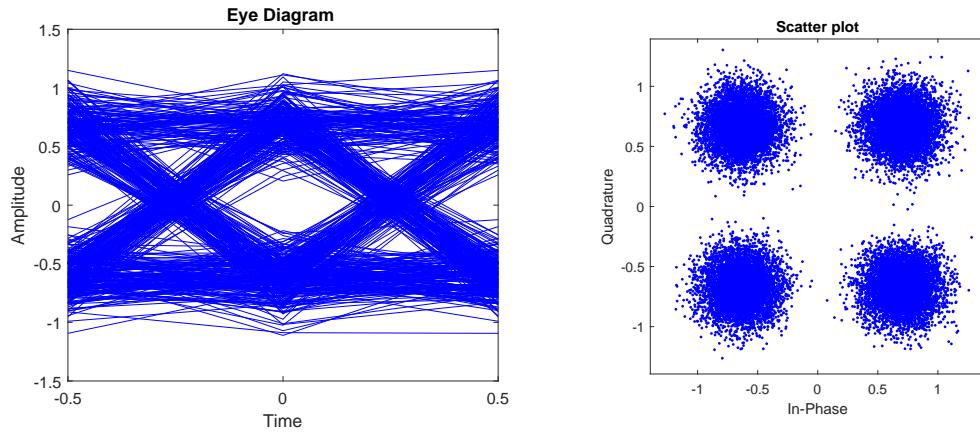


Figure 6.70: Eye Diagram after downsampling to 1 sample per symbol, and final constellation obtained after processing. Final SNR estimated through the moments method [3] is 12.3261 dB. The BER in this case is  $1.2256 \times 10^{-5}$ .

In order to try and disappear with the frequency offset to improve results, the setup was changed to use the same laser beam for both the source and the local oscillator, therefore guaranteeing they have the same frequency.

#### 4 and 16 GBd Signal (DP72004B)

A different change that was also required was changing the oscilloscope, as the one that was previously used broke down. In order to provide a more reliable comparison between both setups, the measurements previously done were repeated, using now the Tektronix DP720004B. For the sake of brevity, only the spectra of the received waveforms will be shown, along with the BER curves.

Tektronix DP720004B Oscilloscope, TekConnect channels	
Analog channels bandwidth	20 GHz
Sample rate per channel	50 GS/s
Rise time (typical)	
10% to 90%	18 ps
20% to 80%	14 ps
Vertical noise, bandwidth filter on, max sample rate 50 mV/div (typical)	0.56% of full scale @ 0 V offset (500 mV <sub>FS</sub> )
Timing resolution	10 ps
Sensitivity range	200 mV <sub>FS</sub> to 5 V <sub>FS</sub>
Vertical resolution	8 bits (11 bits with averaging)
Effective number of bits	5.5 bits @ 50 mV/div, 100 GS/s

Table 6.19

Table 6.19 shows the specifications for the Oscilloscope now in use.

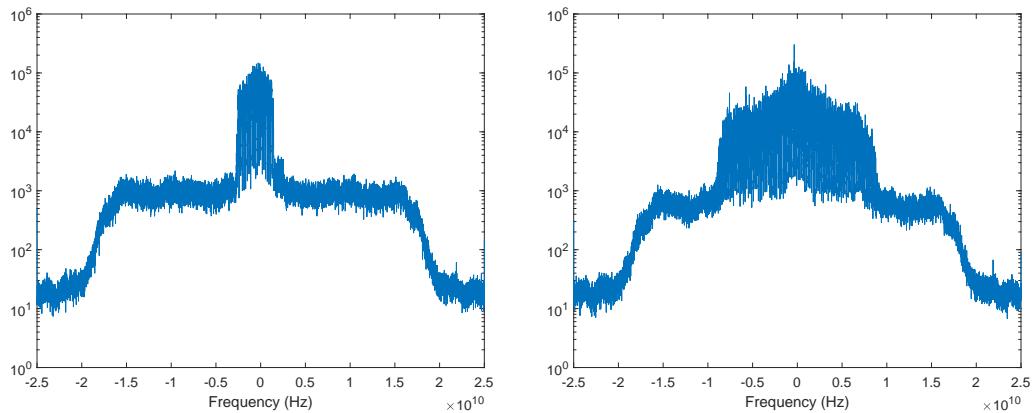


Figure 6.71: Spectrum of the received signals in the oscilloscope, using the Tektronix DP720004B. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

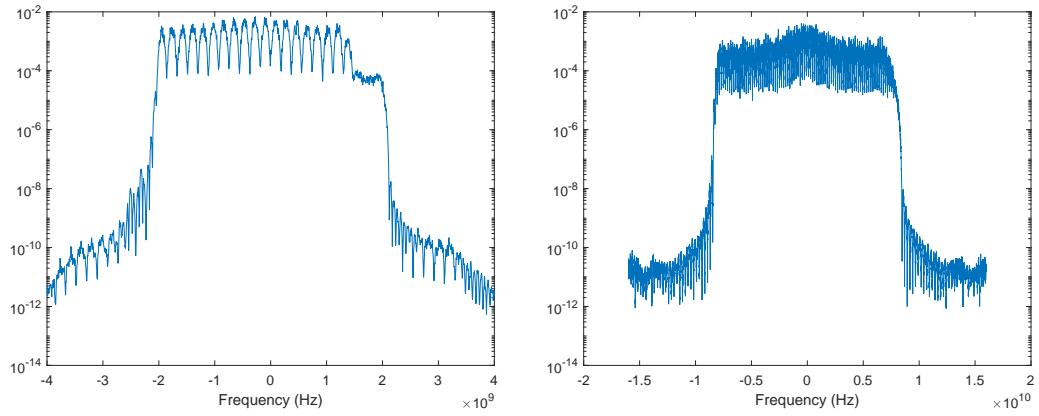


Figure 6.72: Spectrum of the signals received using the Tektronix DP720004B, after filtering the signals with a matched filter and downsampling to twice the symbol-rate. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

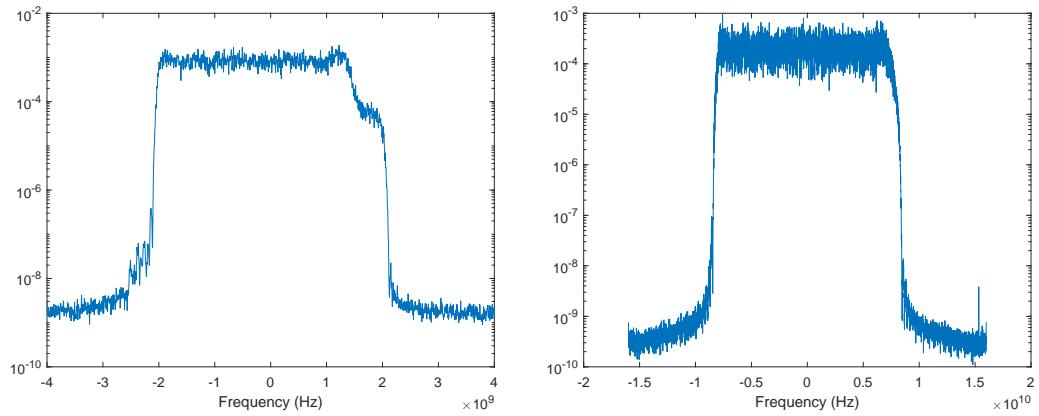


Figure 6.73: Spectrum of the signals received using the Tektronix DP720004B, after the first stage of adaptive equalization. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

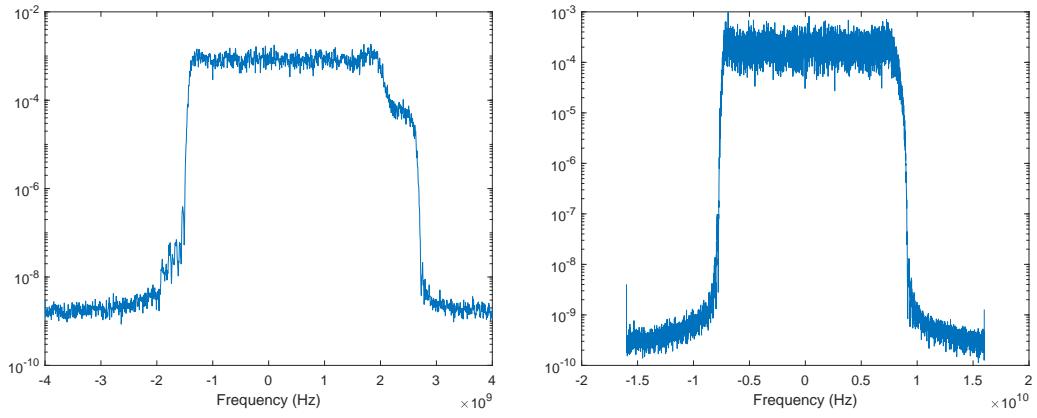


Figure 6.74: Spectrum of the signals received using the Tektronix DP720004B, after frequency estimation. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

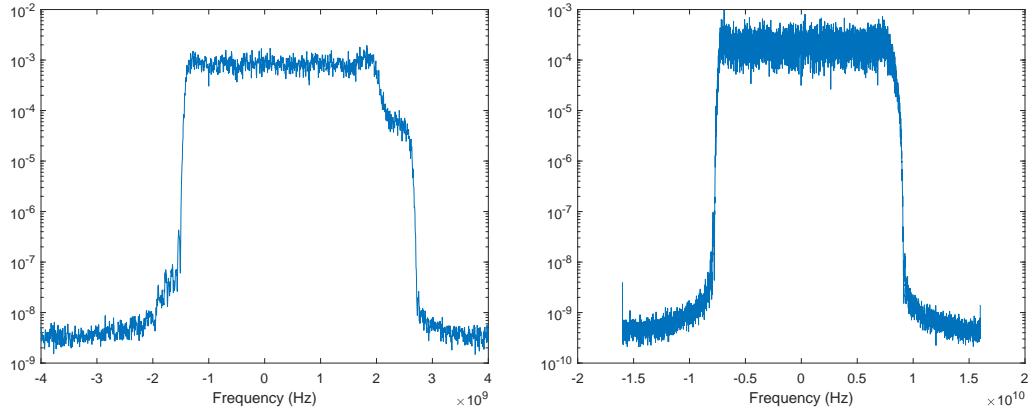


Figure 6.75: Spectrum of the signals received using the Tektronix DP720004B, after the first stage of phase-carrier recovery. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

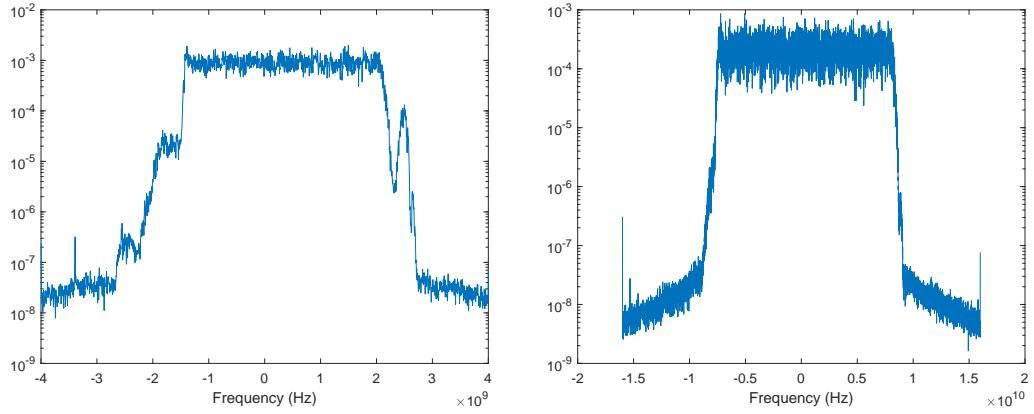


Figure 6.76: Spectrum of the signals received using the Tektronix DP720004B, after the second stage of adaptive equalization. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

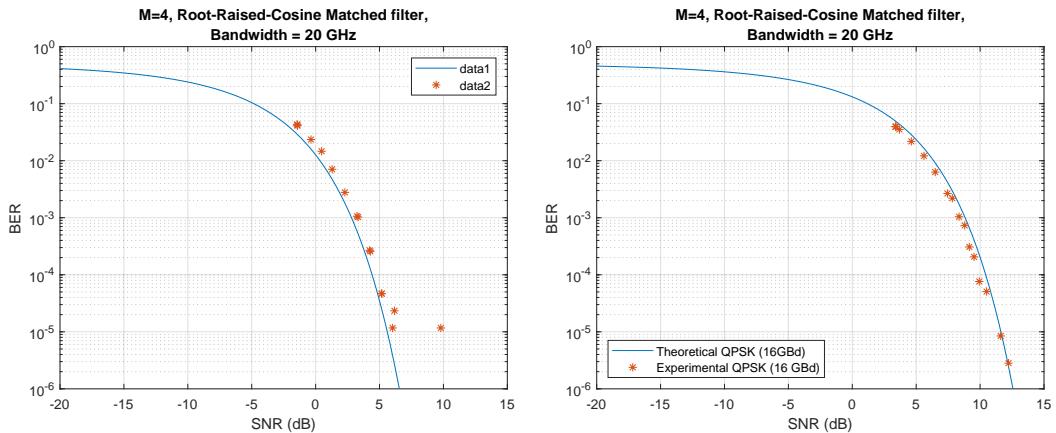


Figure 6.77: Spectrum of the signals received using the Tektronix DP720004B, after the second stage of adaptive equalization. The symbol rates are 4 GBd and 16 GBd, on the left and right, respectively.

#### 4 GBd Signal, Homodyne Detection

As the laser source used for transmitting the signal and for act as local oscillator is the same, the frequency is always the same. Therefore, using this configuration there is no need for frequency estimation or carrier-phase corrections.

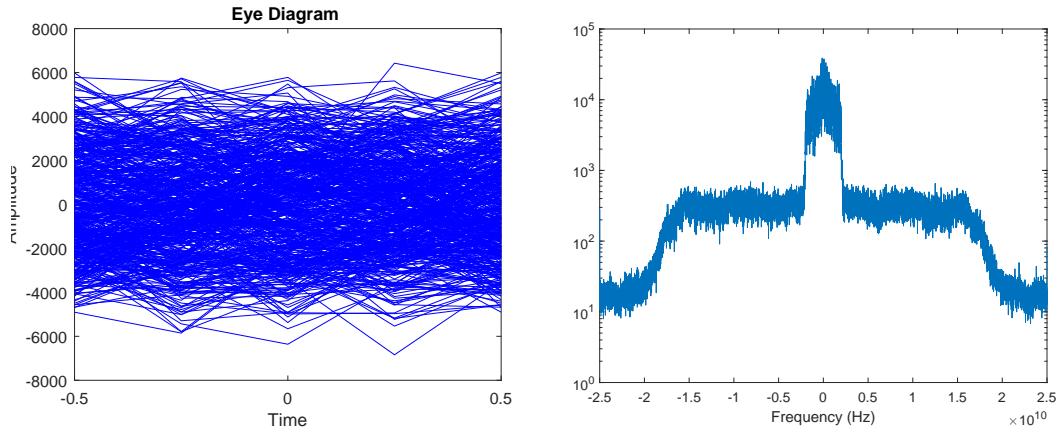


Figure 6.78: Eye Diagram and spectrum of the original signal obtained at the oscilloscope. Initial estimated SNR before any signal processing is 3.91 dB.

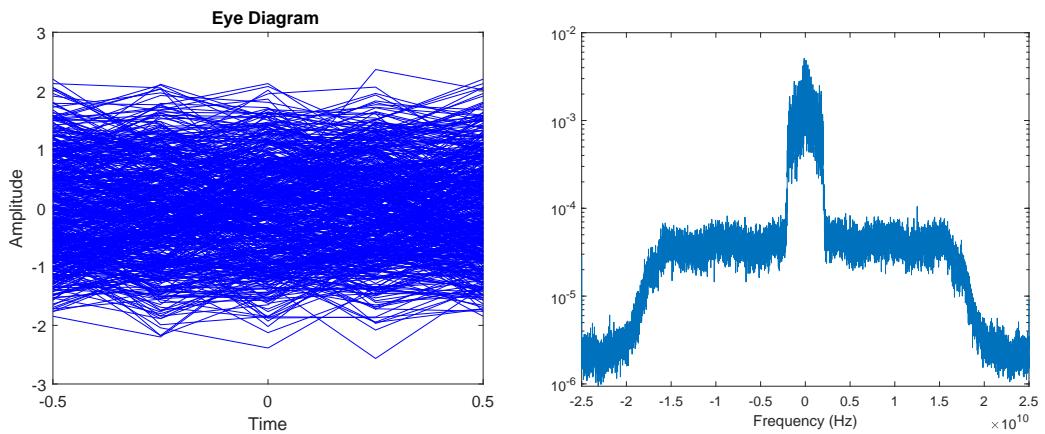


Figure 6.79: Eye Diagram and spectrum after front end corrections, including removal of DC component, deskew and Gram-Schmidt Orthonormalization.

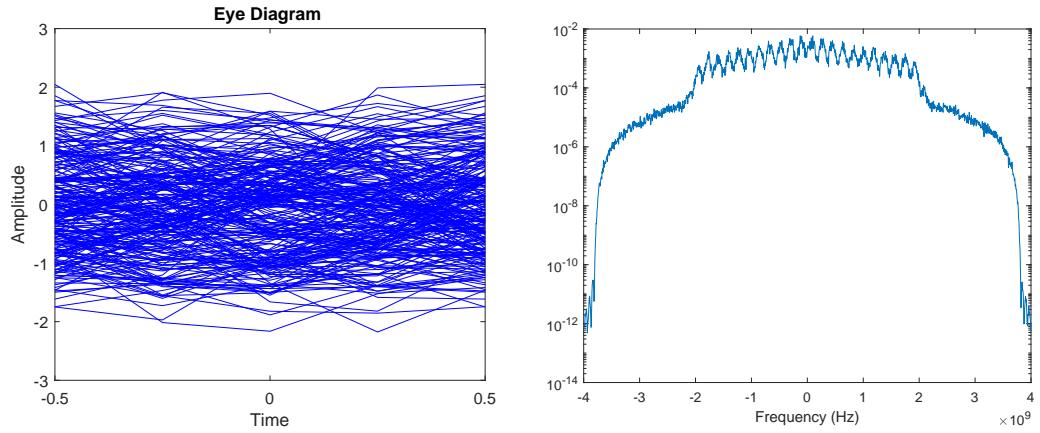


Figure 6.80: Eye Diagram and spectrum after applying the matched filter and downsampling to  $2S_R$ .

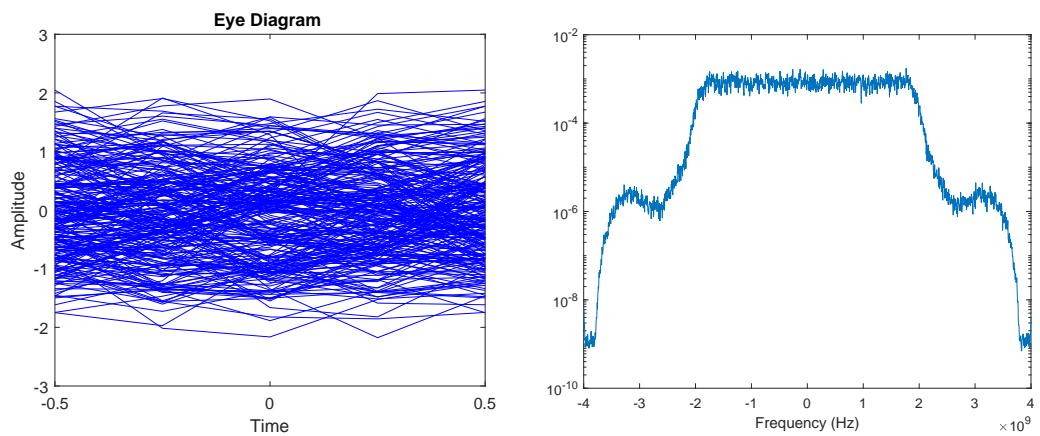


Figure 6.81: Eye Diagram and spectrum after the first stage of adaptive equalization, using a MIMO 2x2 CMA equalizer.

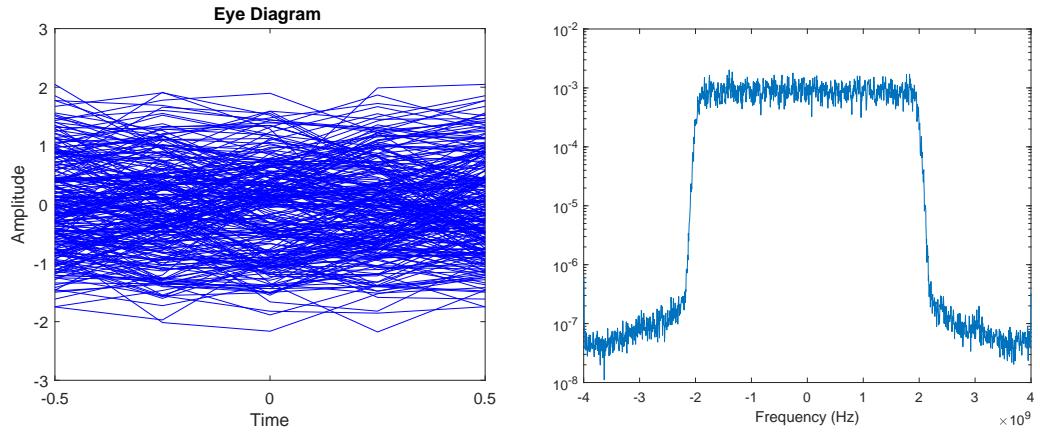


Figure 6.82: Eye Diagram and spectrum after the second stage of adaptive equalization, with a MIMO 4x4 LMS equalizer.

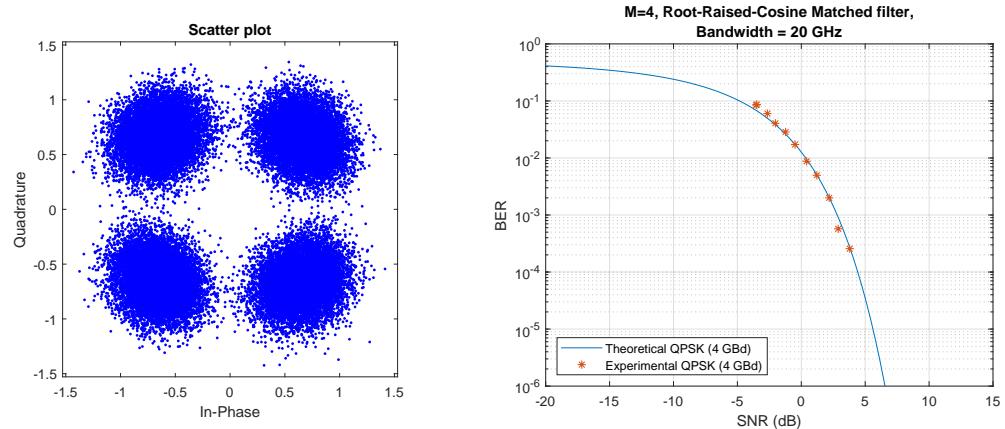


Figure 6.83: Final constellation and BER curve as a function of the SNR of the received waveform.

## 16 GBd Signal, Homodyne Detection

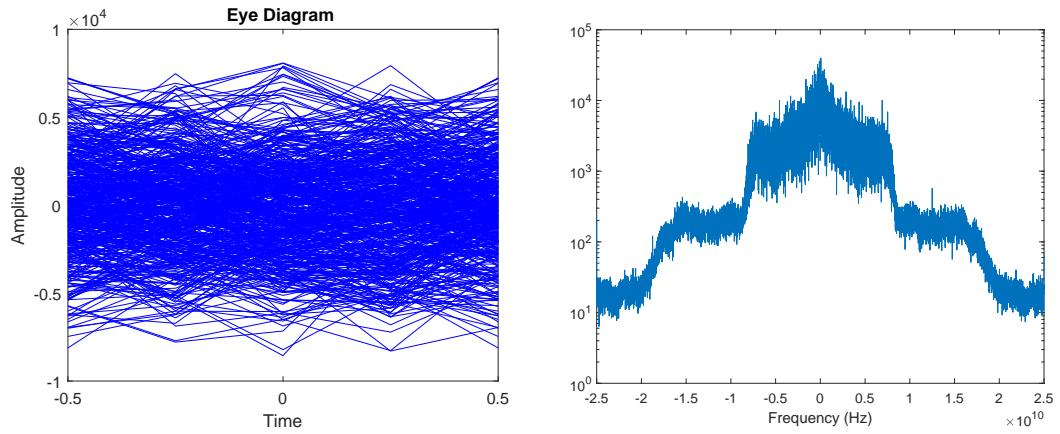


Figure 6.84: Eye diagram and spectrum of the received waveform. Estimated SNR of 8.4 dB.

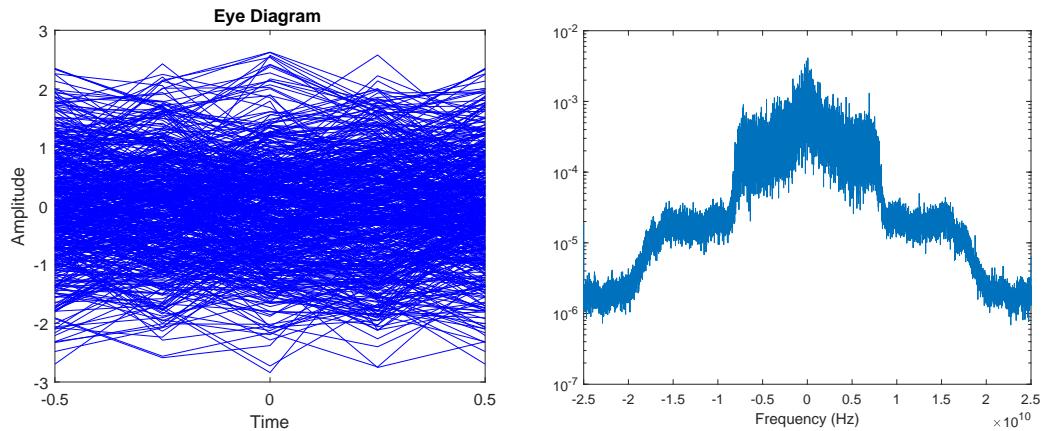


Figure 6.85: Eye Diagram and spectrum after front end corrections, including removal of DC component, deskew and Gram-Schmidt Orthonormalization.

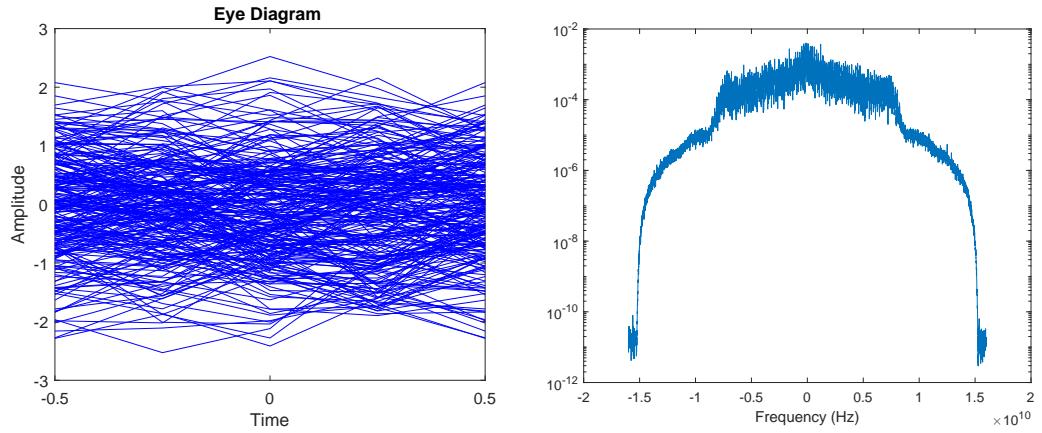


Figure 6.86: Eye Diagram and spectrum after applying the matched filter and downsampling to  $2S_R$ .

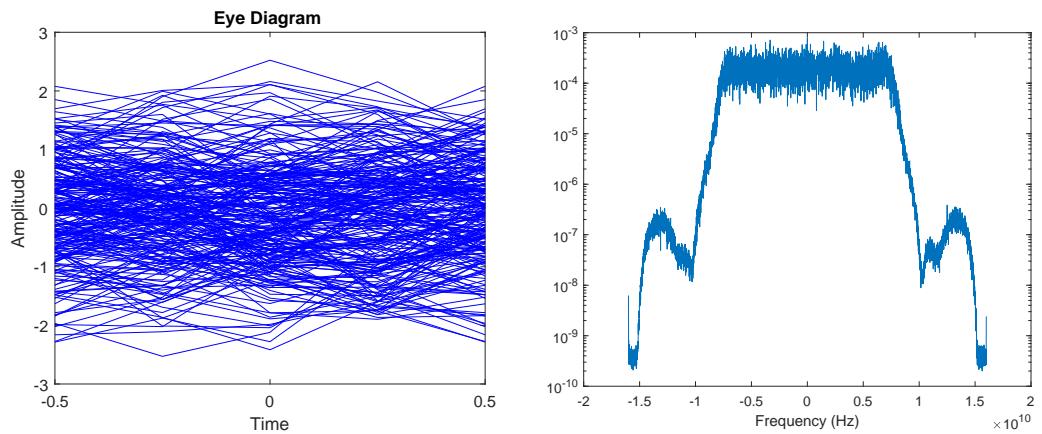


Figure 6.87: Eye Diagram and spectrum after the first stage of adaptive equalization, using a MIMO 2x2 CMA equalizer.

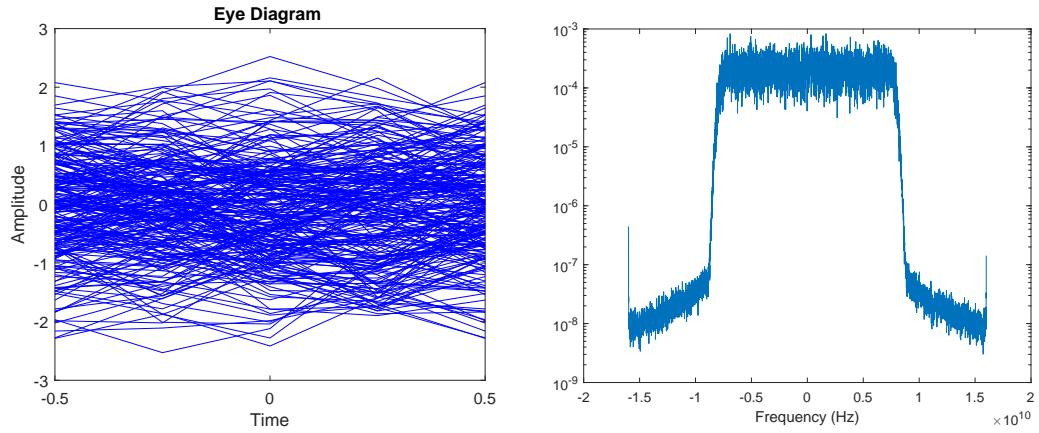


Figure 6.88: Eye Diagram and spectrum after the second stage of adaptive equalization, with a MIMO 4x4 LMS equalizer.

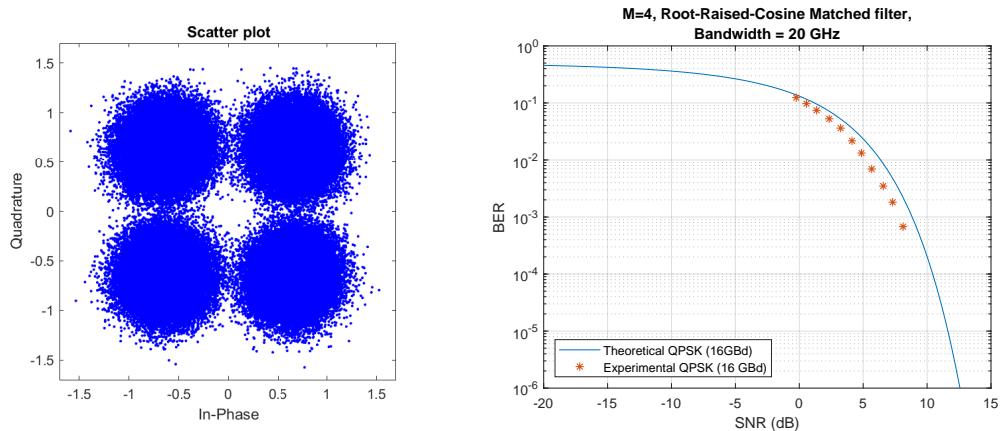


Figure 6.89: Constellation and BER curve as a function of the SNR.

In order to try and disappear with the frequency offset to improve results, the setup was changed to use the same laser beam for both the source and the local oscillator, therefore guaranteeing they have the same frequency.

#### 6.4.4 DSP

#### 6.4.5 Open Issues

## References

- [1] Mischa Schwartz. *Information Transmission, Modulation and Noise*. McGraw-Hill College, 1990. ISBN: 9780070559097.
- [2] A Bruce Carlson. *Communication Systems: An Introducton to Signals and Noise in Electrical Communication*. McGraw Hill, 1986.
- [3] Rolf Matzner. "An SNR estimation algorithm for complex baseband signals using higher order statistics". In: *Facta Universitatis (Nis)* 6.1 (1993), pp. 41–52.

## 6.5 Kramers-Kronig Transceiver

<b>Student Name</b>	:	Romil Patel (16/08/2017 - Cont.)
<b>Goal</b>	:	Develop a simplified structure (low cost) for a coherent transceiver, that can be used in coherent PON, inter-data center connections, or metropolitan networks (optical path lengths < 100 km). We are going to explore a Kramers-Kronig transceiver with Stokes based PolDemux.
<b>Directory</b>	:	LinkPlanner\doc\tex\sdf\kramers_kronig_transceiver

Coherent optical transmission schemes are spectrally efficient since they allow the encoding of information in both quadrature of sinusoid signal. However, the cost of coherent receiver becomes a major obstacle in the case of short-reach links applications like PON, inter-data-center communications and metropolitan network. In order to make the transceiver applicable in short-reach links, an architecture has been proposed which combines the advantages of coherent transmission and cost effectiveness of direct detection. The working principle of the proposed transceiver is based on the Kramers-Kronig (KK) relationship. The KK transceiver scheme allows digital compensation of propagation impairment because both amplitude and phase of the electrical field can be retrieved at the receiver.

### 6.5.1 Theoretical Analysis

The Kramers-Kronig relations are bidirectional mathematical relations, connecting the real and imaginary parts of any complex function that is analytic in the upper half-plane. For instance, a signal  $x(t) = x_r(t) + ix_i(t)$  satisfies the Kramers-Kronig relationship if [1, 2],

$$x_r(t) = -\frac{1}{\pi} p.v. \int_{-\infty}^{\infty} \frac{x_i(t')}{t - t'} dt'$$

$$x_i(t) = \frac{1}{\pi} p.v. \int_{-\infty}^{\infty} \frac{x_r(t')}{t - t'} dt'$$

where *p.v.* stands for *principle value* which is a standard method applied in mathematical applications by which an improper, and possibly divergent, integral is measured in a balanced way around singularities or at infinity [3]. A signal that satisfies the Kramers-Kronig relationship is also named as an analytical signal. This relationship imposes that the real and the imaginary parts of the signal are related to each other through Hilbert transform. Therefore, if we have the real part of the signal then the imaginary part can be calculated by its Hilbert transform.

For a signal that satisfies the Kramers-Kronig relationship, the real and imaginary part can be obtained only from the module. The following questions would give a comprehensive overview of Kramers-Kroning relation and the detailed mathematical calculation which depicts how phase can be extracted from the amplitude information.

### 1. What is the Hilbert transform?

If we consider a filter  $H(\omega)$ , described in Figure 6.90, that has a unity magnitude response for all frequencies and the phase response is  $\pi/2$  for all positive frequencies and  $-\pi/2$  for negative frequencies. The transfer function of this filter is given by

$$H(\omega) = i \operatorname{sgn}(\omega) = \begin{cases} i & \text{for } \omega > 0 \\ -i & \text{for } \omega < 0 \\ 0 & \text{for } \omega = 0 \end{cases} \quad (6.81)$$

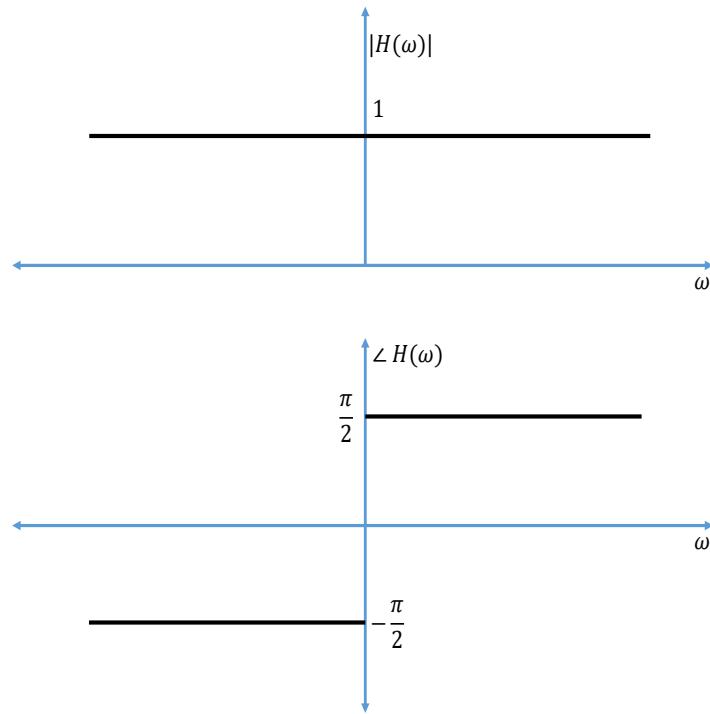


Figure 6.90: Magnitude and phase of Hilbert transform filter

The impulse response of this filter can be given as,

$$\begin{aligned} h(t) &= \mathcal{F}^{-1}[H(i\omega)] \\ &= i\mathcal{F}^{-1}[\operatorname{sgn}(\omega)] \\ &= i\left(\frac{i}{\pi t}\right) \\ &= \frac{-1}{\pi t} \end{aligned} \quad (6.82)$$

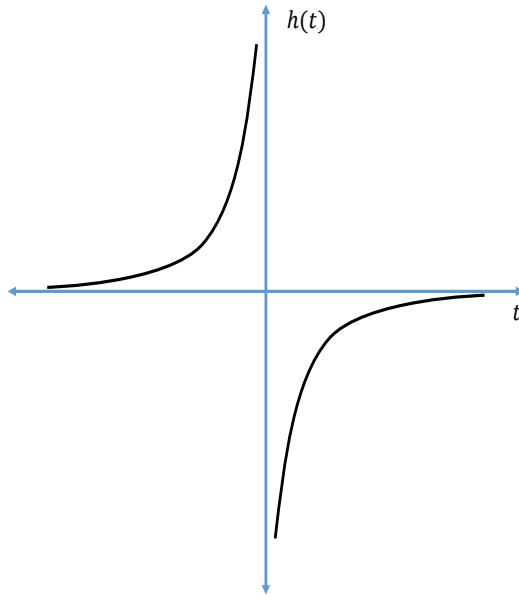


Figure 6.91: Impulse response  $h(t)$  of Hilbert transform filter

where  $F^{-1}$  is the inverse transform. When this filter driven by an arbitrary signal  $s(t)$ , the filter produces the output as,

$$\begin{aligned}\hat{s}(t) &= s(t) \circledast h(t) \\ &= \int_{-\infty}^{\infty} \frac{s(u)}{\pi(t-u)} du\end{aligned}\quad (6.83)$$

The time function  $\hat{s}(t)$  is called the Hilbert transform of  $s(t)$ . Note that

$$\mathcal{F}[\hat{s}(t)] = H(\omega)S(\omega) = i \operatorname{sgn}(\omega)S(\omega) \quad (6.84)$$

which means that the Fourier transform of  $\hat{s}(t)$  equals the Fourier transform of  $s(t)$  multiplied by the factor  $i \operatorname{sgn}(\omega)$ , where  $\operatorname{sgn}(\omega)$  is -1 for negative frequencies and 1 for positive frequencies. In conclusion, if we convolve any time domain signal with  $\frac{1}{\pi t}$  then it will give us Hilbert transformed signal in time domain. Similarly, from the convolution property of the Fourier transform, if we multiply  $i \operatorname{sgn}(\omega)$  with any frequency domain signal  $S(\omega)$  then it'll give us Hilbert transformed signal in frequency domain.

### Example of rectangle function

Consider the rectangular signal  $r(t)$  and its Hilbert transformed  $\hat{r}(t)$  as shown in the Figure 6.92. The Fourier transformed version of the signal  $r(t)$  and  $\hat{r}(t)$  are described as  $R(f)$

and  $\hat{R}(f)$  and their magnitude and the phase spectrum are shown in Figure 6.93 and 6.94, respectively.

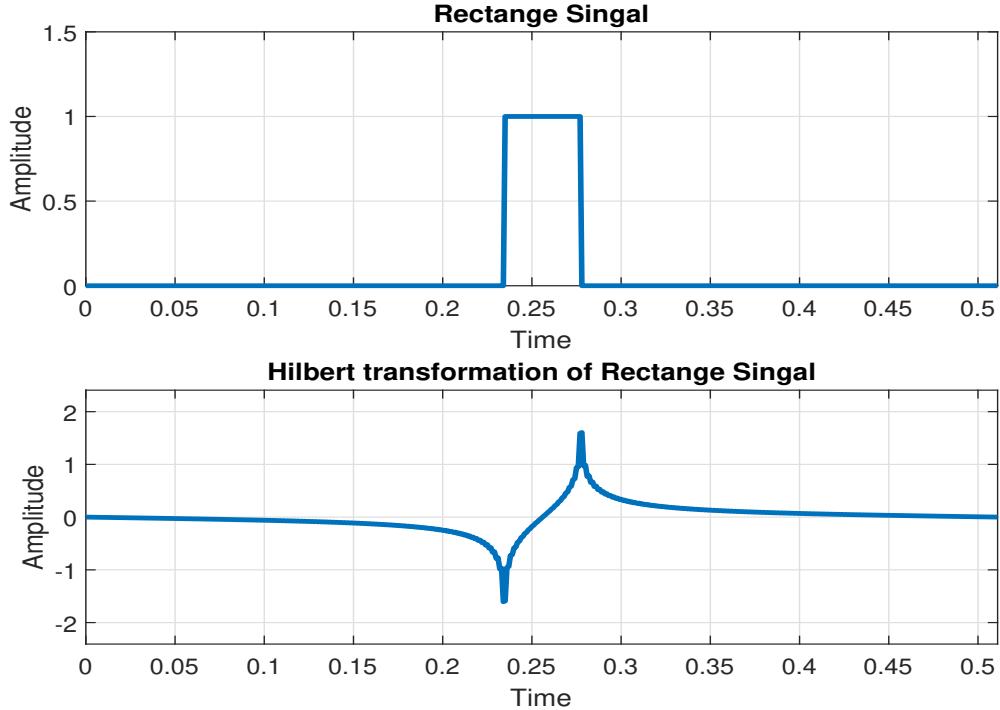


Figure 6.92: Rectangle signal  $r(t)$  and its Hilbert transformed  $\hat{r}(t)$

## 2. What is an analytical signal?

An analytic signal, i.e. a signal that satisfies the KK relationship, is a complex-valued signal that has no negative frequency components, and its real and imaginary parts are related to each other by the Hilbert transform.

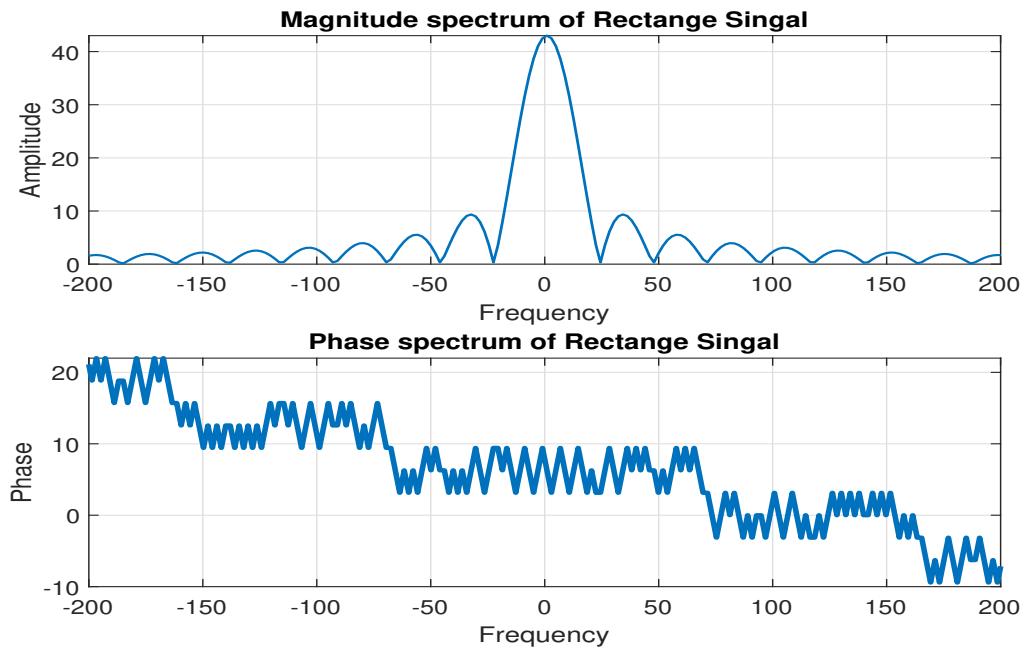
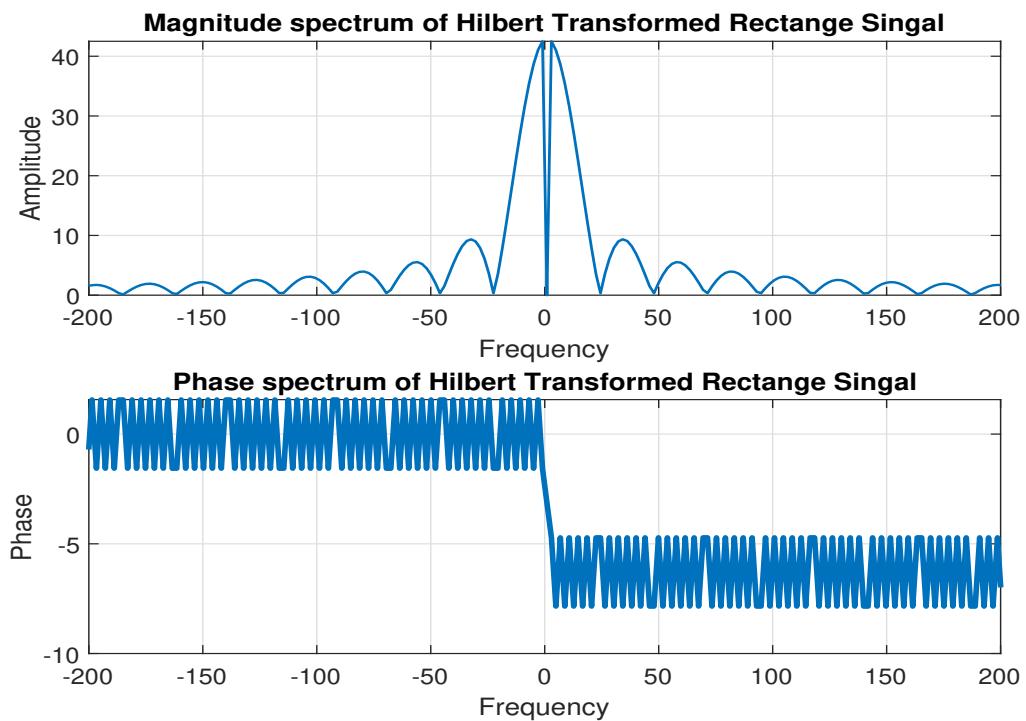
$$s_a(t) = s(t) + i\hat{s}(t) \quad (6.85)$$

where,  $s_a(t)$  is an analytical signal and  $\hat{s}(t)$  is the Hilbert transform of the signal  $s(t)$ . Such analytical signal can be used to generate Single Sideband Signal (SSB) signal. If we denote an analytic signal as,

$$s_a(t) = s_{a,r}(t) + i s_{a,i}(t) \quad (6.86)$$

then in the equation 6.86, the real and imaginary parts  $s_{a,r}(t)$  and  $s_{a,i}(t)$  are related through the Kramers-Kronig relation with each other. An intuitive way to analyze this relation is based on expressing its Fourier transform  $S_a(\omega)$  as follows,

$$S_a(\omega) = \frac{1}{2}[1 + \text{sgn}(\omega)]S_a(\omega) \quad (6.87)$$

Figure 6.93: Magnitude and phase spectrum of  $R(f)$ Figure 6.94: Magnitude and phase spectrum of  $\hat{R}(f)$

The equation 6.87 follows the SSB signal condition  $S_a(\omega) = 0$  for  $\omega < 0$ . Further, simplification of the signal can be summarized as follows:

$$\begin{aligned} S_a(\omega) &= \frac{1}{2}[1 + \text{sgn}(\omega)]S_a(\omega) \\ &= \frac{1}{2}S_a(\omega) + \frac{1}{2}\text{sgn}(\omega)S_a(\omega) \end{aligned} \quad (6.88)$$

Taking inverse Fourier transform of the equation 6.88,

$$\begin{aligned} s_a(t) &= \mathcal{F}^{-1}\{S_a(\omega)\} \\ &= \frac{1}{2}s_a(t) + \frac{1}{2}\underline{[\mathcal{F}^{-1}\{\text{sgn}(\omega)\} \circledast s_a(t)]} \end{aligned} \quad (6.89)$$

The underlined term in Equation 6.89 displays that multiplication in frequency domain converted into the convolution in the time domain. Further, IFT of the function  $\text{sgn}(\omega)$  given as  $(-i/\pi t)$ . As a consequence, we can further simplify our equation as,

$$\begin{aligned} s_a(t) &= \frac{1}{2}s_a(t) + \frac{1}{2}\left[\frac{i}{\pi t} \circledast s_a(t)\right] \\ \frac{s_a(t)}{2} &= \frac{1}{2}\left[\frac{i}{\pi t} \circledast s_a(t)\right] \\ s_a(t) &= i\left[\frac{1}{\pi t} \circledast s_a(t)\right] \\ s_a(t) &= \frac{i}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_a(t')}{t-t'} dt' \end{aligned} \quad (6.90)$$

Using Equation 6.86 into Equation 6.90,

$$s_{s,r}(t) + i s_{a,i}(t) = \frac{i}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_a(t')}{t-t'} dt' \quad (6.91)$$

Therefore,

$$\begin{aligned} s_{s,r}(t) + i s_{a,i}(t) &= \frac{i}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_{a,r}(t') + i s_{a,r}(t')}{t-t'} dt' \\ s_{s,r}(t) + i s_{a,i}(t) &= -\frac{1}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_{a,i}(t')}{t-t'} dt' + \frac{i}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_{a,r}(t')}{t-t'} dt' \end{aligned} \quad (6.92)$$

which leads to,

$$\begin{aligned} s_{a,r}(t) &= -\frac{1}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_{a,i}(t')}{t-t'} dt' \\ s_{a,i}(t) &= \frac{1}{\pi}p.v. \int_{-\infty}^{\infty} \frac{s_{a,r}(t')}{t-t'} dt' \end{aligned} \quad (6.93)$$

In conclusion, the signal with its  $S_a(\omega) = 0$  for  $\omega < 0$  (i.e. analytical in the upper half of the complex plane) satisfies the Kramers-Kronig relationship.

### 3. What is a SSB signal and how it can be generated?

This section will represent the brief idea of generating SSB signal using Hilbert transform method. To understand this, we may express signal  $s(t)$  as a summation of the two complex-valued functions.

$$s(t) = \frac{1}{2}[s(t) + i\hat{s}(t)] + \frac{1}{2}[s(t) - i\hat{s}(t)] \quad (6.94)$$

From Equation 9.15,

$$s(t) = s_a(t) + is_a^*(t) \quad (6.95)$$

where  $s_a^*(t)$  is the complex conjugate of  $s_a(t)$ . Such representation of  $s_a(t)$  and  $s_a^*(t)$  divide the signal into non-negative frequency component and non-positive frequency component respectively. Considering only non-negative frequency  $s_a(t)$  part, we can write it as

$$\frac{1}{2}S_a(f) = \begin{cases} S(f) & \text{for } f > 0 \\ 0 & \text{for } f < 0 \end{cases} \quad (6.96)$$

where  $S_a(f)$  and  $S(f)$  are the Fourier transform of  $s_a(t)$  and  $s(t)$  respectively. The frequency translated version of  $S_a(f - f_0)$  contains only one side (positive) of  $S(f)$  and hence it is called single sideband signal  $s_{ssb}(t)$ ,

$$F^{-1}\{S_a(f - f_0)\} = s_a(t)e^{i2\pi f_0 t} \quad (6.97)$$

Therefore, from the Euler's formula,

$$\begin{aligned} s_{ssb}(t) &= Re\{s_a(t)e^{i2\pi f_0 t}\} \\ &= Re\{[s(t) + i\hat{s}(t)][\cos(2\pi f_0 t) + i\sin(2\pi f_0 t)]\} \\ &= s(t)\cos(2\pi f_0 t) - \hat{s}(t)\sin(2\pi f_0 t) \end{aligned} \quad (6.98)$$

This Equation 6.98 displays the mathematical modeling of the upper sideband SSB signal. Similarly, we can generate lower sideband SSB signal by,

$$s_{ssb}(t) = s(t)\cos(2\pi f_0 t) + \hat{s}(t)\sin(2\pi f_0 t) \quad (6.99)$$

### Graphical explanation SSB signal generation

This section describes the generation of SSB signal using Hilbert transformation method (Phase Shift Method). Consider a message signal  $m(t)$  with its frequency domain spectrum  $|M(F)|$  as shown in Figure 6.95. From the Figure 6.95, we can see that both the side are scaled by factor '1' which means it represents the original signal.

Now let's consider the modulated signal  $x(t)$  given as,

$$x(t) = m(t)\cos(2\pi f_c t) \quad (6.100)$$

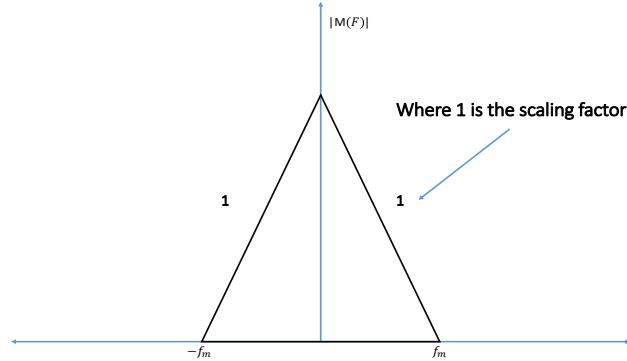
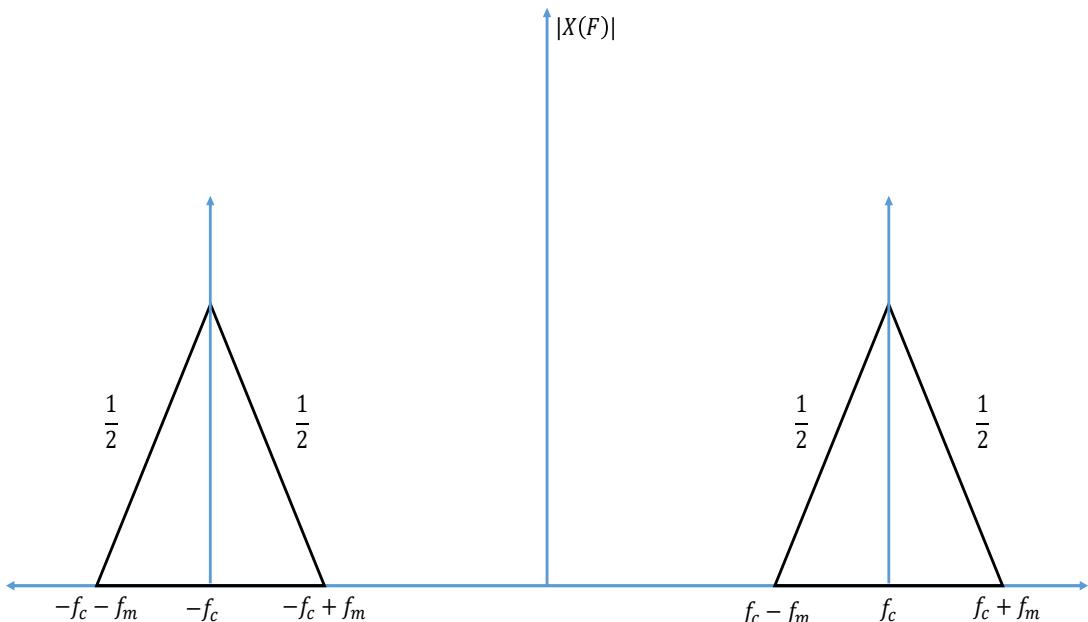


Figure 6.95: Original baseband signal

Frequency domain representation (see Figure 6.96) of the equation 6.100 can be given as,

$$X(F) = \frac{1}{2}M(f - f_c) + \frac{1}{2}M(f + f_c) \quad (6.101)$$

Here in equation 6.101, we can observe that each side band are scaled by  $\frac{1}{2}$  on the frequency spectrum. Figure displays the frequency domain representation of the modulated signal  $X(F)$ .

Figure 6.96: Frequency spectrum of modulated signal  $X(F)$ 

Next, we will discuss something more interesting which is called as Hilbert transform of the

original message signal  $m(t)$ . As we discussed earlier, in the frequency domain, the Hilbert transformed signal  $\hat{M}(f)$  can be achieved by multiplying the Fourier transformed signal  $M(F)$  with  $[-i \operatorname{sgn}(F)]$  (see Figure 6.97). Suppose we modulate the Hilbert transformed

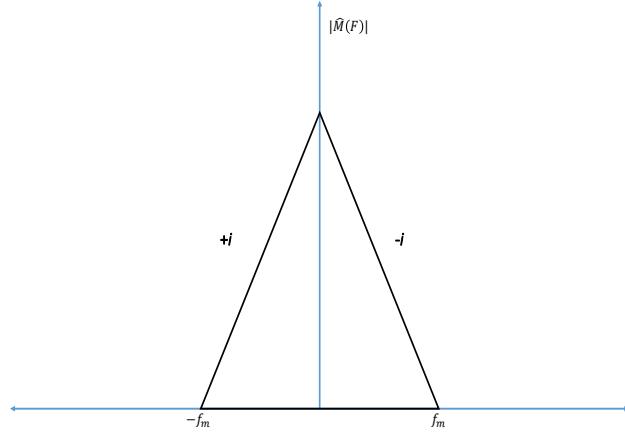


Figure 6.97: Spectrum of Hilbert transformed message signal  $\hat{M}(F)$

message signal  $\hat{m}(t)$  with the  $\sin(2\pi f_c t)$  (quadrature phase carrier), then we get the following results:

$$\begin{aligned}
 \hat{m}(t) \sin(2\pi f_c t) &= \hat{m}(t) \frac{e^{i2\pi f_c t} - e^{-i2\pi f_c t}}{2} \\
 &= \hat{m}(t) \frac{e^{i2\pi f_c t}}{2} - \hat{m}(t) \frac{e^{-i2\pi f_c t}}{2} \\
 &= \frac{\hat{M}(f - f_c)}{2i} - \frac{\hat{M}(f + f_c)}{2i} \\
 &= \frac{-i}{2} \hat{M}(f - f_c) + \frac{i}{2} \hat{M}(f + f_c)
 \end{aligned} \tag{6.102}$$

The detailed explanation of the equation 6.102 has been given in the Figure 6.98 and 6.99. Figure 6.98 displays the spectrum of the  $\hat{M}(f + f_c)$  and  $\hat{M}(f - f_c)$  for the positive and negative frequencies respectively. The final equation resolution of equation displays that both positive and negative side of the spectrum multiplied with  $\frac{i}{2}$  and  $\frac{-i}{2}$  respectively. Finally the spectrum of the signal  $\hat{m}(t) \sin(2\pi f_c t)$  can be given as Figure 6.99.

Further, summation of the two signals  $m(t) \cos(2\pi f_c t)$  and  $\hat{m}(t) \sin(2\pi f_c t)$  will generate the upper sideband SSB signal as follows,

$$u(t) = m(t) \cos(2\pi f_c t) - \hat{m}(t) \sin(2\pi f_c t) \tag{6.103}$$

From the above discussion, the spectrum of the Equation 6.103 can be given by the Figure 6.100. Similarly, for the lower sideband SSB can be generated by Equation,

$$u(t) = m(t) \cos(2\pi f_c t) + \hat{m}(t) \sin(2\pi f_c t) \tag{6.104}$$

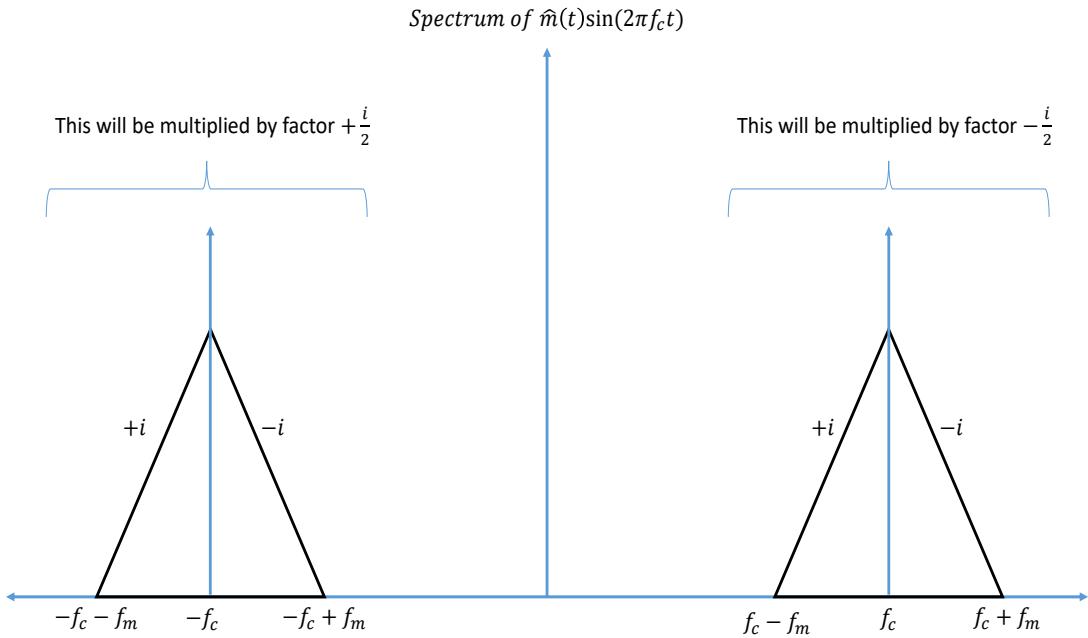


Figure 6.98: Hilbert transformed modulated signal

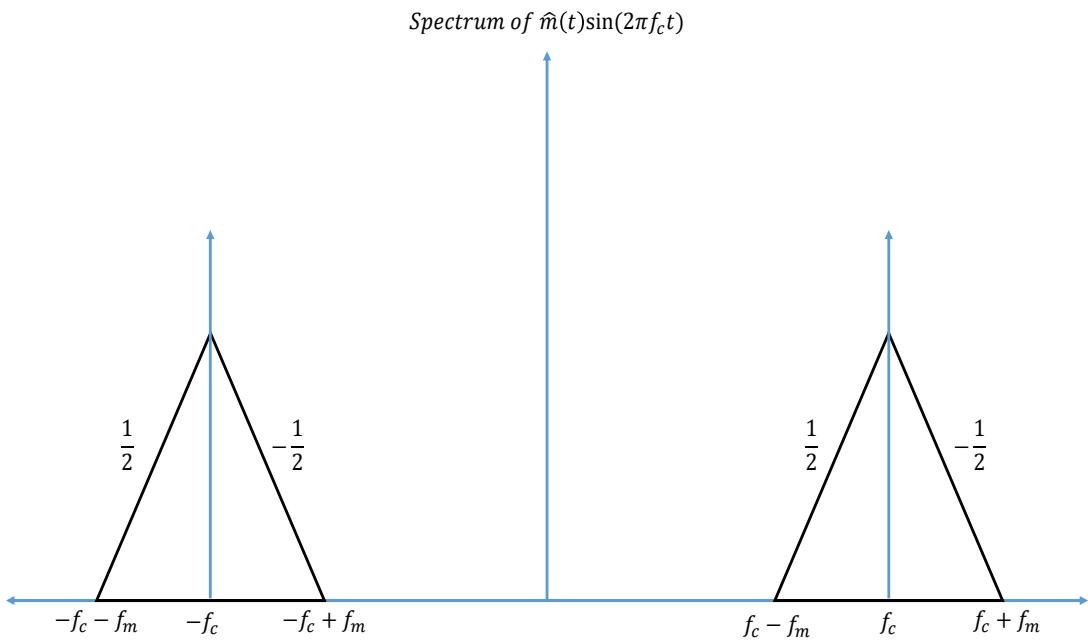


Figure 6.99: Hilbert transformed modulated signal

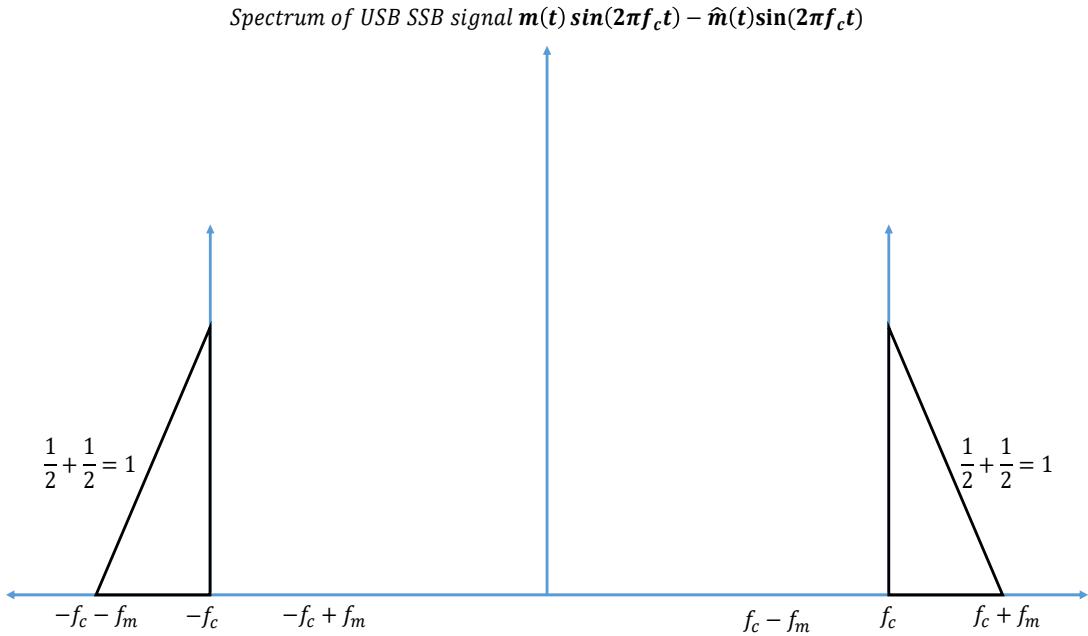


Figure 6.100: SSB signal spectrum

#### 4. What is minimum phase signal how we can profit from it?

A necessary and sufficient condition for a complex signal  $E(t)$  to be minimum phase is that the curve described in a complex plane by  $A(t)$  when  $t \rightarrow -\infty$  to  $t \rightarrow \infty$  does not encircle the origin [4]. A minimum-phase signal has an useful property that the natural logarithm of the magnitude of the frequency response is related to the phase angle of the frequency response by the Hilbert transform.

For instance, if we consider a complex data-carrying signal  $E_s(t)$  with its optical bandwidth  $B$  and the LO is assumed to be a continuous-wave (CW) signal with its amplitude of  $E_o$  and the frequency which coincides with the left edge of the information-carrying signal spectrum and the constructed signal  $E(t)$  as,

$$E(t) = E_o \exp(i\pi Bt) + E_s(t) \quad (6.105)$$

where  $E_o$  is a constant; Here,  $E(t)$  is minimum phase if and only if the winding number of its trajectory into complex plane is zero i.e. the representation of the signal in the complex plane do not circle the origin. The condition  $|E_o| > |E_s(t)|$  is sufficient for guaranteeing minimum phase property [3]. Therefore, when  $E(t)$  is minimum phase signal, its phase  $\phi(t)$  can be reconstructed from its magnitude  $|E(t)|$  by means using Kramers-Kronig relationship. In other words,  $E_o$  should be large enough to ensure that the signal presented by Equation 6.106 becomes a minimum phase signal.

$$E(t) \exp(-i\pi Bt) = E_o + E_s(t) \exp(i\pi Bt) \quad (6.106)$$

Once the minimum phase condition is satisfied for the received optical signal then its phase can be constructed using its intensity and the original signal can be reconstructed in the following manner.

$$E_s(t) = \left[ \sqrt{I} \exp[i\phi_E(t)] - E_o \right] \exp(i\pi Bt) \quad (6.107)$$

$$\phi_E(t) = \frac{1}{2\pi} p.v. \int_{-\infty}^{\infty} dt' \frac{\log[I(t')]}{t - t'} \quad (6.108)$$

### 6.5.2 MATLAB Simulation

#### Case I : 16-QAM signal

Consider a 16-QAM complex signal  $E_s(t) = I(t) + jQ(t)$  generated using the simulator as shown in Figure 6.101. The detail of the generated signal  $I(t)$  and  $Q(t)$  are given in the table.

Parameter	Value
bitPeriod	1.0 / 30e9
numberOfBits	1000
numberOfSamplesPerSymbol	16
rollOffFactor	0.3

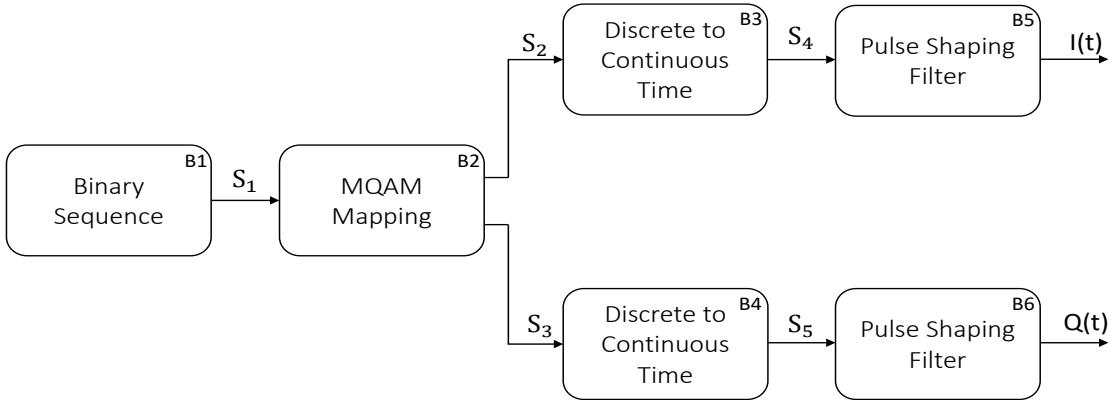


Figure 6.101: set-up to generate 16-QAM modulation signal

The constellation diagram and the magnitude spectrum of the signal are shown in the Figure 6.103a and 6.103b respectively. The signal  $E_s(t)$  is confined within the bandwidth of  $\sim 10.5$  GHz. A continuous wave (CW) tone is required at the edge (either lower or higher) of the information bandwidth to make the signal to be a minimum phase signal. In this example, a

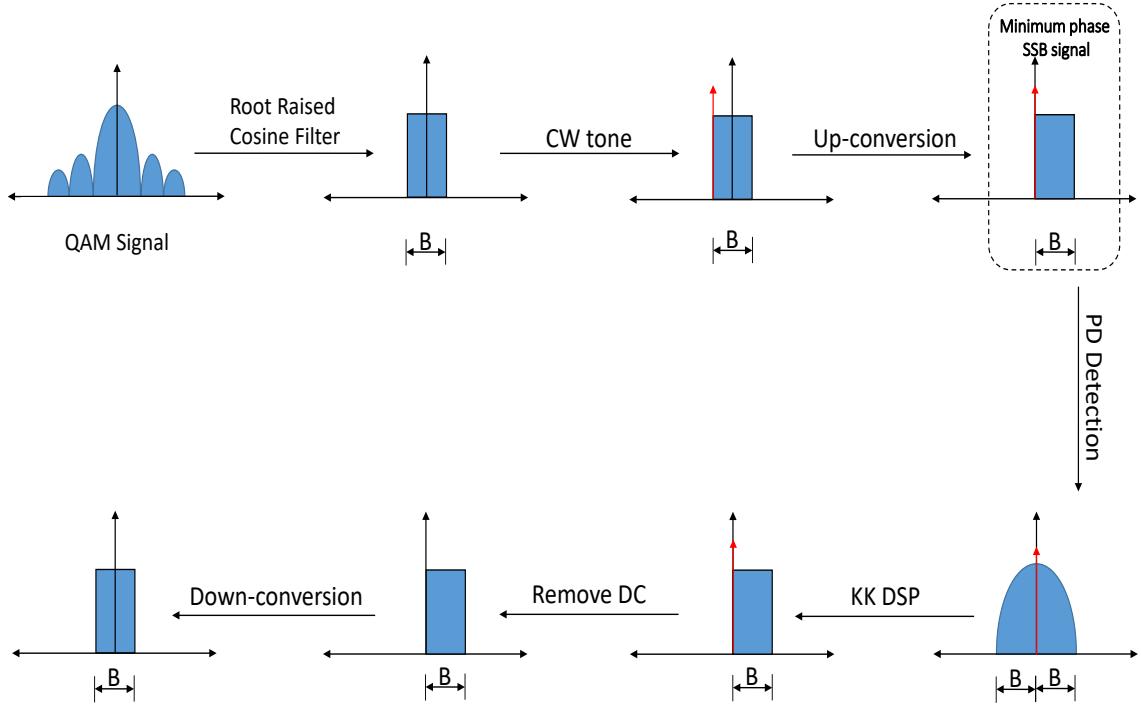
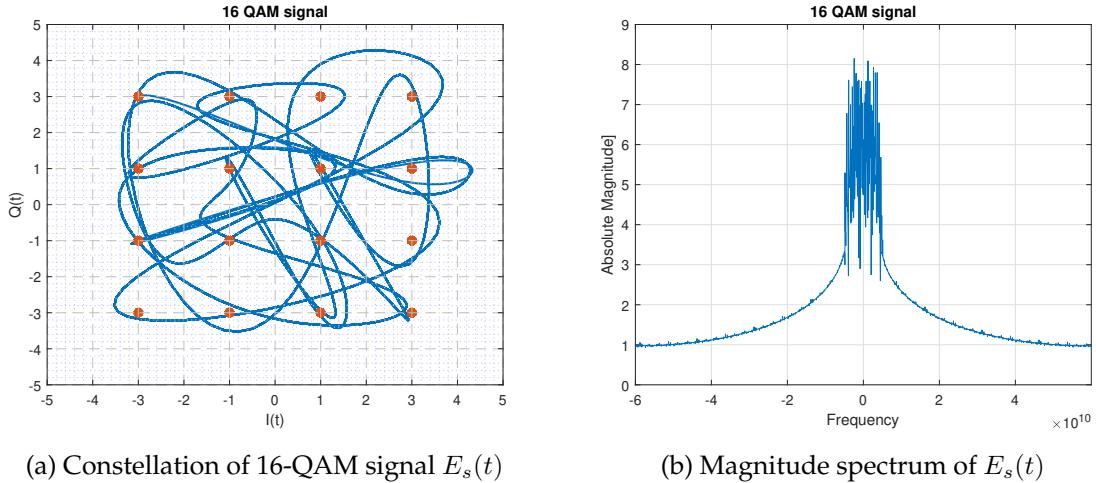


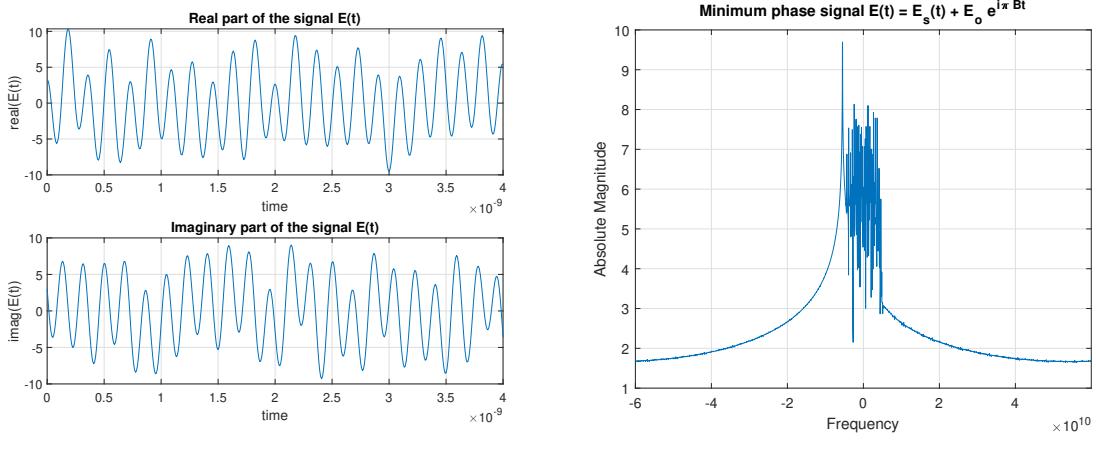
Figure 6.102: Generate minimum phase SSB of QAM signal

Figure 6.103: 16-QAM signal  $E_s(t) = I(t) + iQ(t)$ 

CW tone has been added at the lower edge of the information bandwidth as shown In Figure 6.104b. The mathematical model of the CW tone added signal can be written as,

$$E(t) = E_s(t) + E_o e^{-i\pi B t} \quad (6.109)$$

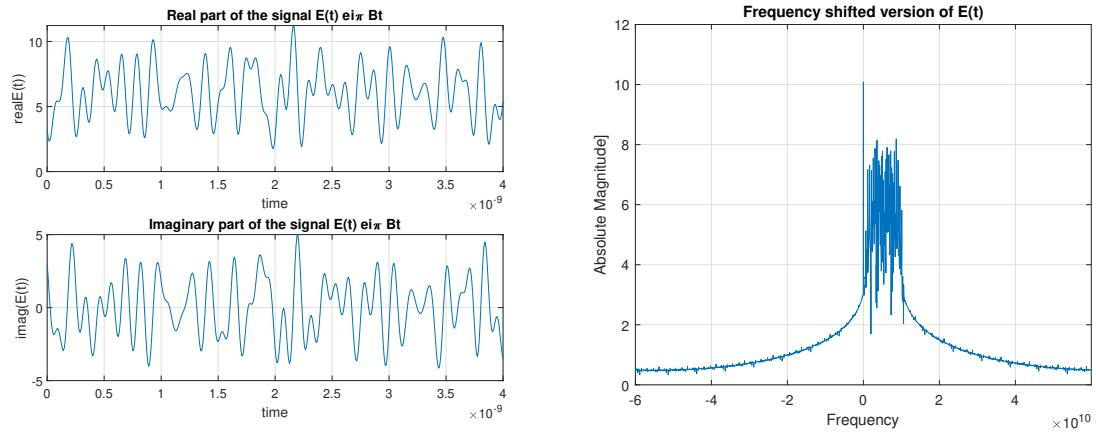
where the frequency of the CW is -5.5 GHz (half of the information bandwidth) and amplitude  $E_o \geq |E_s(t)|$ .

(a) Real and imaginary part of the signal  $E(t)$ (b) Magnitude spectrum of  $E(t)$ Figure 6.104: 16-QAM minimum phase signal  $E(t) = E_s(t) + E_o e^{-i\pi Bt}$ 

The frequency shifted version (frequency up-converted signal) of the minimum phase signal can be written in the following form.

$$[h]E(t) e^{i\pi Bt} = E_s(t) e^{i\pi Bt} + E_o \quad (6.110)$$

Figure 6.105 displays that the real part of the frequency shifted signal is non-negative. As long as this non-negativity of the real part is preserved, the signal can be fully recovered from its intensity. Therefore, if we sample the optical signal with the sampling frequency equals to the twice of the information signal bandwidth then we can achieve the resulting spectrum similar to 6.105b and we can recover the full spectrum using its magnitude value.

(a) Real and imaginary part of the signal  $E(t) e^{i\pi Bt}$ (b) Magnitude spectrum of  $E(t) e^{i\pi Bt}$ Figure 6.105: Frequency shifted version of  $E(t)$  as  $E(t) e^{i\pi Bt} = E_s(t) e^{i\pi Bt} + E_o$

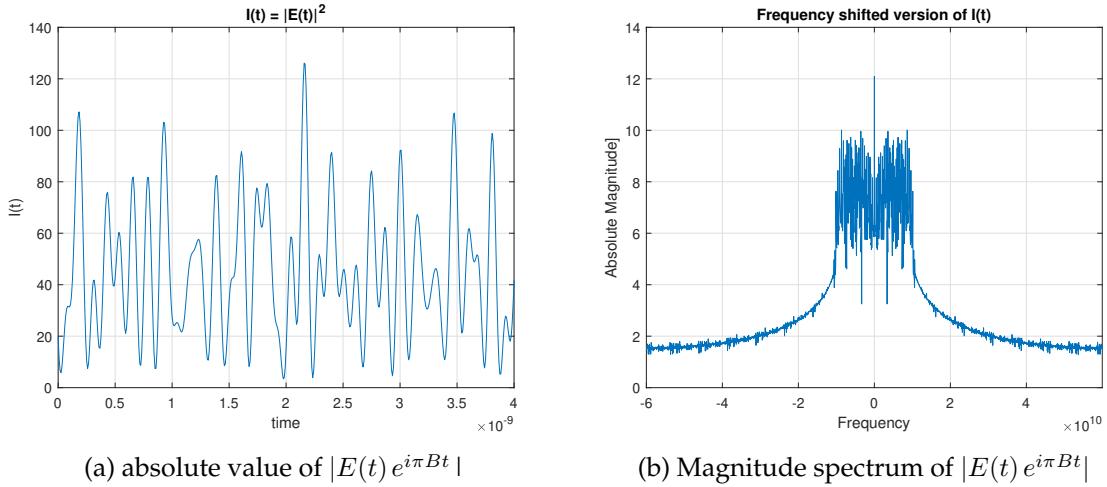


Figure 6.106: Frequency shifted minimum phase signal  $|E(t) e^{i\pi Bt}| = \sqrt{|E_s(t) e^{i\pi Bt} + E_o|^2}$

Next, if we take the square of modulus of the signal as  $|E(t) e^{i\pi Bt}|^2$  then the resultant output is as shown in Figure 6.106. By employing Kramers-Kronig algorithm on the  $|E(t) e^{i\pi Bt}|^2$ , we can recover the full  $E(t) e^{i\pi Bt}$  as shown in Figure 6.107. Next, the signal is down-converted (see Figure 6.108) and then the CW tone has been removed (see Figure 6.109) to recover the original complex signal  $E_s(t) = I(t) + iQ(t)$ .

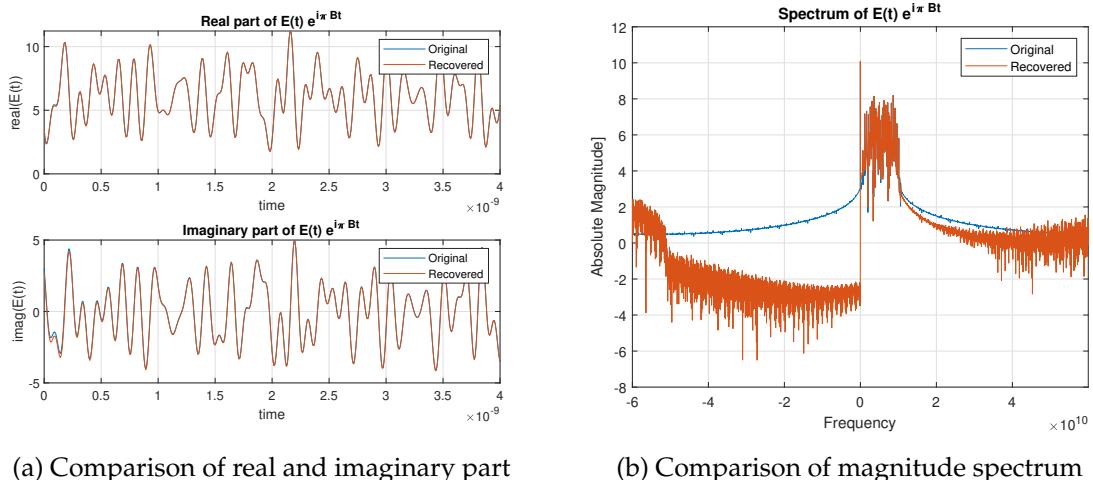
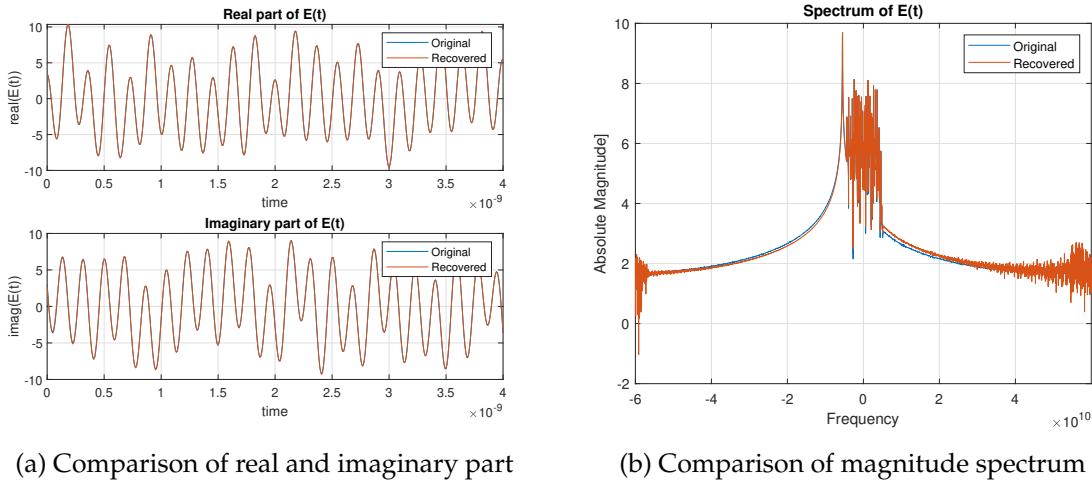
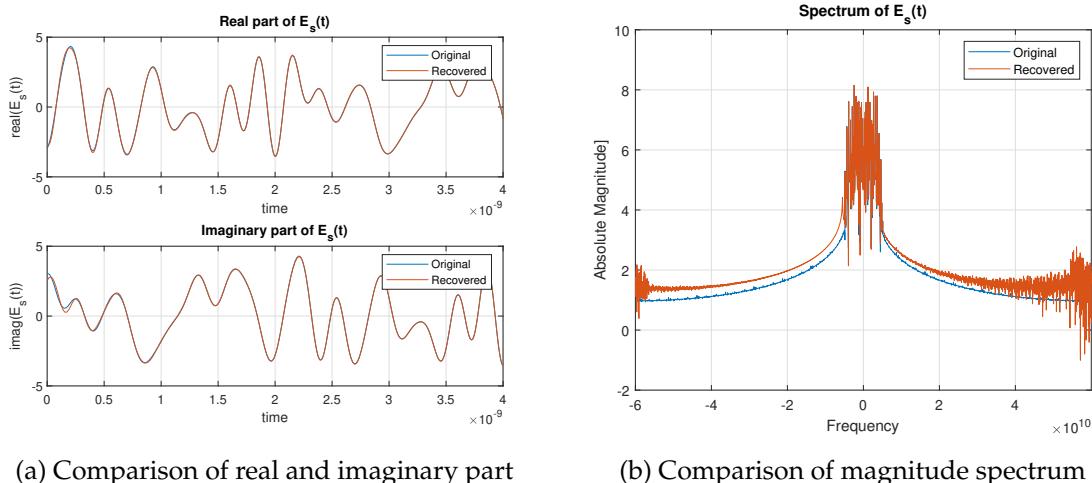


Figure 6.107: Comparison between original and recovered  $E(t) e^{i\pi Bt}$

Figure 6.108: Comparison between original and recovered  $E(t) e^{i\pi Bt}$ Figure 6.109: Comparison between original and recovered  $E_s(t)$ 

### Generalized KK transceiver for complex signal

The generalized architecture of the KK transmitter and receiver is shown in the Figure 6.110 and 6.111 respectively. The transmitter consists of a laser and an IQ modulator fed by two DACs for data generation and modulation. The CW tone can be generated either electronically or optically, however, in this setup we choose to generate the tone optically. In order to generate a tone, we split a portion of unmodulated laser carrier and pass it through the frequency shifter ( $\Delta f$ ) and added it with the IQ modulated signal as shown in the Figure 6.110. The CW tone does not need to be synchronized with the signal in order work with the KK algorithm.

At the receiver end, the signal is detected by direct detection method and full complex envelope recovered by employing the Kramers-Kronig algorithm. The dotted box in the Figure 6.111 displays the DSP of the kramres-Kronig algorithm. First the square root data of

the photo-detected signal is calculated and then converted into the frequency domain after computing its logarithmic value. In the frequency domain, we can most conveniently apply the Hilbert transformation algorithm. Here,  $\text{sgn}(\omega)$  is the sign function, which is equal to 1 for  $\omega > 0$ , and 0 for  $\omega = 0$  and -1 for  $\omega < 0$ . At the end, the Hilbert transformed signal is converted into the time domain and then its exponential value calculated to get the phase information of the detected signal. Finally, we can recover the full complex signal and applied to the post processing section to recover the the data.

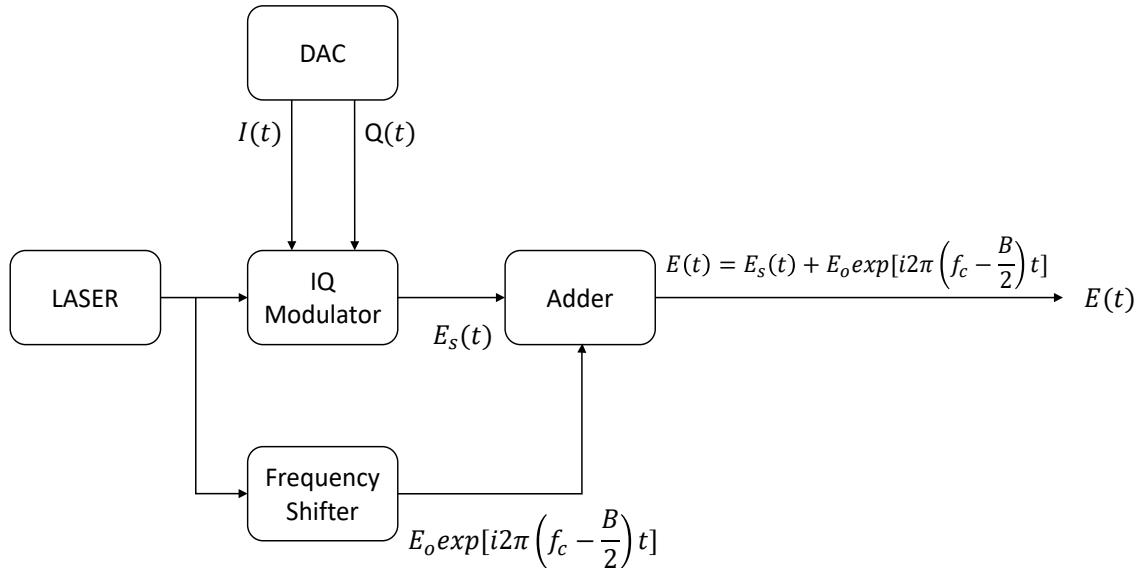


Figure 6.110: Transmitter setup

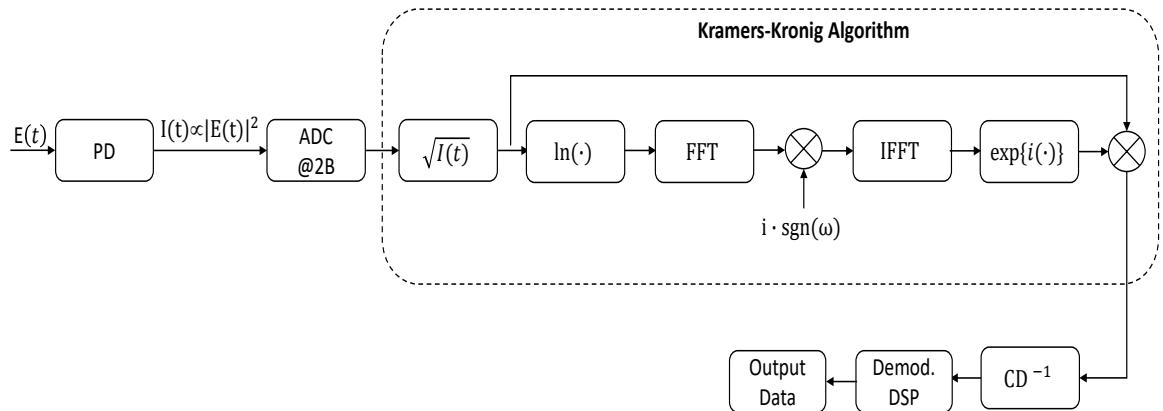


Figure 6.111: Receiver setup

### Case II : PAM signal

In case of PAM signal, we can use the idea displayed in the Figure 6.113 to generate minimum phase SSB signal. The figure displays the idea of generating SSB signal using the Hilbert transformation method. The same thing can be achieved by using the SSBOF (Single Side Band Optical Filter) which makes use of the simplicity of amplitude modulation instead of using IQ modulator in the Hilbert filter method. Therefore, in the real-valued signal, a less complex configuration has been recently proposed [5] which facilitates to generating minimum phase SSB signal (see Figure 6.120).

Consider a 4-PAM real signal  $E_s(t) = I(t)$  generated using the simulator as shown in Figure 6.112. The detail of the signal  $I(t)$  is given in the following table.

Parameter	Value
bitPeriod	1.0 / 30e9
numberOfBits	1000
numberOfSamplesPerSymbol	16
rollOffFactor	0.3

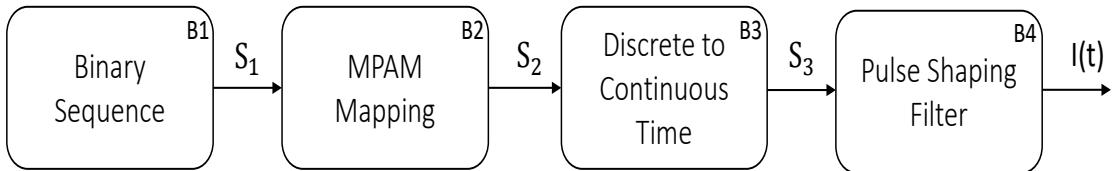


Figure 6.112: set-up to generate 4-PAM modulation signal

The baseband signal  $E_s(t) = I(t)$  and its magnitude spectrum is displayed as shown in Figure 6.114a and 6.114b respectively. The signal is confined within the bandwidth of approximately  $\sim 22$  GHz. A DC bias is applied to the signal  $E_s(t)$  to ensure the signal becomes non-negative. The mathematical model of the signal can be written as,

$$E(t) = E_s(t) + E_o \quad (6.111)$$

where  $E_o$  is the DC bias which should be large enough to ensure that the signal should become non-negative as shown in Figure 6.115. Next, this signal has been passed through the Hilbert filter which generates the minimum phase analytical SSB signal (Practically same thing can be achieved by using SSBOF) as shown in Figure 6.116. This SSB minimum phase signal  $E_{mp}(t)$  is launched into the optical fiber for the transmission. At the receiver end, a photo detector will detect the intensity of the signal  $|E_{mp}(t)|^2$  (see Figure 6.117). By

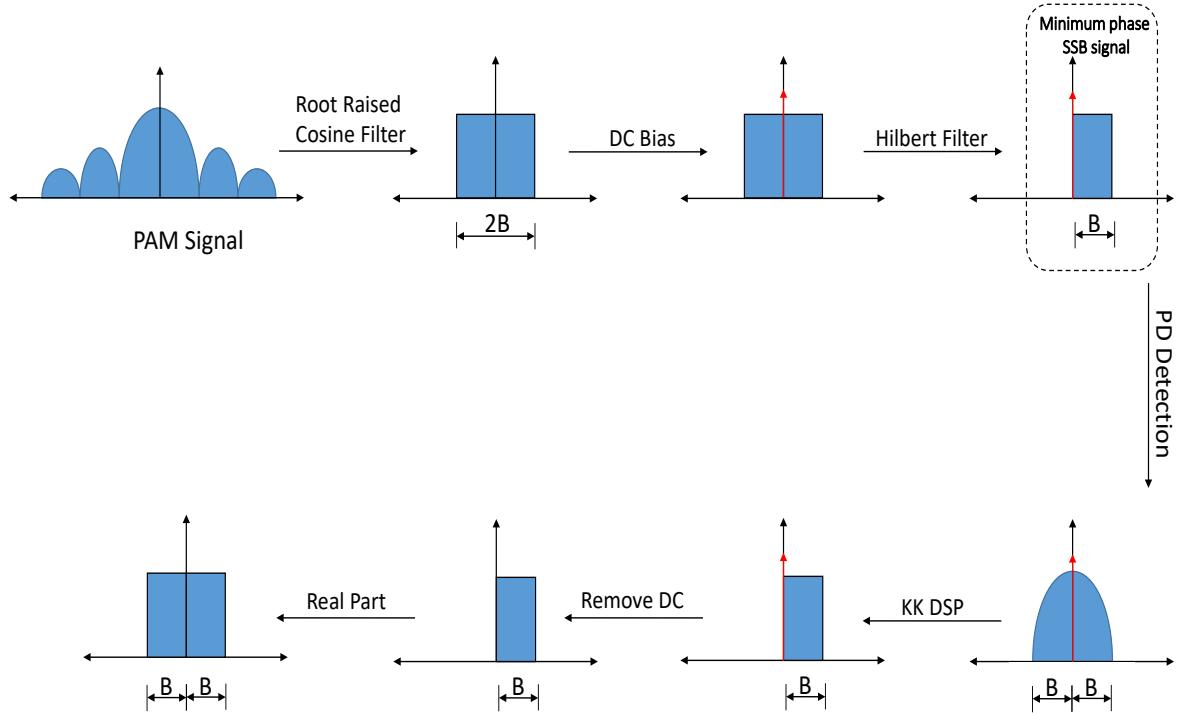
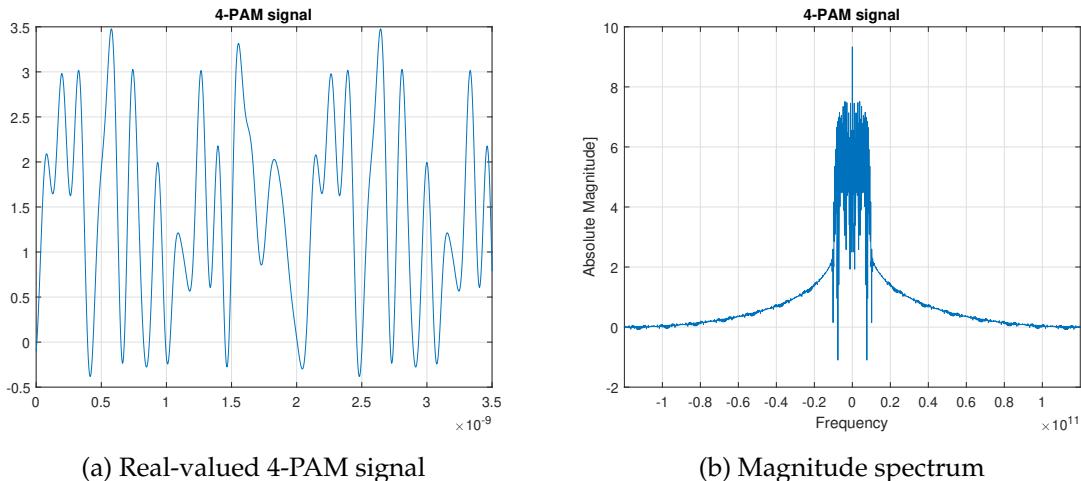
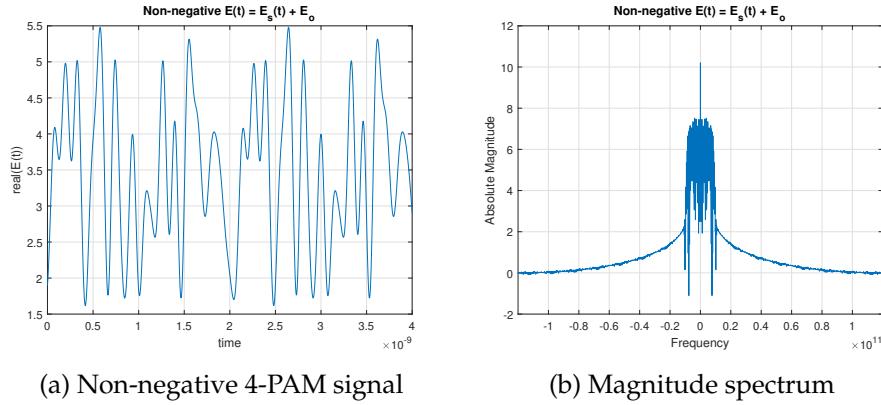
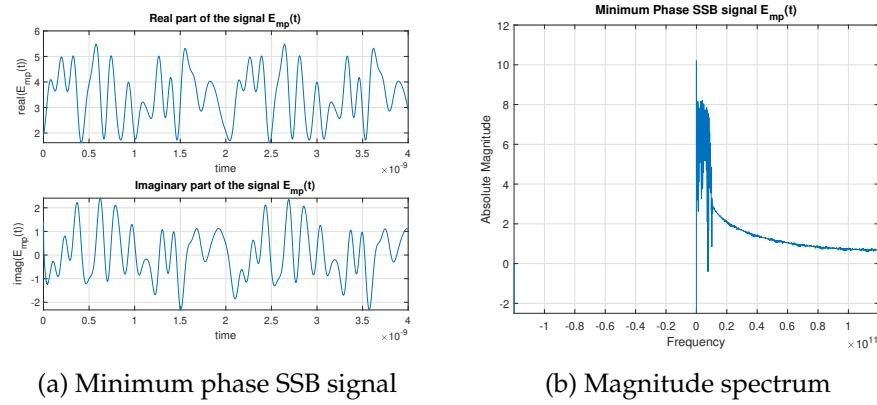


Figure 6.113: Generate minimum phase SSB of PAM signal

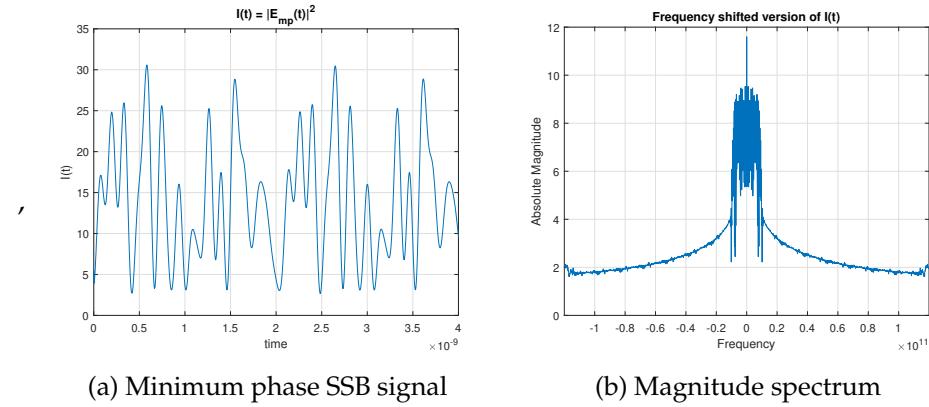
Figure 6.114: 4-PAM signal  $E_s(t) = I(t)$ 

employing the Kramers-Kronig algorithm on the  $|E_{mp}(t)|^2$ , we can recover the full complex signal  $E_{mp}(t)$  as shown in Figure 6.118. From the recovered complex signal, we can select only real part and removing the DC bias content, we can recover the signal  $I(t)$  as shown in Figure 6.119

Figure 6.115: Non-negative real-valued information signal  $E(t)$ Figure 6.116: SSB minimum phase signal  $E_{mp}(t)$ 

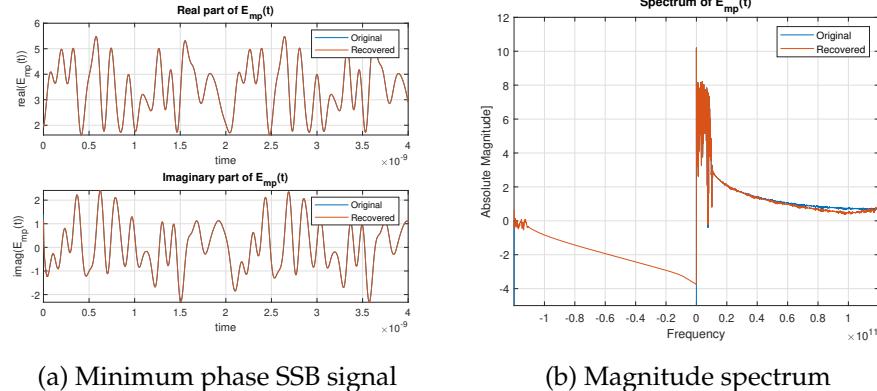
### Generalized KK transceiver for real-valued signal

In order to extract the benefits of the KK receiver, the experimental setup discussed in Figure 6.110 relied on expensive transmitter using IQ modulators. In the quest for the cost-effective and highly performing transmission solutions, a simplified transmitter setup has been proposed which helps to transmit the real-valued signal for the KK transceiver [5]. The experimental setup for the real-valued signal is shown in the Figure 6.120 which consists of a laser followed by a single-drive MZM and a sharp optical filter (OF) suppressing the one of the optical sidebands. The modulator is driven by an arbitrary waveform generator or we can feed any real-valued signal, i.e. an RF signal to it. The MZM bias level was set in the vicinity of the middle between the quadrature and null-points. The driving signal amplitude was adjusted so that it never cross the MZM null-point, thus preventing the optical field from assuming negative values. The receiver architecture remains same like as we discussed in Figure 6.111.



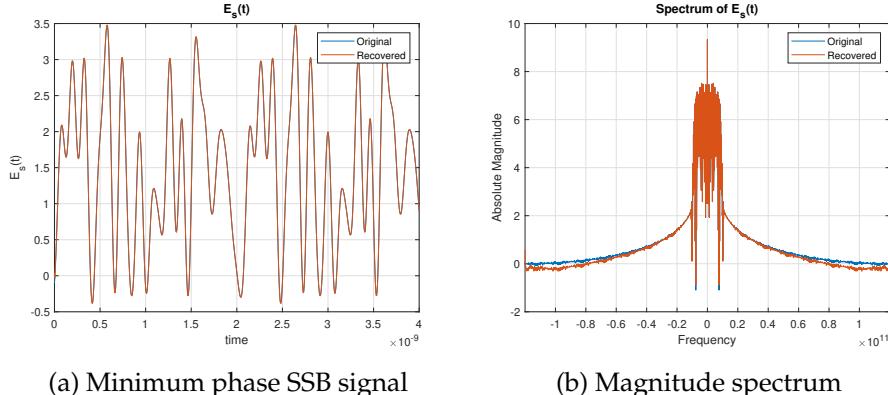
(a) Minimum phase SSB signal

(b) Magnitude spectrum

Figure 6.117: Detected SSB minimum phase signal  $I(t) = |E_{mp}(t)|^2$ 

(a) Minimum phase SSB signal

(b) Magnitude spectrum

Figure 6.118: Recovered non-negative signal  $E_{mp}(t)$ 

(a) Minimum phase SSB signal

(b) Magnitude spectrum

Figure 6.119: Comparison between original and recovered signal  $E_s(t)$

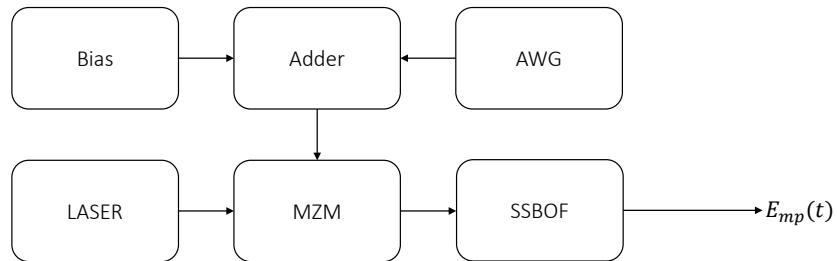


Figure 6.120: Simplified transmitter setup for the real-valued signal

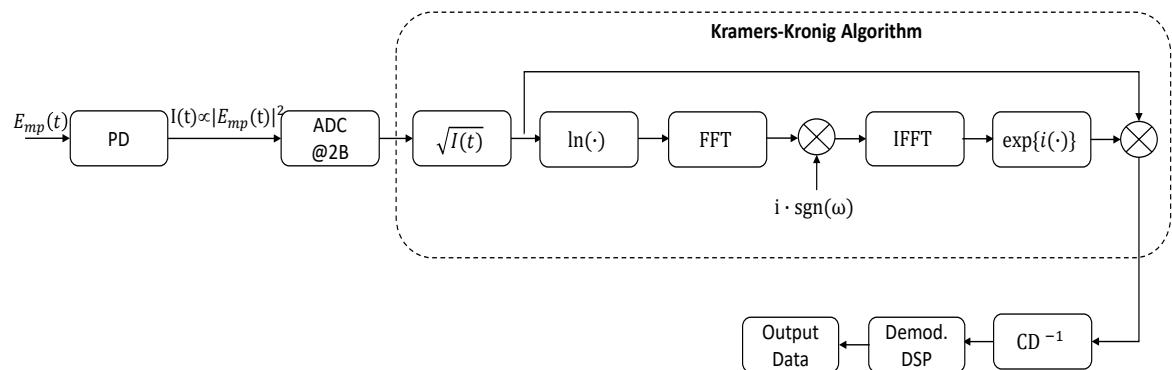


Figure 6.121: Receiver setup for the real-valued signal

## References

- [1] Antonio Mecozzi, Cristian Antonelli, and Mark Shtaif. "Kramers-Kronig coherent receiver". In: *Optica* 3.11 (Nov. 2016), p. 1220. ISSN: 2334-2536. DOI: [10.1364/OPTICA.3.001220](https://doi.org/10.1364/OPTICA.3.001220). URL: <https://www.osapublishing.org/abstract.cfm?URI=optica-3-11-1220>.
- [2] Antonio Mecozzi. "Retrieving the full optical response from amplitude data by Hilbert transform". In: *Optics Communications* 282.20 (Oct. 2009), pp. 4183–4187. ISSN: 0030-4018. DOI: [10.1016/j.optcom.2009.07.025](https://doi.org/10.1016/j.optcom.2009.07.025). URL: <https://www.sciencedirect.com/science/article/pii/S0030401809006907>.
- [3] Matilde Legua and Luis Sánchez-Ruiz. "Cauchy Principal Value Contour Integral with Applications". In: *Entropy* 19.5 (May 2017), p. 215. ISSN: 1099-4300. DOI: [10.3390/e19050215](https://doi.org/10.3390/e19050215). URL: <http://www.mdpi.com/1099-4300/19/5/215>.
- [4] Antonio Mecozzi. "A necessary and sufficient condition for minimum phase and implications for phase retrieval". In: *IEEE TRANSACTIONS ON INFORMATION THEORY* 13.9 (2014). URL: <https://pdfs.semanticscholar.org/1ae2/690a2a435f94b74320d14f135f3e4928f08a.pdf>.
- [5] M. Presi et al. "Transmission in 125-km SMF with 3.9 bit/s/Hz spectral efficiency using a single-drive MZM and a direct-detection Kramers-Kronig receiver without optical CD compensation". In: *Optical Fiber Communication Conference*. Washington, D.C.: OSA, Mar. 2018, Tu2D.3. ISBN: 978-1-943580-38-5. DOI: [10.1364/OFC.2018.Tu2D.3](https://doi.org/10.1364/OFC.2018.Tu2D.3). URL: <https://www.osapublishing.org/abstract.cfm?URI=OFC-2018-Tu2D.3>.

## 6.6 DSP Laser Phase Noise Compensation

<b>Student Name</b>	:	Celestino Martins (08-01-2018 - )
<b>Goal</b>	:	DSP algorithms for laser phase noise compensation applied in optical coherent receiver systems.
<b>Directory</b>	:	sdf/dsp_laser_phase_compensation

The increasing digital traffic data demand in the core networks and the advances in digital signal processing (DSP) is pushing the deployment of coherent optical technology into optical networks. Based on this approach higher spectral efficient modulation formats can be used to increase the systems data rate, such as M-QAM. However, high order modulations are more susceptible to the different systems impairments such as chromatic dispersion (CD), polarization mode dispersion (PMD), fibre nonlinearities and laser phase noise. Laser phase noise is one of the fundamental impairments in coherent optical systems, because it can severely limit the synchronization between transmitter and receiver in demodulation and detection of the transmitted data. Since, in these modulation formats the data is encoded in the amplitude and phase of an optical carrier, an accurate carrier phase recovery is required.

In coherent optical systems, carrier phase recovery has been performed primarily in the electrical domain as part of the DSP, using both feedforward and feedback-based algorithms. Nevertheless, the research experiments have shown that feedforward carrier phase recovery schemes are more tolerant to laser phase and facilitate the parallel implementation in a hardware unit, owing to the high parallelization and pipelining required in a real ASIC implementation.

### 6.6.1 Theoretical Analysis

Generally, laser phase noise can be modeled as a Wiener process [1] described as,

$$\phi(k) = \phi(k-1) + \Delta\phi(k), \quad (6.112)$$

where,  $\Delta\phi(k)$  is an independent and identically distributed random Gaussian variable with zero mean and variance given as,

$$\sigma^2 = 2\pi(\Delta f T), \quad (6.113)$$

where  $\Delta f$  corresponds to the sum of linewidth of the signal and local oscillator lasers and  $T$  is the symbol period.

Typically, laser phase noise compensation for coherent optical receivers is performed by feedforward algorithms based on the well-known Viterbi-Viterbi (VV) algorithm [2, 1] or blind phase search algorithm (BPS) [3].

#### Viterbi-Viterbi Algorithm

VV algorithm is a n-th power feed-forward approach employed for uniform angular distribution characteristic of m-PSK constellations, where the information of the modulated

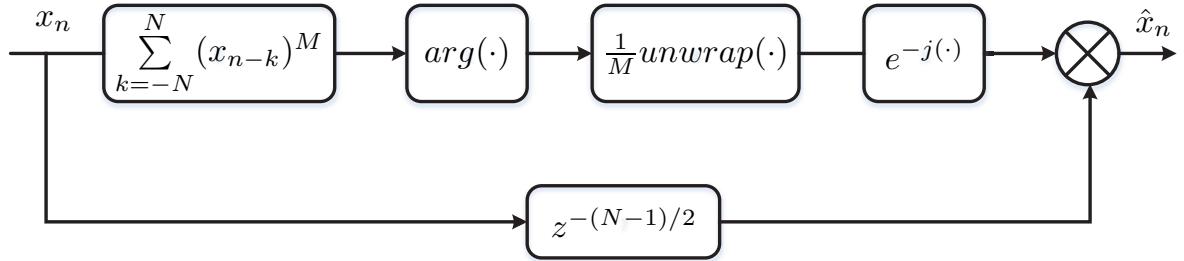


Figure 6.122: Block diagram of Viterbi-Viterbi algorithm for carrier phase recovery.

phase is removed by employing the  $n$ -th power operation on the received symbols. The algorithm implementation diagram is shown in Figure 6.122, starting with  $M$ -th power operation on the received symbols. In order to minimize the impact of additive noise in the estimation process, a sum of  $2N + 1$  symbols is considered, which is then divided by  $M$ . The resulting estimated phase noise is then submitted to a phase unwrap function in order to avoid the occurrence of cycle slip. The final phase noise estimator is then used to compensate for the phase noise of the original symbol in the middle of the symbols block.

### Blind Phase Search Algorithm

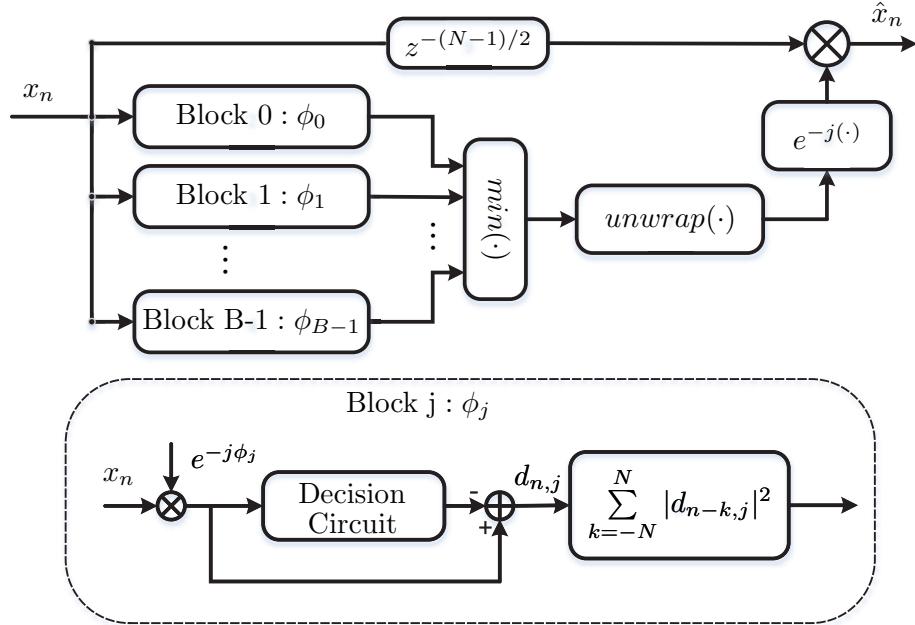


Figure 6.123: Block diagram of blind phase search algorithm for carrier phase recovery.

An alternative to the VV phase noise estimator is the so-called BPS algorithm, in which the operation principle is shown in the Figure 6.123. Firstly, a block of  $2N + 1$  consecutive

received symbols is rotated by a number of  $B$  uniformly distributed test phases defined as,

$$\phi_b = \frac{b}{B} \frac{\pi}{2}, b \in \{0, 1, \dots, B - 1\}. \quad (6.114)$$

Then, the rotated blocks symbols are fed into decision circuit, where the square distance to the closest constellation points in the original constellation is calculated for each block. Each resulting square distances block is summed up to minimize the noise distortion. After average filtering, the test phase providing the minimum sum of distances is considered to be the phase noise estimator for the symbol in the middle of the block. The estimated phase noise is then unwrapped to reduce cycle slip occurrence, which is then used employed for the compensation for the phase noise of the original symbols.

### 6.6.2 Simulation Analysis

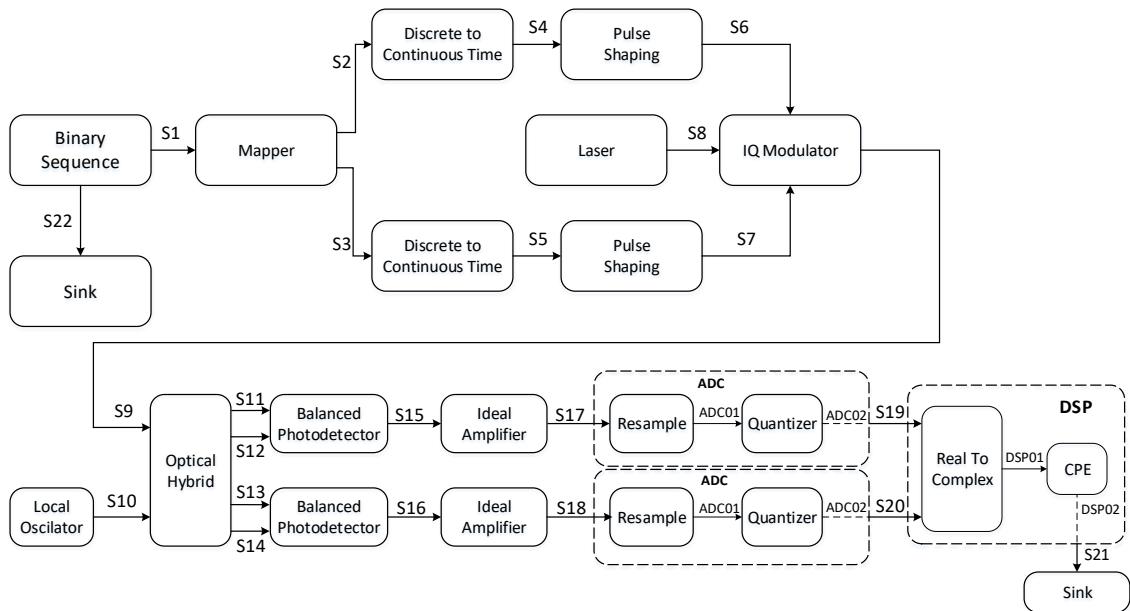


Figure 6.124: Simulation setup for the QPSK transmission system for the simulation of laser phase noise.

Figure 6.124 depicts the simulation setup for a QPSK transmission system, including the simulation of carrier phase noise. As can be observed in the figure 6.124, we have employed an IQ modulation which modulates the signal after pulse shaping,  $S_6$  and  $S_7$ , on the laser carrier,  $S_8$ . The output signal,  $S_9$ , is then fed to coherent receiver front-end, comprising optical hybrid, photodiode and ideal amplifier. The output signal to the coherent receiver front-end,  $S_{17}$  and  $S_{18}$ , are fed to the ADC super blocks, which comprises resample block followed by quantizer block. The two real output signals of ADC blocks,  $S_{19}$  and  $S_{20}$ , are fed into DSP super block, where is performed the compensation of transmission impairments and recover of transmitted data. The DSP super block is composed in this case

by a real to complex block, which combine the two input real signal into a complex signal and then fed to CPE block, where the laser phase noise compensation is performed. Then, the output,  $S21$ , is fed to the Sink block.

### 6.6.3 Simulation Results

The time-domain output signal of IQ modulator  $S9$  is presented in figure 6.125 (a) and its corresponding constellation diagram is shown in figure 6.125 (b). From the figure 6.125 (b) it can be clearly observed the effect of laser phase noise, where the signal  $S9$  presents a rotated constellation points. In general, the amount of the rotation is directly proportional to by the laser linewidth and the signal symbol rate.

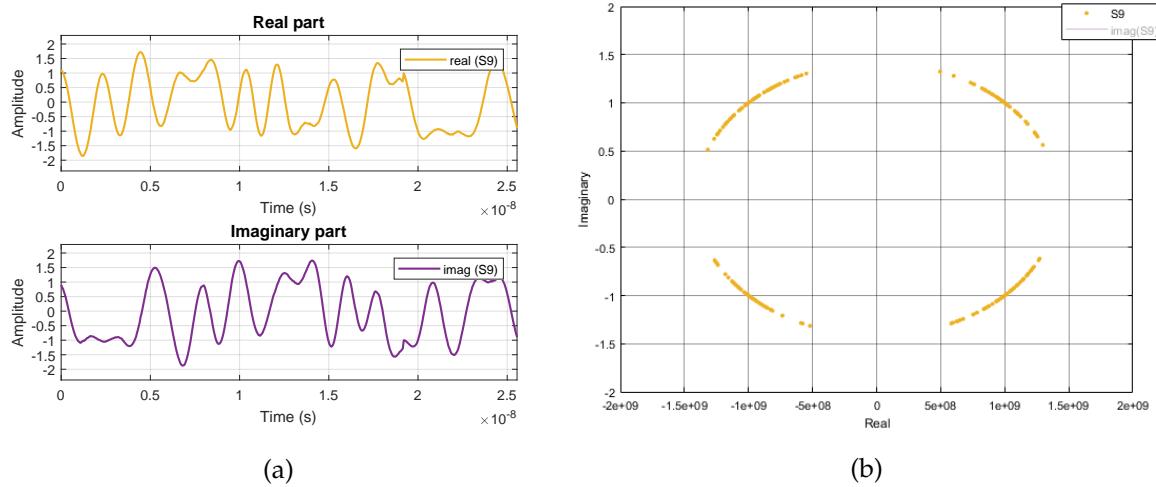


Figure 6.125: Output signal of IQ modulator,  $S9$ , including the laser phase noise. (a) Time-domain; (b) Constellations diagram.

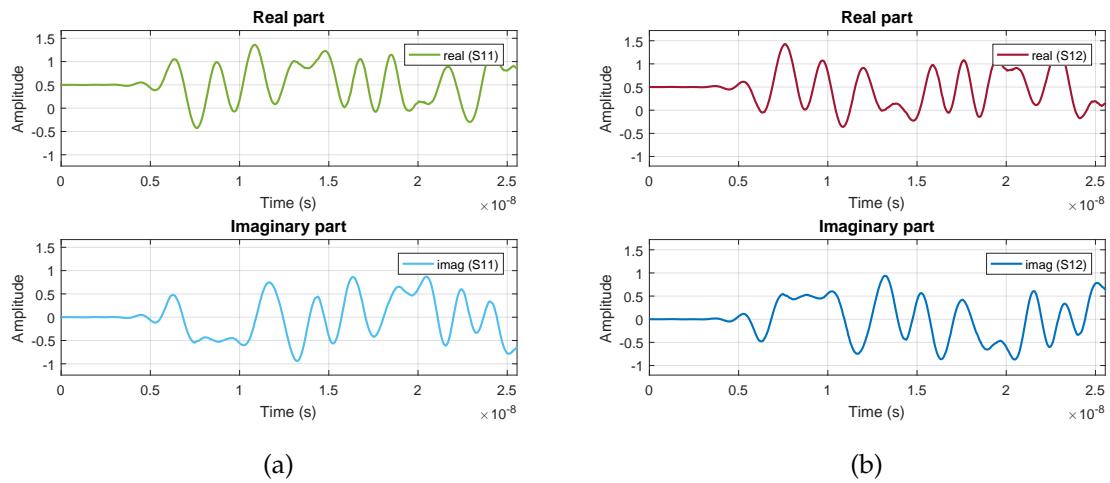


Figure 6.126: The in-phase output of optical hybrid block. (a)  $S11$ ; (b)  $S11$ .

The in-phase and quadrature of signal are obtained by using the optical hybrid, which is illustrated in the figure 6.127 and figure 6.128, respectively.

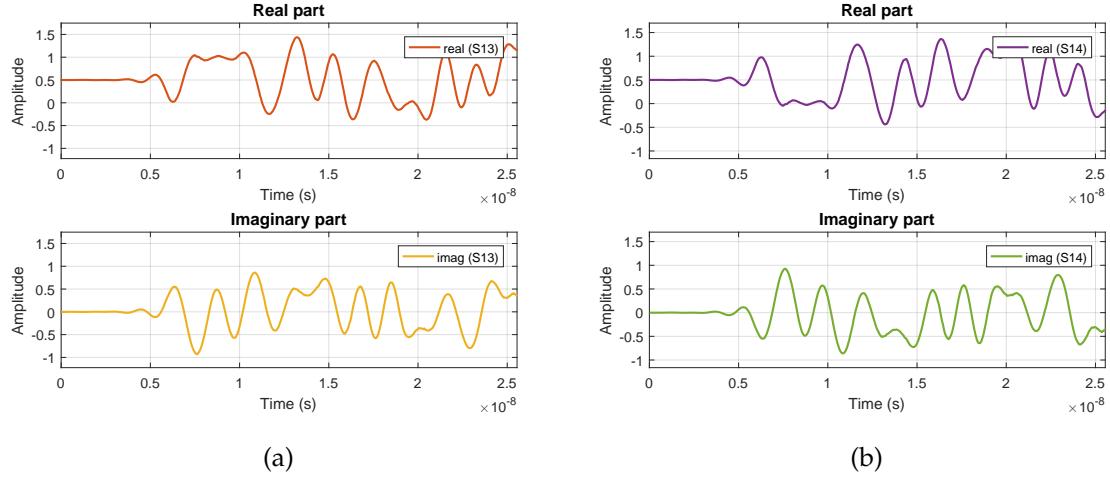


Figure 6.127: The quadrature output of optical hybrid block. (a)  $S13$ ; (b)  $S14$ .

The signal after balanced photodiodes is presented in the figure 6.128.

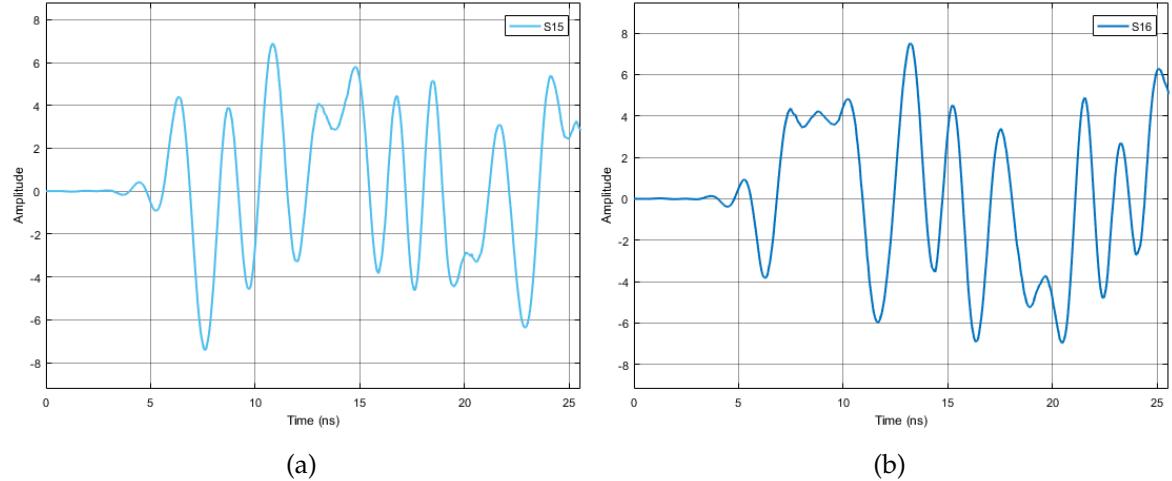


Figure 6.128: The output signal of balanced photodiodes block. (a) In-phase component,  $S15$ ; (b) Quadrature component,  $S16$ .

The signal at the output of the amplifier is shown in the figure 6.129.

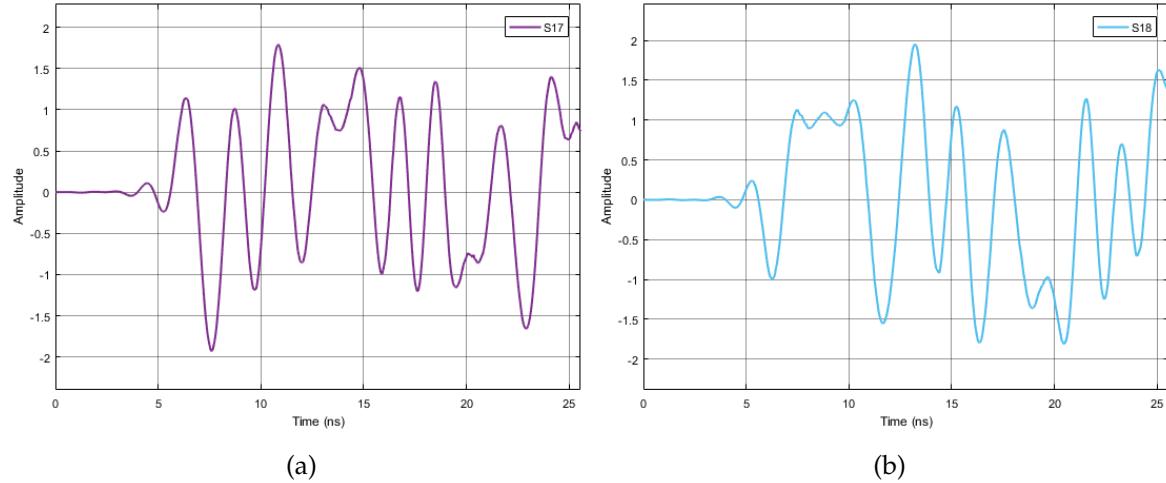


Figure 6.129: The output signal of the ideal amplifier block. (a) In-phase component,  $S17$ ; (b) Quadrature component,  $S18$ .

The signal after the super block ADC is shown in the figure 6.130

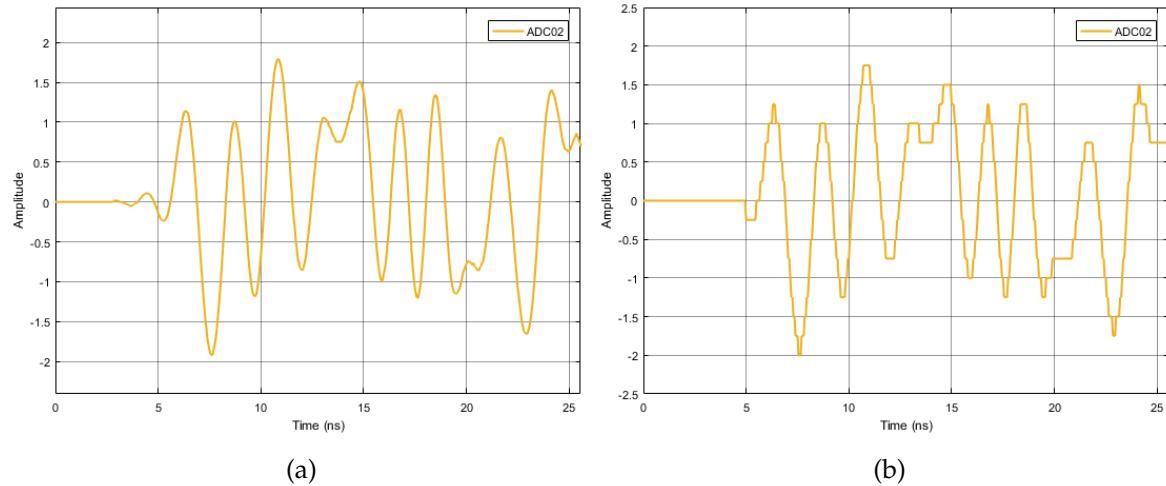


Figure 6.130: The output signal of the ADC super block, which includes the resample and quantizer blocks. (a) 8-bits quantization signal; (b) 4-bits quantization signal.

Figure 6.131 and Figure 6.132 show the signal after phase noise compensation using VV and BPS algorithms, respectively. As we can note in the figure 6.131 (b) and figure 6.132 (b), the correct constellation diagram points is recovered, which indicates that the laser phase noise is effectively compensated.

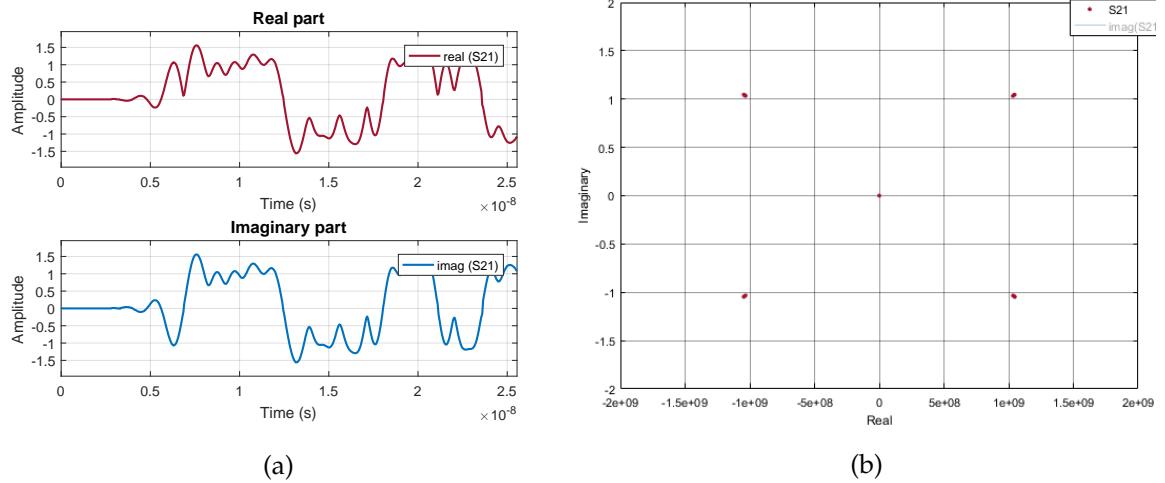


Figure 6.131: The signal after DSP block including VV algorithm for laser phase noise compensation. (a) Time-domain signal; (b) Constellation diagram.

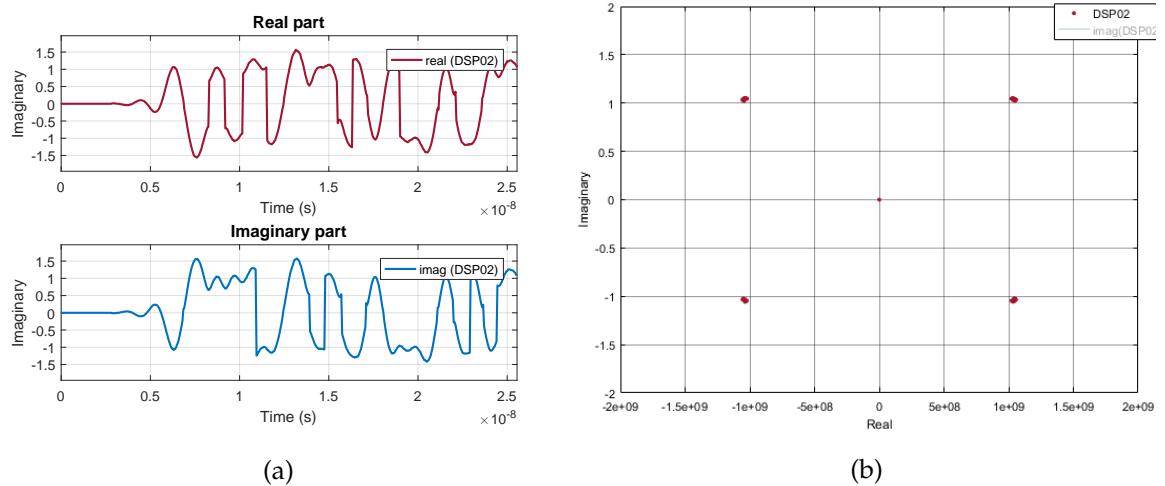


Figure 6.132: The signal after DSP block including BPS algorithm, using 64 test phases, for laser phase noise compensation. (a) Time-domain signal; (b) Constellation diagram.

## References

- [1] E. Ip and J. M. Kahn. "Feedforward Carrier Recovery for Coherent Optical Communications". In: *Journal of Lightwave Technology* 25.9 (Sept. 2007), pp. 2675–2692. ISSN: 0733-8724. DOI: [10.1109/JLT.2007.902118](https://doi.org/10.1109/JLT.2007.902118).
- [2] A. Viterbi. "Nonlinear estimation of PSK-modulated carrier phase with application to burst digital transmission". In: *IEEE Transactions on Information Theory* 29.4 (July 1983), pp. 543–551. ISSN: 0018-9448. DOI: [10.1109/TIT.1983.1056713](https://doi.org/10.1109/TIT.1983.1056713).
- [3] T. Pfau, S. Hoffmann, and R. Noe. "Hardware-Efficient Coherent Digital Receiver Concept With Feedforward Carrier Recovery for  $M$ -QAM Constellations". In: *Journal of Lightwave Technology* 27.8 (Apr. 2009), pp. 989–999. ISSN: 0733-8724. DOI: [10.1109/JLT.2008.2010511](https://doi.org/10.1109/JLT.2008.2010511).

## 6.7 Quantum Random Number Generator

<b>Students Name</b>	: Mariana Ramos (12/01/2018 - 11/04/2018)
<b>Goal</b>	: Simulate and implement an experimental setup of a Quantum Random Number Generator.
<b>Directory</b>	: sdf/quantum_random_number_generator.

True random numbers are indispensable in the field of cryptography [1]. There are two approaches for random number generation: the pseudorandom generation which are based on an algorithm implemented on a computer, and the physical random generators which consist in measuring some physical observable with random behaviour. Since classical physics description is deterministic, all classical processes are in principle predictable. Therefore, a true random number generator must be based on a quantum process [2].

In this chapter, it is presented the theoretical, the simulation and the experimental analysis of a quantum random generator based on the use of single photons linearly polarized at  $45^\circ$ .

### 6.7.1 Theoretical Analysis

One of the optical processes available as a source of randomness is the splitting of a polarized single photon beam. The principle of operation of the random generator is shown in figure 6.133. Each individual photon coming from the source is linearly polarized at  $45^\circ$  and has equal probability of be found in the horizontal (H) or in the vertical (V) output of the PBS. Quantum theory estimates for both cases that the individual choices are truly random, independent one from each other, and with a probability of  $1/2$ .

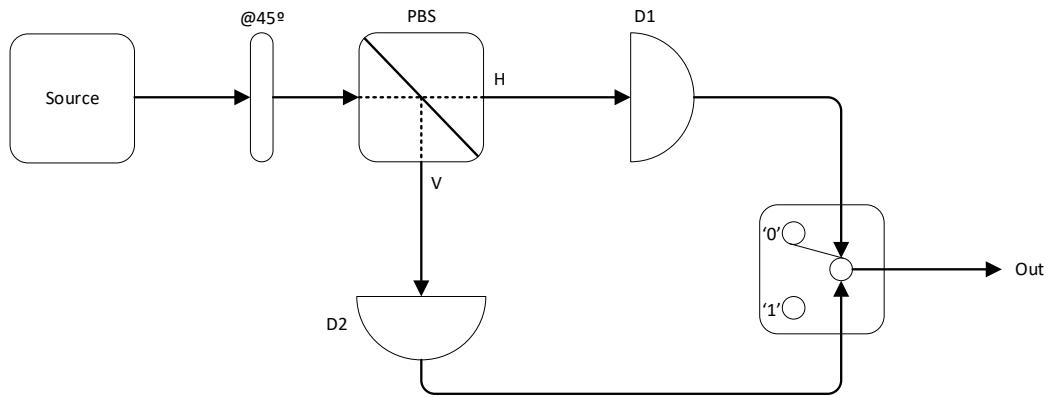


Figure 6.133: Source of randomness with a polarization beam splitter PBS where the incoming light is linearly polarized at  $45^\circ$  with respect to the PBS. Figure adapted from [2].

From a classical approach, the information is stored as binary bits that can take the logical value '0' or '1'. From a quantum approach, the information can be stored in quantum

bits or qubits for short. As a consequence of the superposition principle of quantum mechanics, qubits can not only represent the pure '0' or '1' states, but they can also represent a superposition of both. This way, qubits are governed by a quantum wave function  $\psi$ . Lets use the Dirac notation to represent the general state of the qubit:

$$|\psi\rangle = C_0|0\rangle + C_1|1\rangle, \quad (6.115)$$

and the normalization condition of  $|\psi\rangle$  requires that  $|C_0|^2 + |C_1|^2 = 1$ . This way, the relative proportion of each of the binary states on a qubit is governed by the amplitude coefficients  $C_0$  and  $C_1$ . In the present example, we consider a linear polarization in which the two possible states are orthogonal, such that:  $\langle 0|1\rangle = 0$ . We define the  $|0\rangle$  and  $|1\rangle$  states to correspond to the horizontal and vertical polarization states, respectively:

$$|\psi\rangle = C_0|0\rangle + C_1|1\rangle \quad (6.116)$$

$$= C_0|0^\circ\rangle + C_1|90^\circ\rangle. \quad (6.117)$$

Amplitude coefficients  $C_0$  and  $C_1$  store the quantum information. Therefore, if one makes a measurement, the result will be '0' with probability  $|C_0|^2$  or '1' with probability  $|C_1|^2$ . Moreover, the state of a single photon can be also described by a wave function as a column vector:

$$|\psi\rangle = \begin{pmatrix} C_0 \\ C_1 \end{pmatrix}, \quad (6.118)$$

which will be used in simulation analysis.

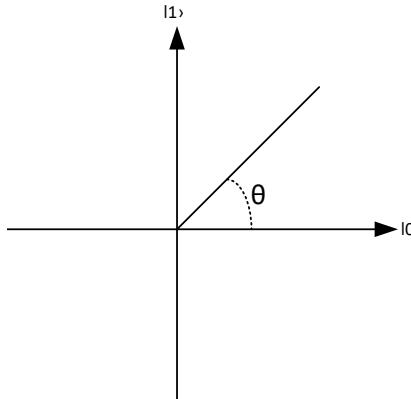


Figure 6.134: Representation of polarization states of a qubit in a bi-dimensional space.

As one can see in figure 6.134 the amplitude coefficients can be written as a function of  $\theta$ :

$$C_0 = \cos(\theta) \quad (6.119)$$

$$C_1 = \sin(\theta). \quad (6.120)$$

According with the setup presented in figure 6.133 and considering the polarization angle  $\theta = 45^\circ$ , the single photon has the probability of reach **D1** and outputs a "0" is equals to  $|\cos(\theta)|^2$  and the probability of reach **D2** and outputs a "1" is equals to  $|\sin(\theta)|^2$ , which in the case  $\theta = 45^\circ$  both have the same value equals to 0.5.

### 6.7.2 Simulation Analysis

The simulation diagram of the setup described in the previous section is presented in figure 6.135. The linear polarizer has an input control signal (S1) which allows to change the rotation angle. Nevertheless, the only purpose is to generate a time and amplitude continuous real signal with the value of the rotation angle in degrees. In addition, the photons are generated by single photon source block at a rate defined by the clock rate. At the end of the simulation there is a circuit decision block which will outputs a binary signal with value "0" if the detector at the end of the horizontal path clicks or "1" if the detector at the end of the vertical path clicks.

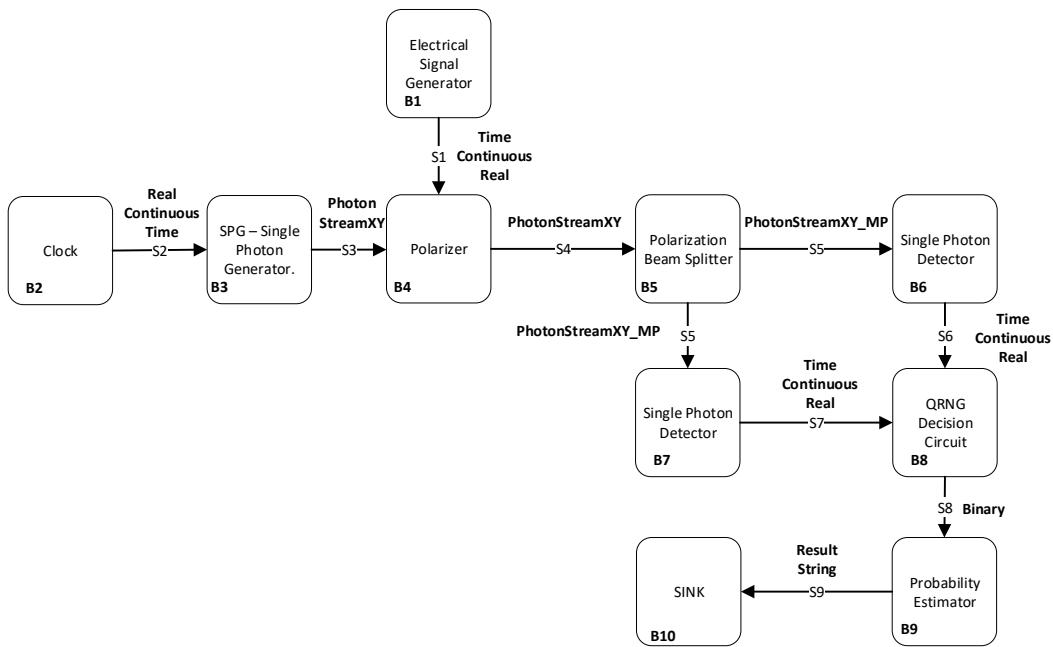


Figure 6.135: Block diagram of the simulation of a Quantum Random Generator.

In table 6.20 are presented the input parameters of the system.

Table 6.20: System Input Parameters

Parameter	Default Value
RateOfPhotons	1e6
NumberOfSamplesPerSymbol	16
PolarizerAngle	45.0

In table 6.21 are presented the system signals to implement the simulation presented in figure 6.135.

Table 6.21: System Signals

Signal name	Signal type
S1	TimeContinuousAmplitudeContinuousReal
S2	TimeContinuousAmplitudeContinuousReal
S3	PhotonStreamXY
S4	PhotonStreamXY
S5	PhotonStreamXYMP
S6	TimeContinuousAmplitudeContinuousReal
S7	TimeContinuousAmplitudeContinuousReal
S8	Binary
S9	Binary

Table 6.22 presents the header files used to implement the simulation as well as the specific parameters that should be set in each block. Finally, table 6.23 presents the source files.

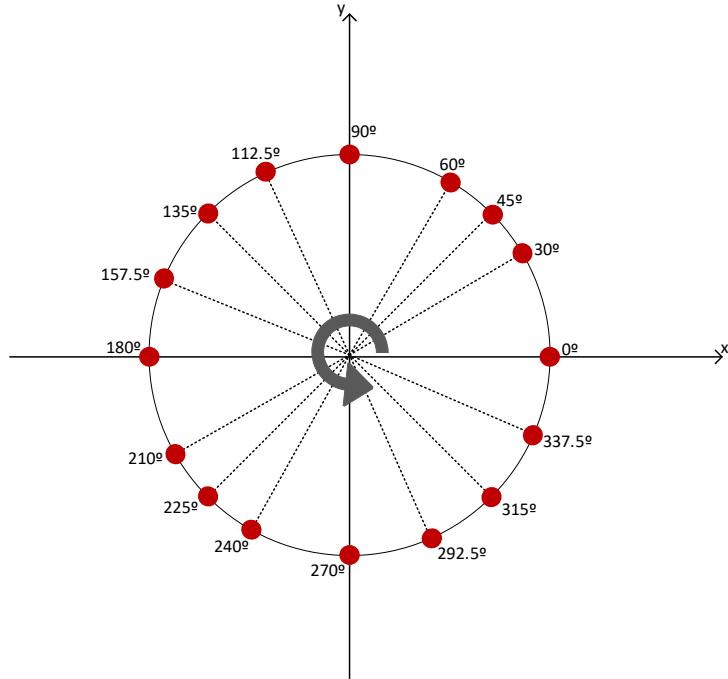
Table 6.22: Header Files

File name	Description	Status
netxpto_20180118.h		✓
electrical_signal_generator_20180124.h	setFunction(), setGain()	✓
clock_20171219.h	ClockPeriod(1 / RateOfPhotons)	✓
polarization_beam_splitter_20180109.h		✓
polarizer_20180113.h		✓
single_photon_detector_20180111.h	setPath(0), setPath(1)	✓
single_photon_source_20171218.h		✓
probability_estimator_20180124.h		✓
sink.h		✓
qrng_decision_circuit.h		✓

Table 6.23: Source Files

File name	Description	Status
netxpto_20180118.cpp		✓
electrical_signal_generator_20180124.cpp		✓
clock_20171219.cpp		✓
polarization_beam_splitter_20180109.cpp		✓
polarizer_20180113.cpp		✓
single_photon_detector_20180111.cpp		✓
single_photon_source_20171218.cpp		✓
probability_estimator_20180124.cpp		✓
sink.cpp		✓
qrng_decision_circuit.cpp		✓
qrng_sdf.cpp		✓

Lets assume, for an angle of  $45^\circ$ , a number of samples  $N = 1 \times 10^6$  and the expected probability of reach each detector of  $\hat{p} = 0.5$ . We have an error margin of  $E = 1.288 \times 10^{-3}$ , which is acceptable. This way, the simulation will be performed for  $N = 1 \times 10^6$  samples for different angles of polarization shown in figure 6.136 with different error margin's values since the expected probability changes depending on the polarization angle.

Figure 6.136: Angles used to perform the qrng simulation for  $N = 1 \times 10^6$  samples.

For a quantum random number generator with equal probability of obtain a "0" or "1" the polarizer must be set at  $45^\circ$ . This way, we have 50% possibilities to obtain a "0" and 50% of possibilities to obtain a "1" . This theoretical value meets the value obtained from the simulation when it is performed for the number of samples mentioned above.

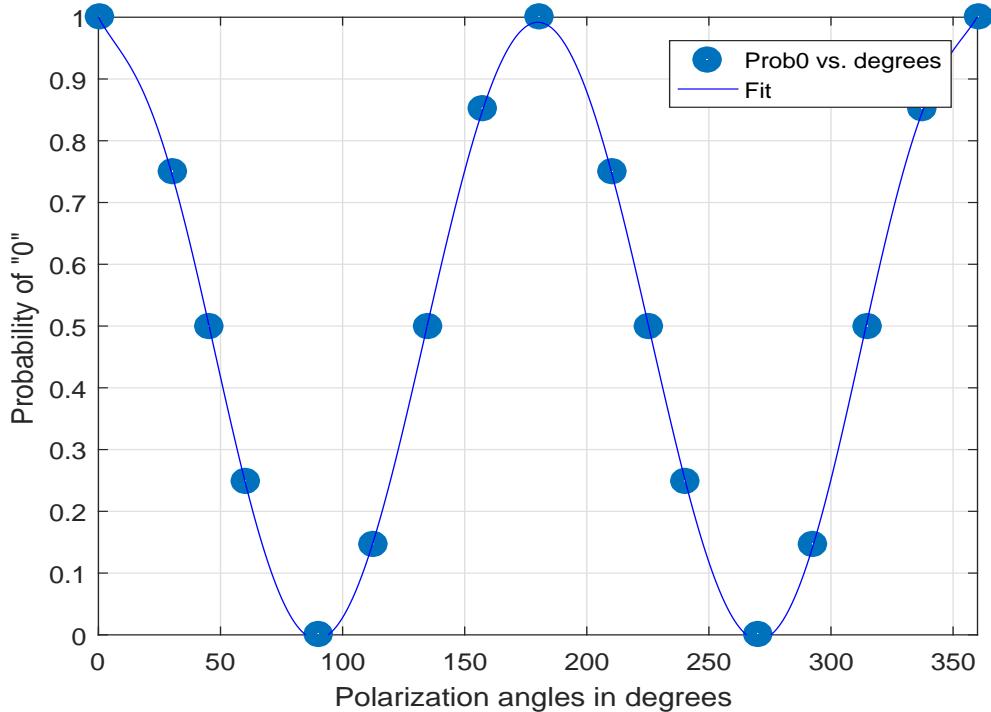


Figure 6.137: Probability of outputs a number "0" depending on the polarization angle.

Figure 6.137 shows the probability of a single photons reaches the detector placed on Horizontal axis depending on the polarization angle of the photon, and this way the output number is "0". The following table shows the goodness of the fit:

SSE:	0.0004785
R-square:	0.9998
Adjusted R-square:	0.9995
RMSE:	0.007734

On the other hand, figure 6.138 shows the probability of a single photon reaches the detector placed on Vertical component of the polarization beam splitter, and this way the output number is "1". As we can see in the figures the two detectors have complementary probabilities, i.e the summation of both values must be equals to 1. One can see that "Probability of 1" behaves almost like a sine function and "Probability of 0" behaves almost like a cosine function with a variable angle.

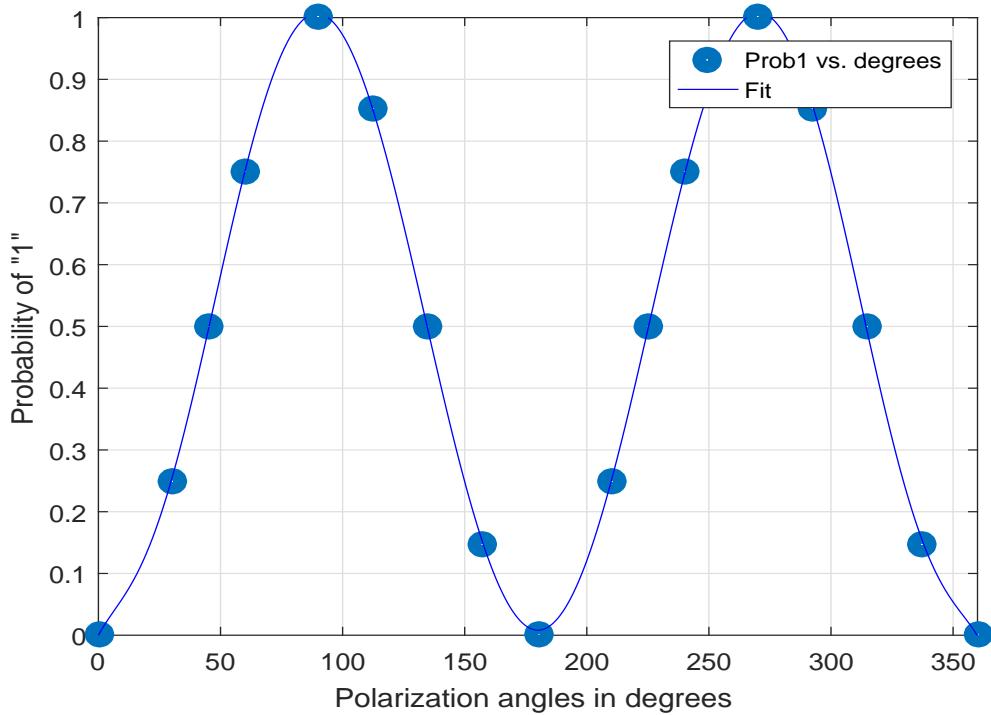


Figure 6.138: Probability of outputs a number "1" depending on the polarization angle.

The goodness of the fit presented in figure 6.138 is shown in the following table:

SSE:	0.0004785
R-square:	0.9998
Adjusted R-square:	0.9995
RMSE:	0.007734

The goodness of the fit is evaluated based on four parameters:

1. The sum of squares due to error (SSE), which measures the total deviation between the fit values and the values that the simulation outputs. This value is calculated from the expression

$$SSE = \sum_{i=1}^n w_i (y_i - \hat{y}_i)^2.$$

A value of SSE closer to 0 means that the model has a small random error component.

2. The R-square measures how good the fit in explaining the data.

$$R\text{-square} = 1 - \frac{SSE}{SST},$$

where,

$$SST = \sum_{i=1}^n w_i (y_i - \bar{y}_i)^2.$$

R-square can take a value between 0 and 1. If the value is closer to 1, it means that the fit better explains the total variation in the data around the average.

3. Degrees of freedom adjusted R-square uses the R-square and adjusts it based on the number of degrees of freedom.

$$\text{adjusted R-square} = 1 - \frac{\text{SSE}(n - 1)}{\text{SST}(v)},$$

where,

$$v = n - m,$$

where  $n$  is the number of values in test and  $m$  is the number of fitted coefficients estimated from the values in test. A value of adjusted R-square close to 1 is a indicative factor of a good fit.

4. The root mean square error (RMSE) is also a fit standard error and it can be calculated from:

$$\text{RMSE} = \sqrt{\text{MSE}},$$

where,

$$\text{MSE} = \frac{\text{SSE}}{v}.$$

### 6.7.3 Experimental Analysis

In order to have a real experimental quantum random number generator, a setup shown in figure 6.139 was built in the lab. To simulate a single photon source we have a CW-Pump laser with 1550 nm wavelength followed by an interferometer Mach-Zenhder in order to have a pulsed beam. The interferometer has an input signal given by a Pulse Pattern Generator. This device also gives a clock signal for the Single Photon Detector (APD-Avalanche Photodiode) which sets the time during which the window of the detector is open. After the MZM there is a Variable Optical Attenuator (VOA) which reduces the amplitude of each pulse until the probability of one photon per pulse is achieved. Next, there is a polarizer controller followed by a Linear Polarized, which is set at 45°, then a Polarization Beam Splitter (PBS) and finally, one detector at the end of each output of the PBS. The output signals from the detector will be received by a Processing Unit. Regarding to acquired the output of the detectors, there is an oscilloscope capable of record  $1 \times 10^6$  samples.

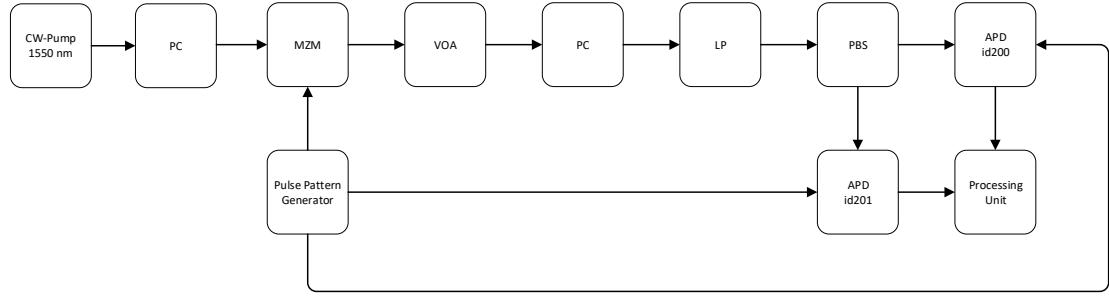


Figure 6.139: Experimental setup to implement a quantum random number generator.

### IDQuantique detector

The detector used in the laboratory is the Thorlabs PDB 450C. This detector consists of two well-matched photodiodes and a transimpedance amplifier that generates an output voltage (RF OUTPUT) proportional to the difference between the photocurrents of the photodiodes. Additionally, the unit has two monitor outputs (MONITOR+ and MONITOR-) to observe the optical input power level on each photodiode separately.

Since we do not have a single photon source, we must use the Poisson Statistics in order to calculate the best value for a mean of photons per pulse. A weak laser pulse follows a Poissonian Statistics[3]:

$$S_n = e^{-\mu} \frac{\mu^n}{n!}, \quad (6.121)$$

where  $\mu$  is the average photon number. In addition, the probability of an optical pulse carries one photon at least is:

$$P = 1 - S_0 = 1 - e^{-\mu}. \quad (6.122)$$

On the other hand, the probability of a detector clicks is:

$$P_{click} = P_{det} + P_{dc} + P_{det}P_{dc}, \quad (6.123)$$

where  $P_{det}$  is the probability of the detector clicks due a photon which cross its window and  $P_{dc}$  is the probability of dark counts. Considering the detector efficiency  $\eta_D$ , the probability of the detector clicks due to a photon is:

$$P_{det} = 1 - e^{-\eta_D \mu}. \quad (6.124)$$

The probability of dark counts is calculated as a ratio between the frequency counts and the trigger frequency when no laser is connected to the detector. Nevertheless, the detector click frequency is

$$f_{click} = f_{trigger} P_{click} \longrightarrow P_{click} = \frac{f_{click}}{f_{trigger}}. \quad (6.125)$$

This way the mean average photon number can be calculate using the following equation:

$$\mu = -\frac{1}{\eta_D} \ln \left[ 1 - \frac{1}{1 - P_{dc}} \left( \frac{f_{click}}{f_{trigger}} - P_{dc} \right) \right] \quad (6.126)$$

#### **6.7.4 Open Issues**

- Experimental Implementation.
- Random number validation/standardization.

## References

- [1] GE Katsoprinakis et al. "Quantum random number generator based on spin noise". In: *Physical Review A* 77.5 (2008), p. 054101.
- [2] Thomas Jennewein et al. "A fast and compact quantum random number generator". In: *Review of Scientific Instruments* 71.4 (2000), pp. 1675–1680. DOI: [10.1063/1.1150518](https://doi.org/10.1063/1.1150518).
- [3] Mark Fox. *Quantum Optics, an Introduction*. Oxford, University Press, 2006.

## 6.8 BB84 with Discrete Variables

<b>Students Name</b>	: Mariana Ramos (7/11/2017 - 9/4/2018)
	Kevin Filipe (7/11/2017 - 10/11/2017)
<b>Starting Date</b>	: November 7, 2017
<b>Goal</b>	: BB84 implementation with discrete variables.

BB84 is a key distribution protocol which involves three parties, Alice, Bob and Eve. Alice and Bob exchange information between each other by using a quantum channel and a classical channel. The main goal is continuously build keys only known by Alice and Bob, and guarantee that eavesdropper, Eve, does not gain any information about the keys.

### 6.8.1 Protocol Analysis

<b>Students Name</b>	: Kevin Filipe (7/11/2017 - 10/11/2017)
<b>Goal</b>	: BB84 - Protocol Description

BB84 protocol was created by Charles Bennett and Gilles Brassard in 1984 [1]. It involves two parties, Alice and Bob, sharing keys through a quantum channel in which could be accessed by a eavesdropper, Eve. A basic model is depicted in figure 6.140.

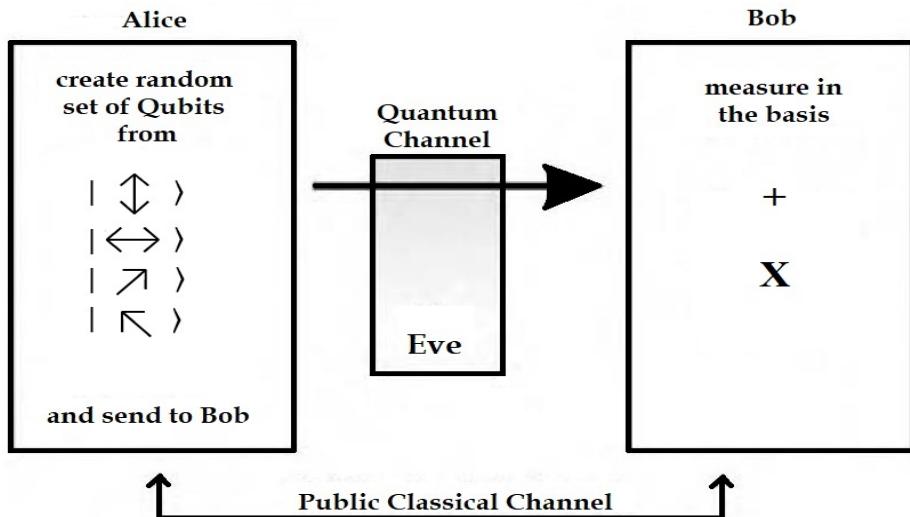


Figure 6.140: Basic QKD Model. Alice and Bob are connected by 2 communication channels, a public quantum channel and a authenticated classical channel, with an eavesdropper, Eve (figure adapted from [2]).

We are going to analyse the BB84 protocol with bit encoding into photon state polarization. Two non-orthogonal basis are used to encode the information, the rectilinear and diagonal basis, + and x respectively. The following table shows this bit encoding.

Bit	Rectilinear Basis, +	Diagonal Basis, $\times$
0	0	-45
1	90	45

The protocol requires the following parameter and it is implemented with the following steps:

Table 6.24: Initial Parameters.

Parameter	Description
$M \times N$	Scrambling Matrix M by N
k	Number of revealed bits for BER calculation
$\alpha$	Confidence level
A	B

1. Alice generates two random bit strings. The random string,  $R_{A1}$ , corresponds to the data to be encoded into photon state polarization.  $R_{A2}$  is a random string in which 0 and 1 corresponds to the rectilinear, +, and diagonal,  $\times$ , respectively.

$$R_{A1} = \{0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1\}$$

$$\begin{aligned} R_{A2} &= \{0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0\} \\ &= \{+, +, \times, +, \times, \times, +, \times, \times, +, \times, +, +, +, \times, +, \times, +, +\} \end{aligned}$$

2. Alice transmits a train of photons,  $S_{AB}$ , obtained by encoding the bits,  $R_{A1}$  with the respective photon polarization state  $R_{A2}$ .

$$S_{AB} = \{\rightarrow, \uparrow, \searrow, \rightarrow, \searrow, \nearrow, \nearrow, \uparrow, \searrow, \nearrow, \searrow, \uparrow, \searrow, \rightarrow, \rightarrow, \uparrow, \nearrow, \rightarrow, \nearrow, \uparrow\}.$$

3. Bob generates a random string,  $R_B$ , to receive the photon trains with the correspondent basis.

$$\begin{aligned} R_B &= \{0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0\} \\ &= \{+, \times, \times, \times, +, \times, +, +, \times, \times, +, +, \times, +, +, \times, \times, +, +\} \end{aligned}$$

4. Bob performs the incoming photon states measurement,  $M_B$ , with its generated random basis,  $R_B$ . If the two photon detectors don't click, means the bit was lost during transference due to attenuation. If both photon detectors click, a false positive was detected. In the measurements,  $M_B$ , the no-click in both detectors is represented by a -1 and the false positives to -2. The measurements done in rectilinear or diagonal basis are represented by 0 or 1, respectively. This is represented 6.141

$$M_B = \{0, 1, 1, 1, -1, 1, 0, 0, -2, 1, 0, 0, -2, 1, 0, 0, 1, -1, 0, 0\}$$

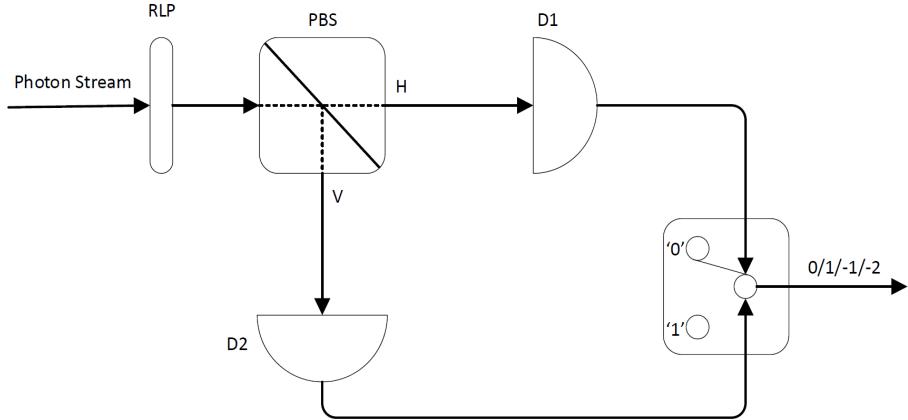


Figure 6.141: Single-Photon Detection block with false-positives, -2, and attenuation, -1, detection depending on D1 and D2 output.

5. After the measurement, Bob sends to Alice, using the classical channel, the used basis values,  $R_B$  with the attenuation, -1, and false positives,-2.
6. Alice performs a modified negated XOR, generating a sequence that detects when the same basis she used  $B_{AB}$ .

$R_{A2}$	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	0
$R_B$	0	1	1	1	-1	1	0	0	-2	1	0	0	-2	1	0	0	1	-1	0
$B_{AB}$	1	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	1	0	0

7. Alice sends the  $B_{AB}$  sequence to Bob, in which he can correlate with,  $M_B$ , and deduce the key  $K_{AB}$ .

$$K_{AB} = \{0, 1, 0, 1, 0, 1, 0, 1, 0, 1\}.$$

8. Alice then by having knowledge of  $R_{A2}$  and  $B_{AB}$  performs a scrambling algorithm over the deduced key. It is generated a matrix  $M \times N$ , according to the input parameter. Assuming a scrambling matrix of  $3 \times 4$ , 6.25. And being the scramble key represented as  $KS_{AB}$

Table 6.25: Scrambling matrix

0	1	0	1
0	1	0	1
0	1	-	-

$$KS_B = \{0, 0, 0, 1, 1, 1, 0, 0, 1, 1\}$$

9. Bob uses the same algorithm as Alice and scrambles his key.
10. Bob then reveals a fixed number of his key to Alice. This number is also an input parameter value,  $k$ . With this the Quantum Bit Error Rate (QBER).

To determine the QBER, it is necessary to know the confidence interval parameter,  $\alpha$  and the QBER limit, in which states the maximum allowed QBER by the user. Then to verify if the channel is reliable or not, the flowchart presented in figure 6.142.

1. Bob will reveals  $k$  bits sequence from the scrambled key,  $SK_{AB}$  to Alice.
2. Alice then returns to Bob the estimated QBER value,  $mQBER$ , with a confidence interval,  $[qLB, qUB]$  using the using the equations in the Bit Error Rate section, but applied to this protocol
3. To check if the channel is compromised or not it is necessary to check if the QBER limit is higher than the QBER upper bound. If QBER limit is between the QBER lower and upper bound it is necessary to reveal more  $k$  bits from the key. Otherwise the channel is compromised and the key determination process needs to restart.

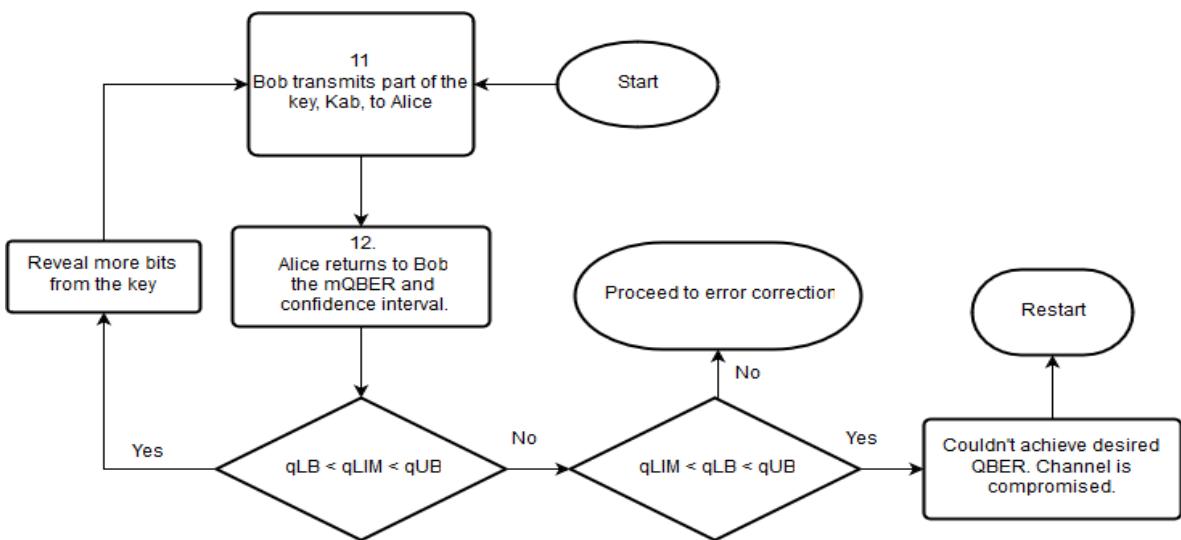


Figure 6.142: Flowchart to determine if the channel is reliable or not.

### 6.8.2 Simulation Analysis

<b>Students Name</b>	:	Mariana Ramos (7/11/2017 - 9/4/2018)
<b>Goal</b>	:	Perform a simulation of BB84 communication protocol.

In this sub section the simulation setup implementation will be described in order to implement the BB84 protocol. In figure 6.143 a top level diagram is presented. Then it will be presented the block diagram of the transmitter block (Alice) in figure 6.144 and the receiver block (Bob) in figure 6.145. In a first approach, we do not consider the existence of eavesdropper.

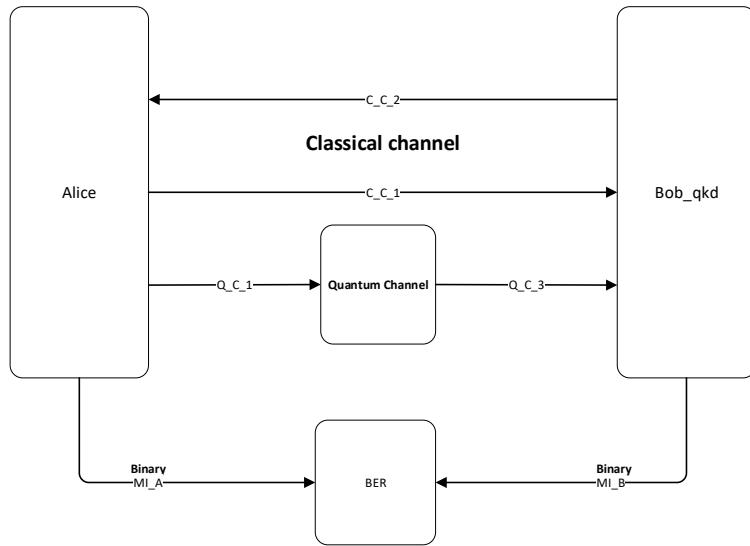


Figure 6.143: Simulation diagram at Alice's side

Figure 6.143 presents the top level diagram of our simulation. The setup contains two parties Alice and Bob, where the communication between them is done throughout two authenticated classical channels and one public quantum channel. In a first approach we will perform the simulation without eavesdropper presence. Furthermore, for bit error rate calculation between Alice and Bob.

In figure 6.144 one can observe a block diagram of the simulation at Alice's side. As it is shown in the figure, Alice must have one block for random number generation which is responsible for basis generation to polarize the photons, and for key random generation in order to have a random state to encode each photon. Furthermore, she has a Processor block for all logical operations: array analysis, random number generation requests, and others. This block also receives the information from Bob after it has passed through a fork's block. In addition, it is responsible for set the initial length  $l$  of the first array of photons which will send to Bob. This block also must be responsible for send classical information

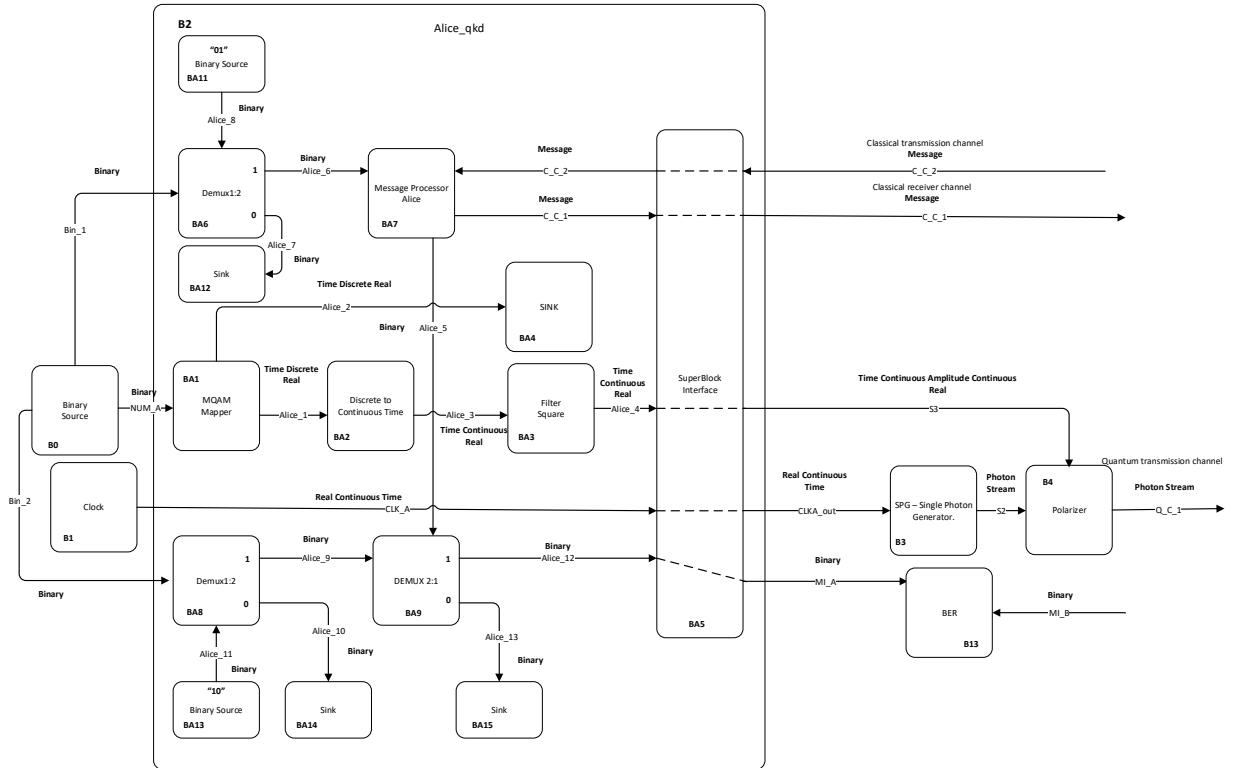


Figure 6.144: Simulation diagram at Alice's side

to Bob. Finally, Processor block will also send a real continuous time signal to single photon generator, in order to generate photons according to this signal, and finally this block also sends to the polarizer a real discrete signal in order to inform the polarizer which basis it should use. Therefore, she has two more blocks for quantum tasks: the single photon generator and the polarizer block which is responsible to encode the photons generated from the previous block and send them throughout a quantum channel from Alice to Bob.

Finally, Alice's processor has an output to Mutual Information top level block,  $M s_A$ .

In figure 6.144 one can observe a block diagram of the transmitter. As it is shown in the figure, the transmitter must have one block for random number generation (binary source) which is responsible for basis generation to polarize the photons, and for key random generation in order to have a random state to encode each photon. This block has three outputs which will be inputs for the super block Alice. Furthermore, Alice block is responsible for all logical operations: random single photons state values generation, receive and send messages to the receiver Bob by using the classical channels, binary output for mutual information calculations. Each block of the super block is described in Library chapter. Finally, Alice block will also send a real continuous time signal to single photon generator (clock sets the rate of photons generation), in order to generate photons polarized in the horizontal axis by default. Therefore, the transmitter has one more block, the polarizer

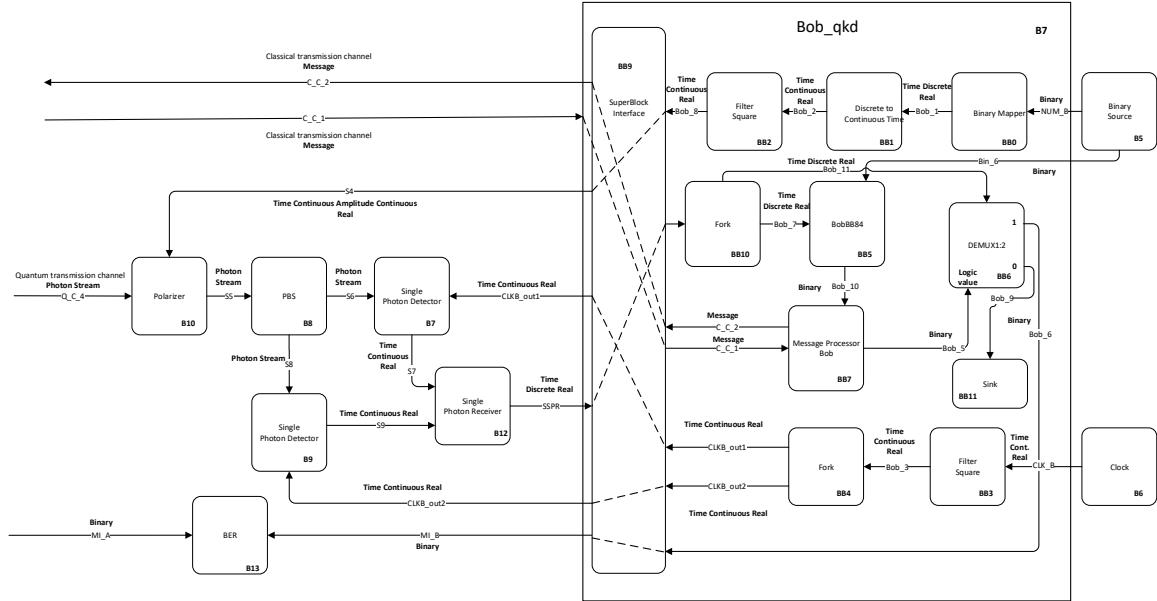


Figure 6.145: Simulation diagram at Bob's side

block, which is responsible to encode the photons generated from the previous block and send them throughout a quantum channel from Alice to Bob.

In figure 6.145 one can see a block diagram of the simulation for receiver (Bob). The receiver has one block for Random Number Generation which is responsible for randomly generate basis values which Bob will use to measure the photons sent by Alice throughout the quantum channel. Like transmitter, the receiver has the Bob block responsible for receive and send messages through the classical channel, receive single photons values detection from the single photon detectors, provides a clock signal to the detectors and send binary values for mutual information calculation. Furthermore, the receiver has two blocks for single photon detection (one for horizontal detection and other for vertical detection) which receives from Bob block a real continuous time signal which will set the detection window for the detector and outputs for Bob block the result value for detection. In addition, there is a polarizer which receives from Bob block a time continuous real signal which provides information about the rotation angle. If the basis chosen by Bob is the diagonal basis he sends "45°", otherwise sends "0°". The polarization beam splitter divides the input photon stream in horizontal component and vertical component.

Table 6.26: System Signals

Signal name	Signal type
NUM_A, NUM_B, Bin_1, Bin_2, Bin_6	Binary
MI_A, MI_B	Binary
CLK_A, CLK_B	TimeContinuousAmplitudeContinuous
CLK_A_out, CLKB_out1, CLKB_out2	TimeContinuousAmplitudeContinuous
S2, S5, S6, S8	PhotonStreamXY
S3, S7, S9	TimeContinuousAmplitudeDiscreteReal
S4	TimeContinuousAmplitudeContinuousReal
C_C_1, C_C_3	Messages
C_C_6, C_C_4	Messages
Q_C_1, Q_C_4	PhotonStreamXY

Table 6.29 presents the system signals as well as them type.

Table 6.27: System Input Parameters

Parameter	Default Value	Description
RateOfPhotons	1K	Number of photon per sample.
iqAmplitudeValues	{-45,0},{0,0},{45,0},{90,0}	Possible photon states.
NumberOfSamplesPerSymbom	16	Number of samples per symbol.
DetectorWindowTimeOpen	0.2	smaller than 1 ms
DetectorPulseDelay	0.7	in units of ms
DetectorProbabilityDarkCount	0.0	Probability of dark counts in single-photon detector.
RotationAngle	0.0	Polarization angle in XY axis to introduce in Deterministic SOP changes.
ElevationAngle	0.0	Polarization angle in Poincare sphere to introduce in Deterministic SOP changes.

Table 6.28: Header Files

File name	Description	Status
netxpto_20180118.h		✓
alice_qkd_20180409.h		✓
binary_source_20180118.h		✓
bob_qkd_20180409.h		✓
clock_20171219.h		✓
discrete_to_continuous_time_20180118.h		✓
m_qam_mapper_20180118.h		✓
polarization_beam_splitter_20180109.h		✓
polarization_rotator_20180113.h		✓
pulse_shaper_20180111.h		✓
single_photon_detector_20180206.h		✓
single_photon_receiver_20180303.h		✓
SOP_modulator_20180319.h		✓
coincidence_detector_20180206.h		✓
single_photon_source_20171218.h		✓
sink_20180118.h		✓
super_block_interface_20180118.h		✓
message_processor_alice_20180205.h		✓
demux_1_2_20180205.h		✓
binary_mapper_20180205.h		✓
bobBB84_20180221.h		✓
message_processor_bob_20180221.h		✓
sampler_20171119.h		✓
optical_attenuator_20180304.h		✓
fork_20180112.h		✓

Table 6.29: Source Files

File name	Description	Status
netxpto_20180118.cpp		✓
bb84_with_discrete_variables_sdf.cpp		✓
alice_qkd_20180409.cpp		✓
binary_source_20180118.cpp		✓
bob_qkd_20180409.cpp		✓
clock_20171219.cpp		✓
discrete_to_continuous_time_20180118.cpp		✓
m_qam_mapper_20180118.cpp		✓
polarization_beam_splitter_20180109.cpp		✓
polarization_rotator_20180113.cpp		✓
pulse_shaper_20180111.cpp		✓
single_photon_detector_20180206.cpp		✓
single_photon_receiver_20180303.cpp		✓
SOP_modulator_20180319.cpp		✓
coincidence_detector_20180206.cpp		✓
single_photon_source_20171218.cpp		✓
sink_20180118.cpp		✓
super_block_interface_20180118.cpp		✓
message_processor_alice_20180205.cpp		✓
demux_1_2_20180205.cpp		✓
binary_mapper_20180205.cpp		✓
bobBB84_20180221.cpp		✓
message_processor_bob_20180221.cpp		✓
sampler_20171119.cpp		✓
optical_attenuator_20180304.cpp		✓
fork_20180112.cpp		✓

### Simulation Results

Figure 6.146 represents the block diagram of the first simulation performed between Alice and Bob. This simulation intends to simulate the communication protocol between Alice and Bob until they do the Basis Reconciliation. At this time, it is not taken into account any attack from an eavesdropper. However, as one can learn from theoretical protocol analysis, the attenuation due the fiber losses, dark counts probabilities from single photon detectors and the SOP drift over the quantum channel are all taken into account.

Alice starts by sending a sequence of photons to Bob, and then he measures the photons according to random basis randomly generated by his binary source. After that, he follows the protocol described above until Alice sends to him a string of '0' and '1' where '0' means that both used different basis and '1' means that they used the same basis. Therefore, Alice

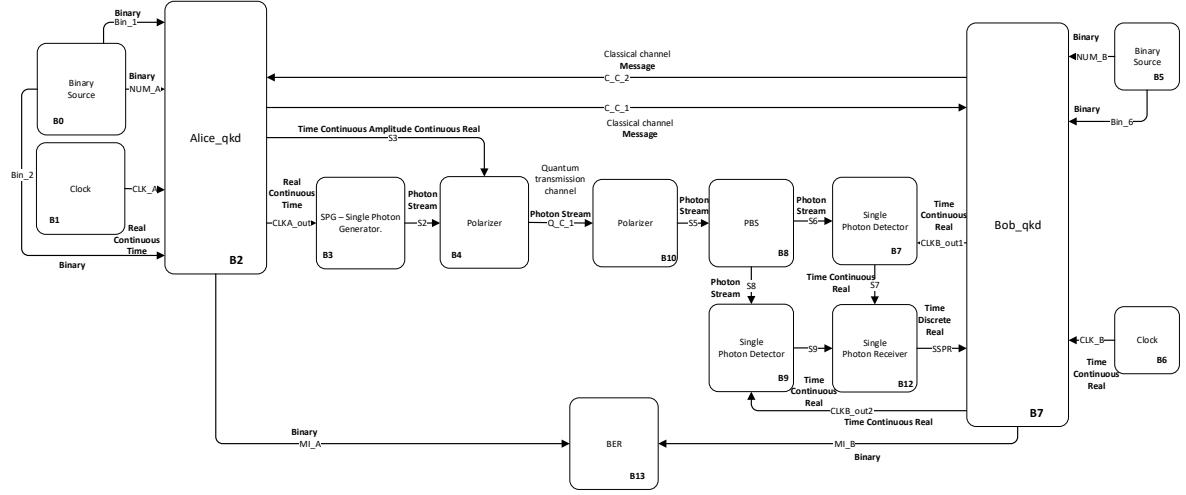


Figure 6.146: Diagram block of simulation performed between Alice and Bob until Basis Reconciliation.

and Bob outputs a binary signal "MI\_A" and "MI\_B", respectively. In case of no errors occurred in the quantum channel, these signals should be equal in order to both have the same sequence of bits. Furthermore, QBER between the two sequences should be 0. This way, Alice can encode messages using these keys and Bob will be capable of decrypt the message using these symmetric keys. When errors are introduced in quantum channel QBER value will increase as we can see later.

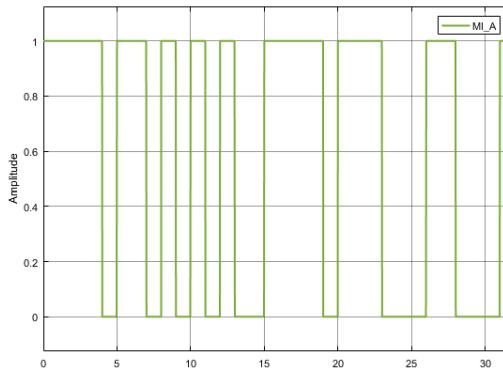


Figure 6.147: MI\_A signal.

Figure 6.147 and figure 6.148 represent the sequence of bits which will be used by Alice to encode the messages and the sequence of bits used by Bob to decode the message when no errors in quantum channel are taken into account, respectively. As one can see the two

signals are equal which meets the expected result. In this way, the first step of the protocol has been achieved.

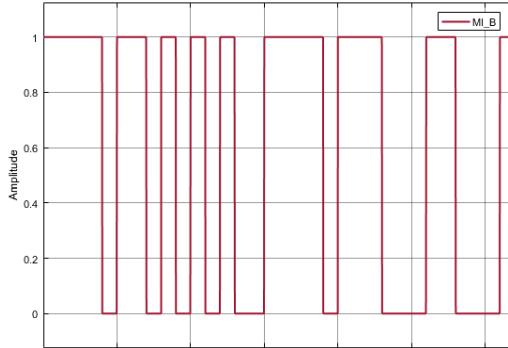


Figure 6.148: MI\_B signal.

As one can see in figure 6.160 a block which calculates QBER is connected to Alice and Bob. This block calculates the QBER between the measurements that Bob performed with the same basis as Alice, based on method described in [3]. Thus, as expected, the QBER is 0% when no errors are taken into account.

Next, some errors due the changes in state of polarization of the single photons transmitted between Alice and Bob were added. This way, a polarization rotator in the middle of the quantum channel was added, which is controlled by a SOP modulator block as it is shown in figure 6.149 with modelled with deterministic [4] and stochastic [5] methods. Additional information about the blocks presented in this quantum channel can be found in library chapter.

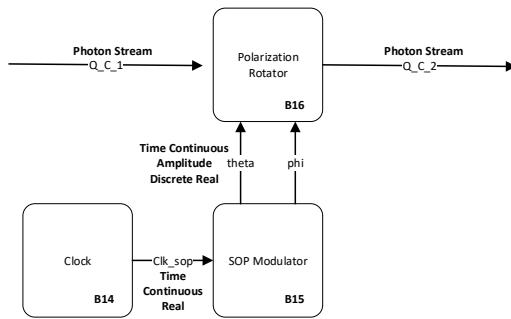


Figure 6.149: Quantum channel diagram.

Now, it is important to calculate the QBER as a function of the rotation angle  $\theta$ . In order to do that, it was simulated a deterministic SOP modulation, in which the  $\theta$  angle varies over the time. In figure 6.151 is presented the variation in the value of QBER with respect with theta changes from  $0^\circ$  to  $45^\circ$ . Theoretically, QBER corresponds to the probability of errors in

the channel. Which means that in practice this probability corresponds to the probability of a photon following the wrong path in the polarization beam splitter immediately before the detection circuit.

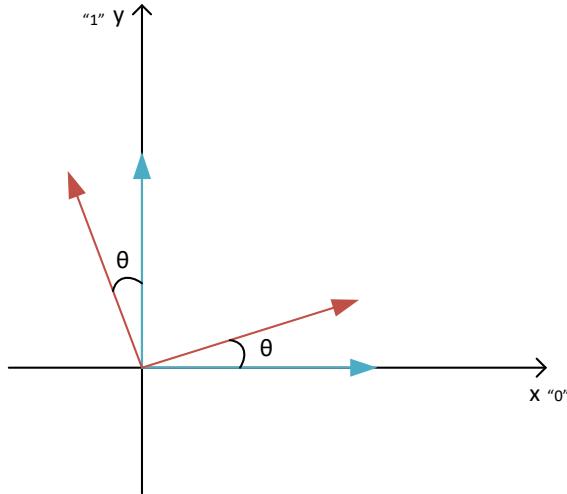


Figure 6.150: Representation of two orthogonal states rotated by an angle  $\theta$ .

Figure 6.150 presents the graphical representation of two orthogonal states rotated by an angle  $\theta$ . This rotation is induced by the SOP modulator block which selects a deterministic  $\theta$  and  $\phi$  angles that do not change over the time. This same rotation is applied for all sequential samples. From figure 6.150 the theoretical QBER can be calculated using the following equation:

$$QBER = P(0)P(1|0) + P(1)P(0|1). \quad (6.127)$$

Since we have been using a polarization beam splitter 50:50,

$$P(0) = P(1) = \frac{1}{2}.$$

This way,

$$QBER = \frac{1}{2}\sin^2(\theta) + \frac{1}{2}\sin^2(\theta) \quad (6.128)$$

$$QBER = \sin^2(\theta). \quad (6.129)$$

In figure 6.151 are represented two curves: QBER calculated from simulated data and QBER calculated using theoretical model from equation 6.128. Furthermore, the cross correlation coefficient between the two signals was calculated using a function from MATLAB `xcorr(x,y,'coeff')` which the result is 99.92%. From that, we can conclude that the QBER calculated from simulated data follows the theoretical curve with high correlation.

Nevertheless, the error bars presented in figure 6.151 were calculated based on a confidence interval of 95%.

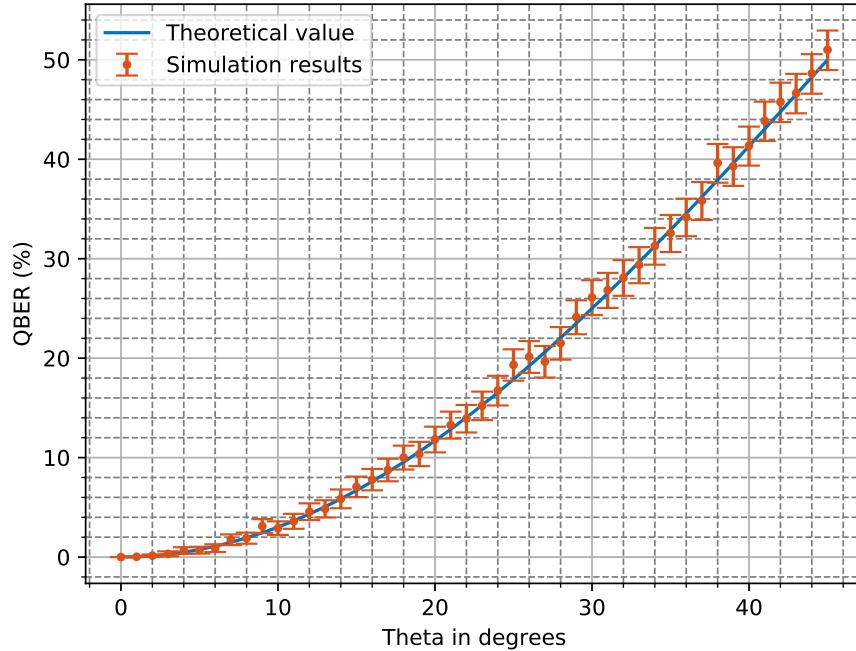


Figure 6.151: QBER evolution in relation with deterministic SOP drift.

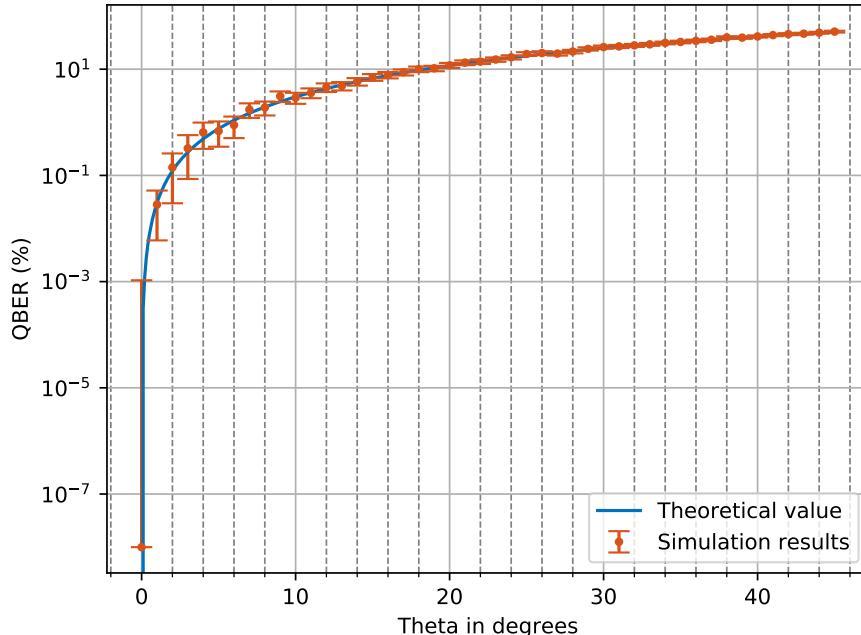


Figure 6.152: QBER evolution in relation with deterministic SOP drift in log scale.

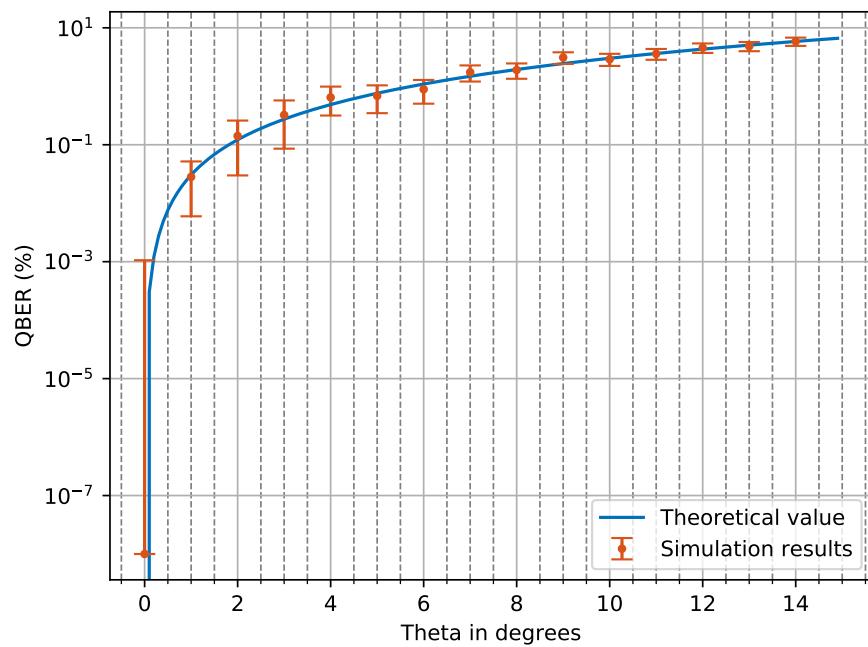


Figure 6.153: QBER evolution in relation with deterministic SOP drift scaled.

### 6.8.3 Open Issues

1. Include the random polarization rotations in the communication channel.
2. Implementation of the control system for polarization rotations.
3. Implementation of a QBER estimation protocol.
4. Implementation of the scrambling algorithm in order to spread the errors.
5. Implementation of the cascade for error correction.
6. Implementation of the output which represents the final key that is built.
7. Introduce EVE in simulation as shown in figure 6.154.

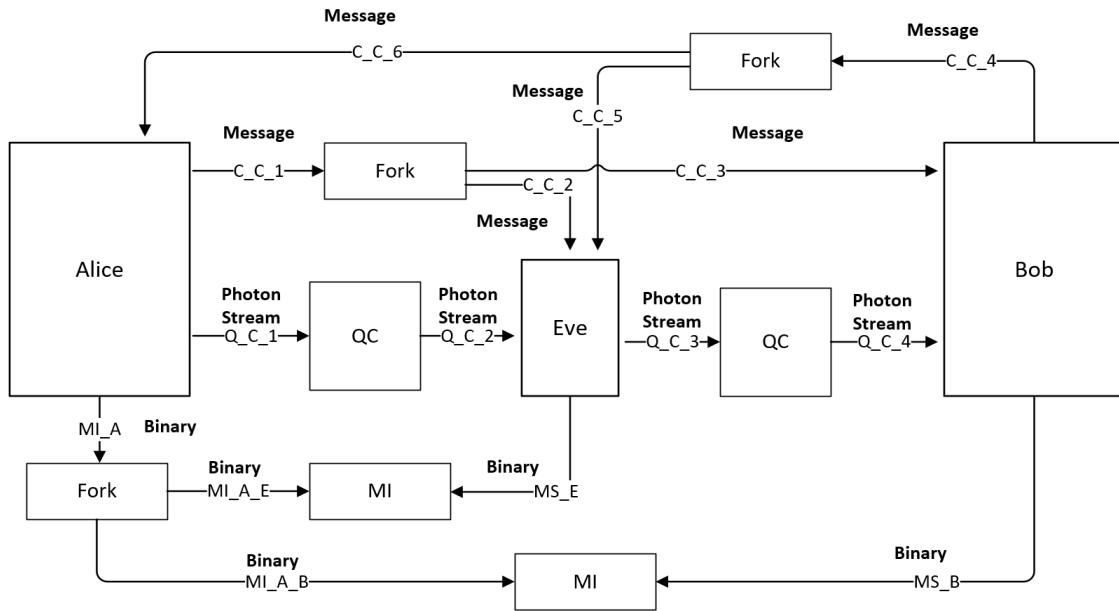


Figure 6.154: Simulation diagram at a top level

8. Analyze different strategies for Eve.
9. Experimental Implementation.

## References

- [1] Charles H Bennet. "Quantum cryptography: Public key distribution and coin tossing". In: *Proc. of IEEE Int. Conf. on Comp., Syst. and Signal Proc., Bangalore, India, Dec. 10-12, 1984.* 1984.
- [2] Christopher Gerry and Peter Knight. *Introductory quantum optics*. Cambridge university press, 2005.
- [3] Nelson J Muga, Mário FS Ferreira, and Armando N Pinto. "QBER estimation in QKD systems with polarization encoding". In: *Journal of Lightwave Technology* 29.3 (2011), pp. 355–361.
- [4] Nelson J Muga and Armando N Pinto. "Extended Kalman filter vs. geometrical approach for stokes space-based polarization demultiplexing". In: *Journal of Lightwave Technology* 33.23 (2015), pp. 4826–4833.
- [5] Cristian B Czegledi et al. "Polarization drift channel model for coherent fibre-optic systems". In: *Scientific reports* 6 (2016), p. 21217.

## 6.9 Quantum Oblivious Key Distribution with Discrete Variables

<b>Student Name</b>	:	Mariana Ramos
<b>Starting Date</b>	:	September 18, 2017
<b>Goal</b>	:	Quantum oblivious key distribution (QOKD) implementation with discrete variables.
<b>Directory</b>	:	sdf/ot_with_discrete_variables.

Oblivious Transfer (OT) is a fundamental primitive in multi-party computation. The one-out-of-two OT consists in a communication protocol between Alice and Bob. At the beginning of the protocol Alice has two messages  $m_1$  and  $m_2$  and Bob wants to know one of them,  $m_b$ , without Alice knowing which one, i.e. without Alice knowing  $b$ , and Alice wants to keep the other message private, i.e. without Bob knowing  $m_{\bar{b}}$ . therefore two conditions must be fulfilled:

1. The protocol must be concealing, i.e at the beginning of the protocol Bob does not know nothing about Alice's messages, while at the end of the protocol Bob will learn the message  $m_b$  chosen by him.
2. The protocol is oblivious, i.e Alice cannot learn anything about Bob's choice, bit  $b$ , and Bob cannot learning nothing about the other message  $m_{\bar{b}}$ .

In order to implement OT between two parties (Alice and Bob) they must be able to exchange continuously oblivious keys, i.e a QOKD system must exist between them.

### 6.9.1 Theoretical Description

#### Quantum Oblivious Key Distribution System (QOKD)

In this section we are going to describe the Quantum Oblivious Key Distribution system (QOKD). The QOKD system enables two parties (Alice and Bob) to share a set of keys. These keys have the particularity of being half right and half wrong. Only Bob knows which are right and wrong keys.

Considering a discrete variables implementation, both Alice and Bob agree with the following correspondence, where  $+$  corresponds to *Rectilinear Basis* and  $\times$  corresponds to *Diagonal Basis*,

<i>Basis</i>	
0	+
1	$\times$

Alice and Bob also agree with the bit correspondence for each direction for each basis. For *Rectilinear basis*, "+",

Basis "+"	
0	$\rightarrow (0^\circ)$
1	$\uparrow (90^\circ)$

and for *Diagonal Basis*, "x",

Basis "x"	
0	$\searrow (-45^\circ)$
1	$\nearrow (45^\circ)$

1. The first step is to establish for both Alice and Bob the block length  $l$ . In this case, lets assume  $l = 16$ . Alice randomly generate a bit sequence with length  $l$ . Therefore, she must define two sets randomly:  $S_{A1}$  which contains the basis values; and  $S_{A2}$ , which contains the key values.

In that case, lets assume she generates the following sets  $S_{A1'}$  and  $S_{A2'}$ :

$$S_{A1'} = \{0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1\},$$

$$S_{A2'} = \{1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1\}.$$

2. Next, Alice sends to Bob throughout a quantum channel  $l$  photons encoded using the basis defined in  $S_{A1'}$  and according to the key bits defined in  $S_{A2'}$ .

Therefore, in the current example, Alice sends the following photons,

$$\begin{aligned} S_{AB} &= \{\uparrow, \uparrow, \nearrow, \searrow, \rightarrow, \rightarrow, \searrow, \nearrow, \uparrow, \rightarrow, \searrow, \nearrow, \downarrow, \uparrow, \nearrow\} \\ &= \{90^\circ, 90^\circ, 45^\circ, -45^\circ, -45^\circ, 0^\circ, 0^\circ, -45^\circ, 45^\circ, 90^\circ, 0^\circ, -45^\circ, 45^\circ, -45^\circ, 90^\circ, 45^\circ\}. \end{aligned}$$

3. Bob also randomly generates  $l = 16$  bits, which are going to define his measurement basis,  $S_{B1'}$ . Lets assume,

$$\begin{aligned} S_{B1'} &= \{0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1\} \\ &= \{+, \times, \times, +, +, \times, +, \times, +, \times, +, +, +, \times\}. \end{aligned}$$

Bob will get  $l$  results:

$$S_{B2'} = \{1, -, \underline{0}, 0, -, 1, \underline{1}, -, 1, -, 1, 0, 1, 1, \underline{0}, 1\}.$$

The "−" corresponds to no clicks in Bob's detector, due to attenuation. The underlined values are bits which were measured with a correct basis but an error has occurred due to imperfections in the quantum communication system.

4. Bob is going to send a "-1" or a hash value to Alice for each measurement that he performed, thereby being "-1" the measurements which correspond to no clicks. In this case, we are going to assume that the hash value is calculated using the *SHA-256* algorithm [1]. In detail, Bob has two sets  $S_{B1'}$  and  $S_{B2'}$  and he is going to generate the set  $S_{BH1}$  with  $l$  values (" -1" or hash values calculated for each position of  $S_{B1'}$  with the correspondent position of  $S_{B2'}$ ). Therefore, Bob will send to Alice the following set:

$$S_{BH1} = \{S_1, -1, S_2, S_3, -1, S_4, S_5, -1, S_6, -1, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}\}.$$

5. Since Alice has received the confirmation of measurement from Bob, i.e after Alice has received  $S_{BH1}$ , she sends throughout a classical channel the basis which she has used to codify the photons updated with the information about the no received photons,

$$S_{A1'} = \{0, -1, 1, 1, -1, 0, 0, -1, 1, -1, 0, 1, 1, 1, 0, 1\}$$

Due to attenuation, the previous sets are reduced to the length 12 and they shall be replaced by the following:

$$S_{A1} = \{0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1\},$$

$$S_{A2} = \{1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1\},$$

$$S_{B1} = \{0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1\},$$

$$S_{B2} = \{1, \underline{0}, 0, 1, \underline{1}, 1, 1, 0, 1, 1, \underline{0}, 1\}$$

Note that  $S_{B2}$  still has errors.

6. In order to know which photons were measured correctly, Bob does the operation  $S_{B3} = S_{B1} \oplus S_{A1}$ . In the current example,

$$\begin{array}{c|cccccccccccc} S_{B1} & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline S_{A1} & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ \oplus & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$$

In this way, Bob gets

$$S_{B3} = \{1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1\}.$$

When Bob uses the right basis he gets the values correctly, apart from possible errors in transmission, when he uses the wrong basis he just guess the value. The values "1" correspond to the values he measured correctly and "0" to the values he just guessed. Thus, Bob is building two sets of keys, one with correct basis measurements values and other with the wrong basis measurement values that he just guessed.

Thus, Bob has two pair of sets, one for the right basis,

$$S_{B_{rp}} = \{1, 2, 5, 6, 8, 11, 12\},$$

$$S_{B_{rb}} = \{1, 0, 1, 1, 0, 0, 1\},$$

where  $S_{B_{rp}}$  is the set of positions and  $S_{B_{rb}}$  is the set of bit values he measured for each position. The other pair is for photons he measured with the wrong basis and then he just guessed the values,

$$S_{B_{wp}} = \{3, 4, 7, 9, 10\},$$

$$S_{B_{wb}} = \{0, 1, 1, 1, 1\},$$

where  $S_{B_{wp}}$  is the set of positions and  $S_{B_{wb}}$  is the set of bit values he measured for each position.

Nevertheless, due to errors in transmission, some bits in  $S_{B_{rb}}$  may be not right.

At this point, in order to test Bob's honesty and to estimate the QBER of the channel, Alice is going to ask Bob to open some pairs of the Bob's sets. The definition of the protocol to test Bob's honesty is still an open issue. However, depending on the QBER estimated by her, Alice must have a parameter to set the number of right position she wants to open, i.e she must open a minimum number of right position in order to guarantee a minimum QBER. This will increase the security of the protocol. Alice chooses some positions to open and tells Bob which positions she wants to open. Bob sends to Alice the pairs she chose and then these pairs are eliminated from them sets. Lets assume she asked to open the positions 10, 11 and 12. If she concludes Bob is not being honest, she stops the protocol and they must start it again. Otherwise, the protocol continues. Lets assume Alice has verified these pairs using the hash function committed by Bob and concluded that he is being honest. Therefore, she sends to Bob the QBER estimated by her.

Now, Bob has the previous sets replaced by the following,

$$S_{B_{rp}} = \{1, 2, 5, 6, 8\}$$

$$S_{B_{rb}} = \{1, 0, 1, 1, 0\}$$

$$S_{B_{wp}} = \{3, 4, 7, 9\}$$

$$S_{B_{wb}} = \{0, 1, 1, 1\}$$

Bob is going to use a modified version of *Cascade algorithm* to correct the errors due transmission.

### Modified version of Cascade Algorithm

The Cascade algorithm is often used with a key set where all values are supposed right. In this case, Bob has two pairs of sets, one with the position and bit values of photon he measured with the correct basis and other with position and bit values of photon

he measured with the wrong basis. He only needs to apply the Cascade algorithm in the set that he measured the photons correctly [2]. However, he must apply a modified version of the Cascade in the other set in order to keep in secret from Alice which set corresponds to right and which set corresponds to wrong measurements.

Bob randomly generates a bit value. If he gets 0, he will send to Alice the set  $\{S_{B_{rp}}, S_{B_{wp}}\}$ . Otherwise, if he gets 1 he will send the set  $\{S_{B_{wp}}, S_{B_{rp}}\}$ . This guarantee that Alice does not know which is the right or wrong set. Lets assume this random bit is "0" and he sends  $\{S_{B_{rp}}, S_{B_{wp}}\}$ .

- (a) Bob starts by applying the normal cascade to the set  $S_{B_{rb}}$ . After both know the error estimative Bob determine if the error rate is above the fail threshold. If it is truth they must start the procedure again. Lets assume the estimated error rate is acceptable. Bob and Alice use a random permutation which is represented in figure 6.155 for a larger number of bits (agreed at the beginning) by applying it to the shifted keys, in order to guarantee the spread out of the error bits randomly and to separate consecutive errors from each other.

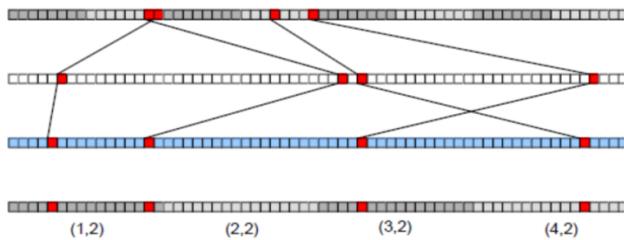


Figure 6.155: Cascade Algorithm - permutation

- (b) Bob and Alice divide all the shifted key bits into blocks of  $N$  bits depending on the estimated error rate in order to have one or no error per block. In general, the sets of keys are too large and it is easier to explain the algorithm based on a larger number of bits. Therefore, figure 6.156 represents the typical cascade initial steps. However, in this case, the set to be corrected only has five bits, therefore they divide the set in two sub-blocks, one with 3 bits and other with 2 bits.
- (c) They use a classical channel to compare the block parities. For blocks with different parities, an odd number of errors must exist, otherwise an even number of errors would mask each other. Thus, the block in which the parities disagree is divided in half into two smaller blocks of length  $\frac{N}{2}$ , and another parity check is performed on the first sub-block, as one can see in figure 6.157. As it was referred above, there is at least one error in one sub-block being the error location revealed by the parity of one sub-block. In other words, if the parity of the first sub-block passes, the error will be in the second sub-block. The sub-block with error will be sub-divided until the error is found.

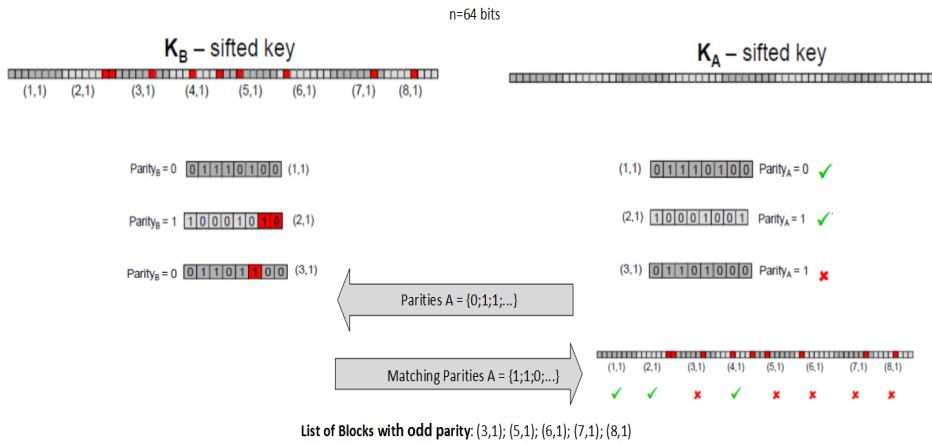


Figure 6.156: Cascade Algorithm

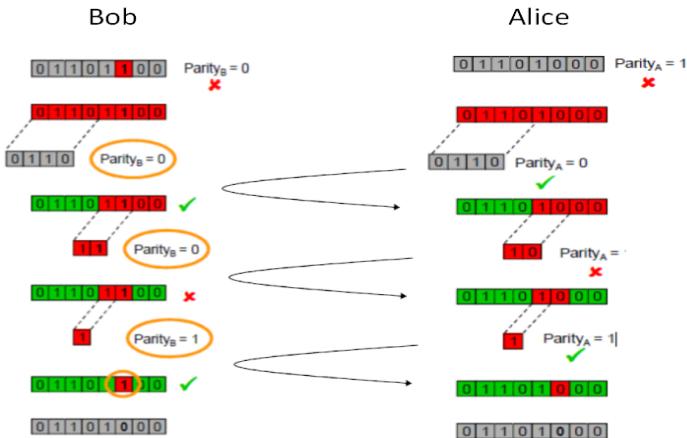


Figure 6.157: Cascade Algorithm - example of error correction

- (d) When the error is corrected, the last bit of the block is discarded in order to prevent the gain of additional information by Bob.

In this case, let's assume the set of right positions was corrected with the algorithm described above and it will be replaced by the following:

$$S_{B_{rp}} = \{1, 2, 5, 6, 8\}$$

$$S_{B_{rb}} = \{1, 1, 0, 1, 0\}$$

In order to test Alice's honesty, Bob must verify if the QBER sent by Alice is a realistic value. If it is not he stops the protocol and they must start again. Otherwise, the protocol continues.

After that, Bob needs to apply the Fake Cascade to the set  $S_{B_{wb}}$ . The main goal of this

step is to convince Alice she is performing the real Cascade but she is not.

- (a) First of all, based on the positions contained in  $S_{B_{wb}}$ , Bob must build an array with the correspondent bits in a random order and informs Alice the order of positions. In order to best explain this version of the algorithm, lets assume a larger set of bits.

Bob sends to Alice throughout a classical channel the new positions order as if it were the permutation step represented in figure 6.155 in real Cascade algorithm.

- (b) Assuming each of them has a set with 32 bits randomly organized by Bob, they divide the supposed shifted keys in blocks with  $N$  bits according to the estimated error rate. As the  $QBER$  is the same as for real cascade, Bob will assume the same number of errors, even if he starts for this modified version he can know the number of errors from  $QBER$  estimated by Alice.
- (c) Bob and Alice use a classical channel to compare the block parities. Alice sends to Bob her parity list. Based on Alice's parity list, Bob sends a block list with odd parities, i.e the blocks position in which parity supposed disagree. This list is randomly built based on the number of errors considered by Bob, i.e if he considered five errors from  $QBER$  estimative, he will distributed them randomly and after that he will fill the remaining spaces with even parities. Bob sends to Alice the set with the list of odd parities, i.e the list of sub-sets he has different parities than Alice.
- (d) The blocks with errors will be consecutively divided until they found the supposed errors. Since we have assumed there were five errors, this is the number of errors that Alice must supposedly correct.

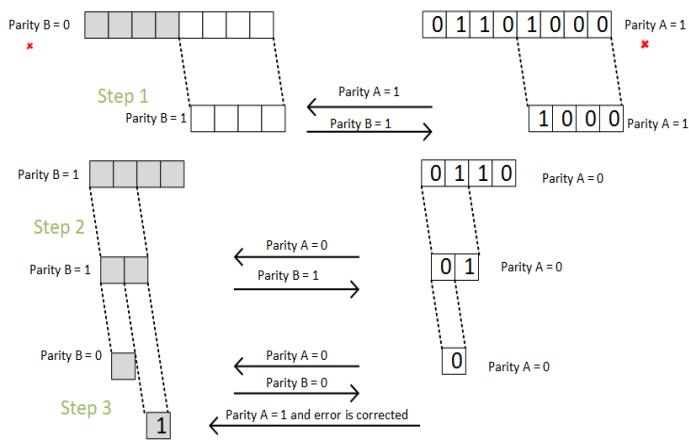


Figure 6.158: Fake cascade - example of error correction

Lets assume one of the blocks with error and analyse figure 6.158. Bob starts with a set filled with random bits, therefore we do not need to know which bits are. Alice starts by dividing her set in half with two blocks with  $N$  bits.

**Step 1:** Bob chooses one of the  $t$  blocks and informs Alice she must send the parity of this block. Lets assume he chose sub-block 2. She sends the parity and Bob is going to send his parity, which after know Alice's block parity he send the opposite parity. As referred in normal Cascade, there must be one error or no error in each block. Thus, since the parities disagree, the error must be in seconde block. They start the procedure to correct it.

**Step 2:** Bob divides again the sub-block in half with  $\frac{N}{2}$  bits and asks Alice for the parity of the first sub-block. She sends her parity equals to 0 and Bob sends to her the opposite parity again.

**Step 3:** They divide the sub-block in half again and Bob asks Alice for the parity of the first bit. Alice sends to him the parity equals to her. As the error is not in the first be, it must be in the second, therefore Bob is able to correct this bit with the information sent by Alice.

Note that Bob make his choice of which half analyse first using a random bit generator result. If he got "0" he starts with the first half of the sub-block, otherwise, if he got "1", he starts with the second half. In addition, they must discard the last bit of each block and sub-block in which fake Cascade were applied in order to guarantee that Bob does not gain additional information.

In this case, after apply the fake Cascade to  $S_{B_{wb}}$ , lets assume,

$$\begin{aligned} S_{B_{wp}} &= \{3, 4, 7, 9\} \\ S_{B_{wb}} &= \{0, 1, 1, 0\} \end{aligned}$$

If Bob starts by applying the fake Cascade, he must test Alice's honesty at the beginning of the real Cascade application, based on the number of errors he has. If he thinks that the *QBER* sent by Alice is unrealistic, he stops the protocol at this point.

7. When Alice sends to Bob a photons set, they are building a set of pairs (array positions and bit values which correspond to measured photons at Bob's side and to the key bit with the photon was encoded at Alice's side). The main goal is to guarantee that Bob has the same number of right and wrong pairs. In addition, they must know information about  $t$  (represented in figure 6.159) which corresponds to the points where the previous condition is verified.

Since Bob has sent to Alice the information about the smallest set, in this example, Alice know that there are four pairs of wrong positions and five pairs of right positions. Alice must destroy one of the right pairs by asking Bob to open it. Therefore, at  $t = 8$  both know that there are the same number of right and wrong pairs thereby being the main goal guaranteed.

As we can see in figure 6.159, unlike Bob, Alice does not know which positions corresponds to right or wrong measurements performed by Bob. They have been building these sets during all protocol.

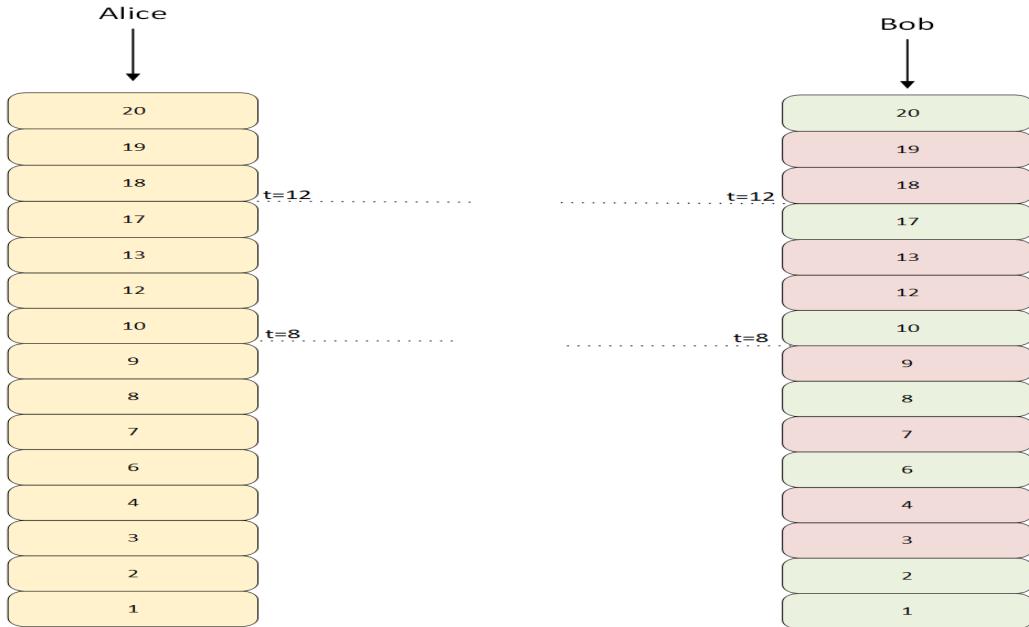


Figure 6.159: Alice and Bob key sets.

### 1-out-of-2 Oblivious Transfer Protocol with QOKD system

At this time, we are going to describe the oblivious transfer protocol with detail. As it was referred at the beginning, Alice sends two messages to Bob and he wants to know one of them. Alice does not know which message Bob wants and Bob only know the message he wants, i.e he does not know anything about the other message. Furthermore, only Alice knows information about messages  $m_0$  and  $m_1$ . In this case, lets assume the following two messages with size  $s = 4$ ,  $m_0 = \{0011\}$  and  $m_1 = \{0001\}$ . Alice must guarantee  $t = s \times 2$ . In order to do that, she must destroy the remaining pairs. In this case, there is no need to do that because they have a set for  $t = 8$  with the same number of wrong and right pairs.

1. Bob defines two new sub-sets,  $I_0$  and  $I_1$ .  $I_0$  is a set of values with photons array positions which Bob just guessed the measurement since he did not measure them with the same basis as Alice,  $I_1$  is a set of values with photons array positions which Bob measured correctly since he used the same basis as Alice used to encoded them. The position of the pairs of each right and wrong message are in the keys sets that they have been building during the protocol.

In this example, the message size is 4. Since, at this time  $t = 10$  and we have 5 right pairs and 5 wrong pairs, Alice ask to Bob to open one right pair and one wrong pair in order to both have exactly the message's size number of right and wrong pairs. Lets assume that Alice opened two pairs, position 15 which is a wrong measurement and position 10 which is a right measurement. We have now  $t = 8$ .

Next, Bob defines two sub-sets with size  $s = 4$ :

$$I_0 = \{3, 4, 7, 9\},$$

and

$$I_1 = \{1, 2, 6, 8\},$$

where  $I_0$  is the sequence of positions in which Bob was wrong about basis measurement and  $I_1$  is the sequence of positions in which Bob was right about basis measurement. Bob sends to Alice the set  $S_b$

Thus, if Bob wants to know  $m_0$  he must send to Alice throughout a classical channel the set  $S_0 = \{I_1, I_0\}$ , otherwise if he wants to know  $m_1$  he must send to Alice throughout a classical channel the set  $S_1 = \{I_0, I_1\}$ .

2. Alice is sure about Bob's honesty, since she knows he only has 4 right basis to measure the photons. In addition, Alice cannot know which message Bob chose because she did not know the order that he sent the sets.
3. Lets assume Bob sent  $S_0 = \{I_1, I_0\}$ . Alice defines two encryption keys  $K_0$  and  $K_1$  using the values in positions defined by Bob in the set sent by him. In this example, lets assume:

$$K_0 = \{1, 1, 1, 0\}$$

$$K_1 = \{0, 0, 0, 1\}.$$

Alice does the following operations:

$$m = \{m_0 \oplus K_0, m_1 \oplus K_1\}.$$

$$\begin{array}{c|cccc} m_0 & 0 & 0 & 1 & 1 \\ \hline K_0 & 1 & 1 & 1 & 0 \\ \oplus & 1 & 1 & 0 & 1 \end{array}$$

$$\begin{array}{c|cccc} m_1 & 0 & 0 & 0 & 1 \\ \hline K_1 & 0 & 0 & 0 & 1 \\ \oplus & 0 & 0 & 0 & 0 \end{array}$$

Adding the two results,  $m$  will be:

$$m = \{1, 1, 0, 1, 0, 0, 0, 0\}.$$

After that, Alice sends to Bob the encrypted message  $m$  through a classical channel.

4. When Bob receives the message  $m$ , in the same way as Alice, Bob uses  $S_{B1}$ , values of positions given by  $I_1$  and  $I_0$  and does the decrypted operation. In this case, he does following operation:

$$\begin{array}{c|cccccccc} m & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \oplus & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}$$

The first four bits corresponds to message 1 and he received  $\{0, 0, 1, 1\}$ , which is the right message  $m_0$  and  $\{0, 1, 1, 0\}$  which is a wrong message for  $m_1$ .

### Nearest Private Query

The Nearest Private Query is another example of QOKD application [Xu2017]. In this case, there are also two parties: a user (who we called Bob) and a data owner (who we called Alice). Bob has an input secrete parameter  $x$  and Alice has a private data set  $A$ .

Lets assume Bob's secret input parameter  $x = 8$  and Alice's data set  $A = \{1, 2, 3, 6, 7, 10, 11, 14\}$ .

Bob wants to know which element  $(x_i)$  is the closest to  $x$  in Alice data set  $A$ , without revealing his secrete  $x$ . Alice does not know which is the Bob's secret parameter, and Bob does not know any information about Alice's set except the closest element  $x_i$  to his  $x$ .

**Step 1** Alice generates a new set with  $N = 2^n$  elements,  $D(j)$  for  $j = 0, 1, \dots, N - 1$ , in which  $D(j) = x_l$  being  $x(l)$  the closest element to  $j$  in  $A$ .  $n$  is the number of bits that Alice needs to represent each element of her data set.

$j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$D(j)$	1	1	2	3	3	6	6	7	7	10	10	11	11	14	14	14
	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	0

**Step 2** Bob and Alice have a set of keys  $K_i^*$  and  $K_i$ , respectively, with 64 elements where 60 are bits resulted from wrong basis measurement and 4 resulted from correct basis measurement. This allows Alice to know that Bob is being honest. They start from QOKD with symmetric keys and then Alice is being asking Bob to destroy pairs of keys until they have a set with the characteristics above.

**Step 3** Bob sends to Alice the set  $S$  with all positions referred random measurements except the position bits at  $j = 8$ .

**Step 4** Alice encoded all bits with the corresponding bit keys at the positions sent by Bob, and then she sends the encoded message to Bob.

**Step 5** Bob receives the encoded message and apply an operation XOR based on the correspondent bits to the positions he sent two Alice above.

**Step 6** At the end, Bob gets the message he wanted, the closest element to 8, which corresponds to message 0, 1, 1, 1 or 7 and he remains know nothing about the other elements of Alice's data set. In the same way, Alice does not know which element Bob wants to know.

### QKD from QOKD

All Quantum Key Distribution systems can be obtained from the QOKD system presented in this report. The Quantum Oblivious Key Distribution system allows to obtain symmetric secret keys and symmetric or asymmetric oblivious keys.

In fact, since Alice and Bob has the same set of keys and at some tab Alice knows that there are the same number of right and random bits, she can obtain any combination from this set by asking Bob to destroy some pairs.

QOKD has a big advantage over other QKD systems because of its versatility. However, the biggest disadvantage is related with a large number of photons consumption which can became the communication rate slower.

#### 6.9.2 Simulation Analysis

First of all, the protocol will be simulated and then a experimental setup will be built in the laboratory.

The main goal of this simulation is to demonstrate that Bob was able to learn correctly message  $m_b$  and he does not know the message  $m_{\bar{b}}$ .

As one may see in figure 6.160 this simulation will have three top level blocks. Two of them are Alice and Bob and they are connected through two classical channels and one quantum channel. In addition, a third block will be performed in order to calculate the *Mutual Information*. The mutual information (MI) between Alice and Bob is defined in terms of their joint distribution.

1. In figure 6.161 one can observe a block diagram of the simulation at Alice's side. As it is shown in the figure, Alice must have one block for random number generation which is responsible for basis generation to polarize the photons, and for key random generation in order to have a random state to encode each photon. Furthermore, she has a Processor block for all logical operations: array analysis, hash function results validation, random number generation requests, and others. This block also receives the start information, i.e. message size  $s$  and messages  $m_0$  and  $m_1$ , as well as information from Bob, i.e sets  $I_0$  and  $I_1$ , hash function results, and others. In addition it is responsible for set the initial length  $l$  of the first array of photons which will send to Bob. This block also must be responsible for send classical information to Bob. Finally, Processor block will also send a real continuous time signal to single photon generator, in order to generate photons according to this signal, and finally this block

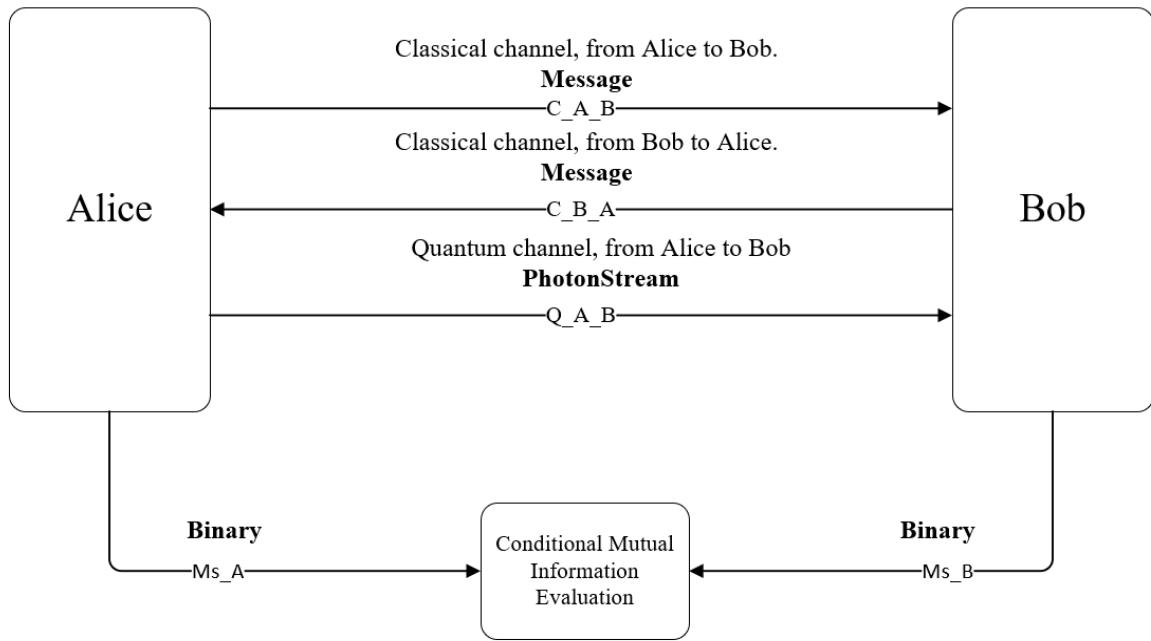


Figure 6.160: Simulation diagram at a top level

also sends to polarizer a real discrete signal in order to inform the polarizer which basis it should use. Therefore, she has two more blocks for quantum tasks: the single photon generator and the polarizer block which is responsible to encode the photons generated from the previous block and send them throughout a quantum channel from Alice to Bob.

Finally, Alice's processor has an output to Mutual Information top level block,  $Ms_A$ .

2. In figure 6.162 one can observe a block diagram of the simulation at Bob's side. From this side, Bob has one block for Random Number Generation which is responsible for randomly generate basis values which Bob will use to measure the photons sent by Alice throughout the quantum channel. Furthermore, this Block will generate the random bits that Bob needs in Modified Version of Cascade Algorithm. Like Alice, Bob has a Processor block responsible for all logical tasks, i.e Hash function generation, analysing functions, requests for random number generator block, etc. Additionally, it receives information from Alice throughout a classical channel and a quantum channel but it sends information to Alice only throughout a classical channel. Furthermore, Bob has one more block for single photon detection which receives from processor block a real discrete time signal, in order to obtain the basis it should use to measure the photons.

Finally, Bob's processor has an output to Mutual Information top level block,  $Ms_B$ .

3. Mutual Information calculation

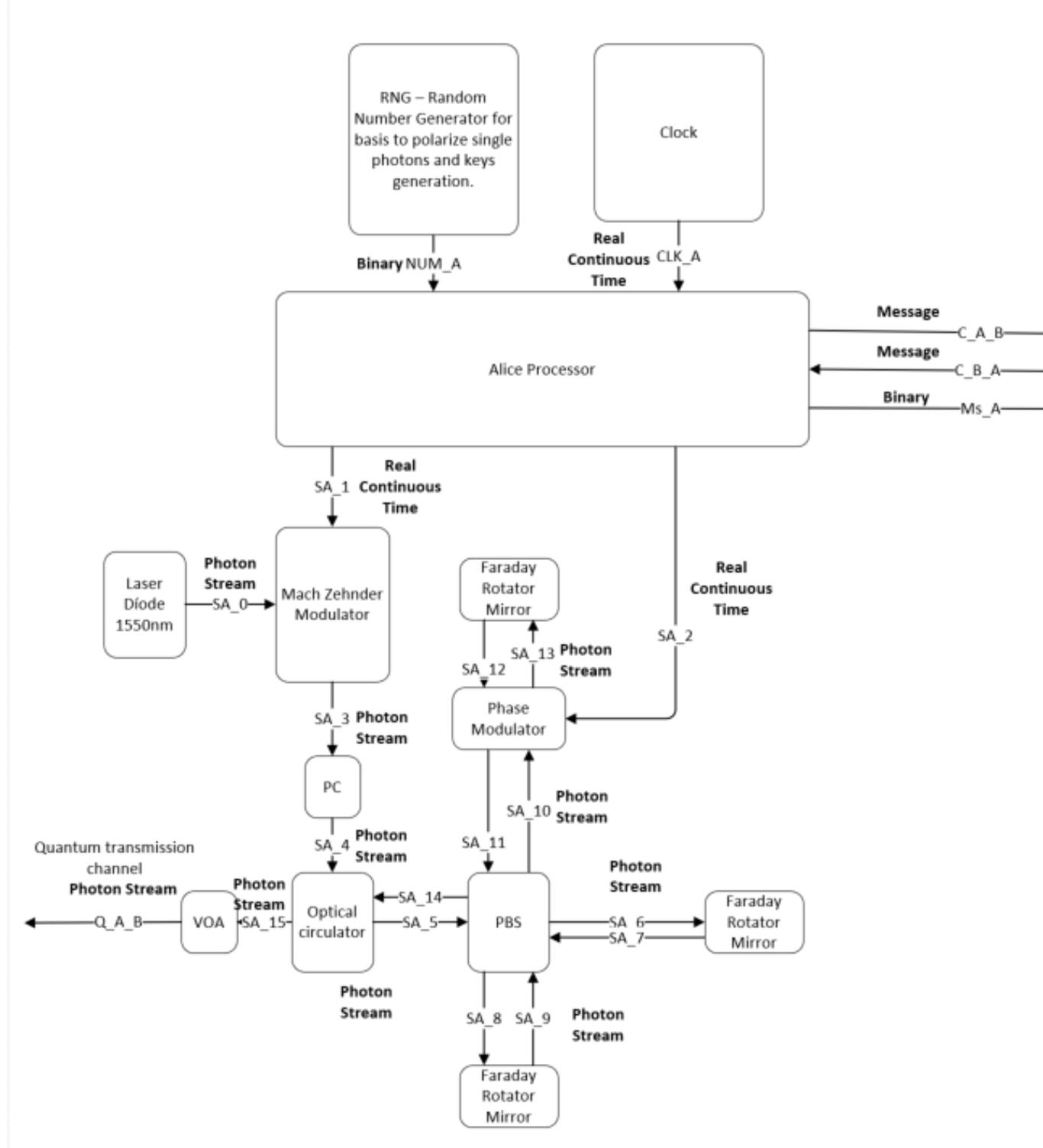


Figure 6.161: Simulation diagram - Alice's side

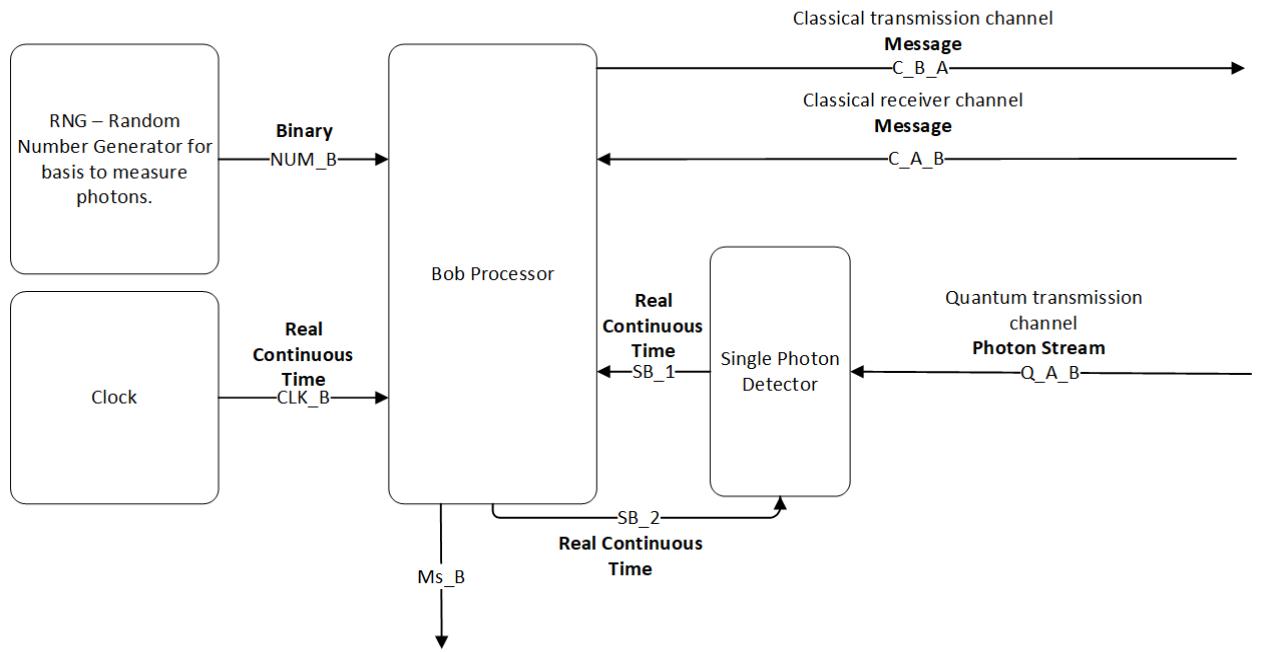


Figure 6.162: Simulation diagram - Bob's side

Table 6.30: System Signals

Signal name	Signal type	Status
NUM_A	Binary signal	
NUM_B	Binary signal	
CLK_A	Real continuous Time	
CLK_B	Real continuous Time	
C_A_B	Message	
C_B_A	Message	
SA_1	Real Continuous Time	
SA_2	Real Continuous Time	
SA_3	Real Discrete Time	
SA_3	Real Continuous Time	
Q_A_1	Photon Stream	
Q_A_2	Photon Stream	
Q_A_B	Photon Stream	
Ms_A	Binary	
Ms_B	Binary	
S_B1	Real continuous Time	
S_B2	Real continuous Time	

Table 6.31: System input parameters

Parameter	Default Value	Description
messageSize	4	Size of the message Alice must send to Bob.
blockLength	16	Block length.
symbolRate	100K	

Table 6.32: Header Files

File name	Description	Status
random_number_generator.h		
single_photons_generator.h		
single_photons_detector.h		
encorder.h		
decoder.h		
messageToSend.h		
message_toReceive.h		
mutual_information.h		
cascade_truth.h		
cascade_fake.h		
Sha256.h		

Table 6.33: Source Files

File name	Description	Status
random_number_generator.c		
single_photons_generator.c		
single_photons_detector.c		
encorder.c		
decoder.c		
messageToSend.c		
message_toReceive.c		
mutual_information.c		
QOKD_main.c		
cascade_truth.c		
cascade_fake.c		
Sha256.c		

### 6.9.3 Experimental Setup

In figure 6.163 are presented the experimental setup to be performed in the lab. The main goal is to build an experimental setup in which Alice and Bob communicate through two

classical channels and one quantum channel that will have only one direction (Alice to Bob).

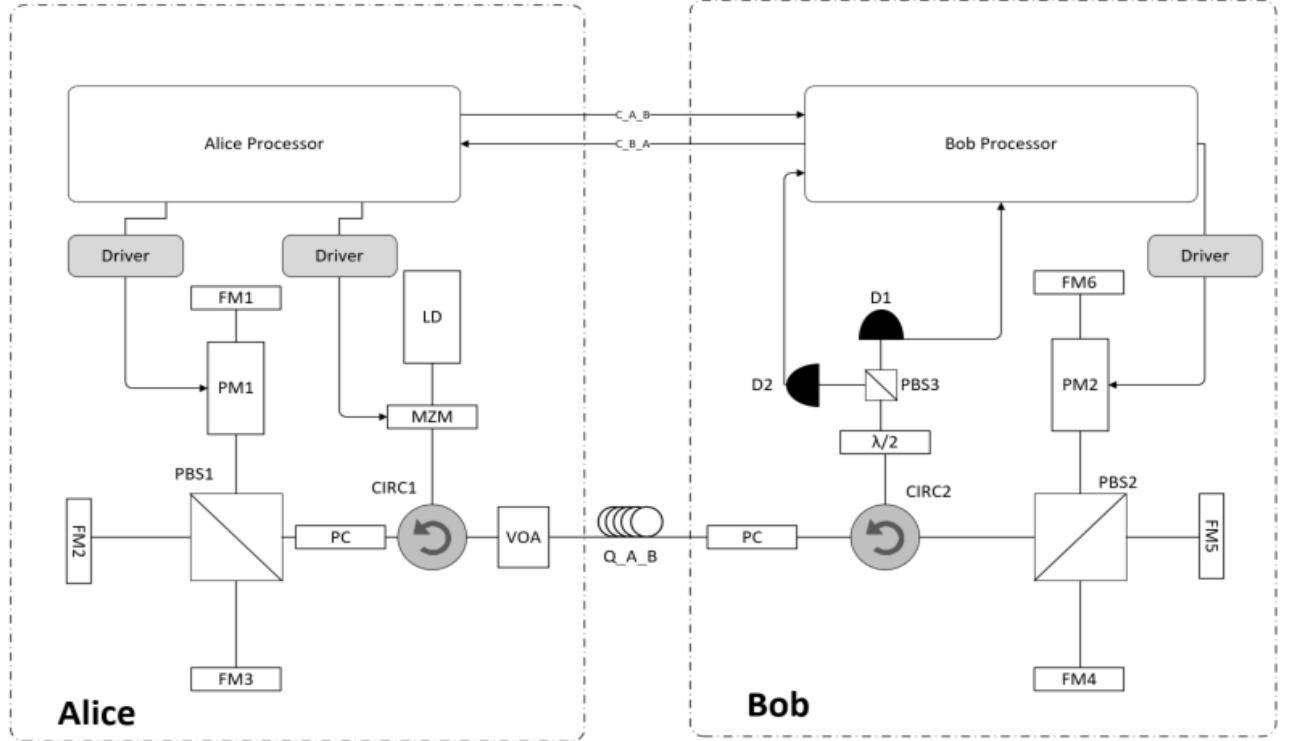


Figure 6.163: QOKD Experimental setup

The laser **LD** emits light with a wavelength of 1550 nm and then the light passes throughout a Mach-Zehnder Modulator (**MZM**) in order to have pulsed light. The light is polarized with a polarizer controller and gets a linear polarization of 45°. When the light reaches the polarization beam splitter, **PBS1**, single photon pulses in 45° linearly polarization state

$$|in\rangle = |45\rangle = \frac{\sqrt{2}}{2}(|H\rangle + |V\rangle)$$

and each pulse is divided in **PBS1** into two orthogonal components  $P_x$  and  $P_y$ .  $P_x$  is directly transmitted in **FM2** direction and it is reflected by it with its stated rotated, which means it has a new direction  $|V\rangle$ . This way, when it reaches again **PBS1** is reflected to **FM3** direction and it happens the same being the  $P_x$  component transmitted through the **PBS1** in **FM1** direction passing through the phase modulator and suffers a certain phase shift. Relatively to vertical component which reaches the **PBS1** and was reflected to **FM1** direction and passes through the phase modulator, **PM**, that applies a phase shift to it, than it follows to the **FM1** and is reflected rotated by 90°, meaning that it follows to **FM3** and it is reflected at its original state. At the end, the two components recombine in **PSB1** and the single photon pulse follows its path with a polarization state defined by the phase shift applied in **PM**.

In table 6.34 is present the phases shifts that must be applied at Phase Modulator in order to get the polarized photons chose by Alice.

In table 6.35 are described all components needed to build the experimental setup.

Table 6.34: Different Alice's output polarization states based on shift phases applied in Phase Modulator.

Alice: $(\phi_1, \phi_2)$	Output Polarization
$(\frac{\pi}{2}, 0)$	Vertical
$(\frac{\pi}{2}, \frac{3\pi}{2})$	Horizontal
$(0, 0)$	$-45^\circ$
$(0, \frac{\pi}{2})$	$45^\circ$

Table 6.35: List of material

Material Name	Quantity	Status
Laser semiconductor 1550nm	1	✓
Manual polarization controller	5	
Faraday Mirror (FM)	6	
Mach-Zehnder Modulator	1	✓ 2.56GHz
Single Photon Detector	2	✓
Phase modulator	2	
Four-port polarization beam splitter	2	
Three-port polarization beam splitter	1	
Half-wave plate	1	
Optical circulator	2	
Variable Optical Attenuator	1	✓
Computer	1	

#### 6.9.4 Comparative Analysis

## References

- [1] Tie-Ming Liu et al. "Researching on Cryptographic Algorithm Recognition Based on Static Characteristic-Code". In: *Security Technology: International Conference, SecTech 2009, Held as Part of the Future Generation Information Technology Conference, FGIT 2009, Jeju Island, Korea, December 10-12, 2009. Proceedings*. Ed. by Dominik Ślęzak et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 140–147. ISBN: 978-3-642-10847-1. DOI: [10.1007/978-3-642-10847-1\\_18](https://doi.org/10.1007/978-3-642-10847-1_18). URL: [https://doi.org/10.1007/978-3-642-10847-1\\_18](https://doi.org/10.1007/978-3-642-10847-1_18).
- [2] Gilles Brassard and Louis Salvail. "Secret-Key Reconciliation by Public Discussion". In: *Advances in Cryptology — EUROCRYPT '93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings*. Ed. by Tor Helleseth. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 410–423. ISBN: 978-3-540-48285-7. DOI: [10.1007/3-540-48285-7\\_35](https://doi.org/10.1007/3-540-48285-7_35). URL: [https://doi.org/10.1007/3-540-48285-7\\_35](https://doi.org/10.1007/3-540-48285-7_35).

## 6.10 Quantum Noise

<b>Contributors</b>	:	Diamantino Silva, (2017-01-23 - ...)
	:	Armando Pinto (2017-01-23 - ...)
<b>Goal</b>	:	Simulation and measurement of quantum noise in double homodyne detection.

Quantum noise is an intrinsic property of light, a manifestation of the vacuum field fluctuations [1]. Contrarily to the majority of noise sources, that are overcomed by better equipment, quantum noise has a lower bound, given by the Heisenberg uncertainty principle, which cannot be broken. This property can be useful in some areas, such in quantum cryptography, were many protocols depend on it to ensure their security. The objective of this work is to develop a numerical model for the quantum noise in a homodyne detection and to validate the numerical model with experimental results.

$$\hat{a} |n\rangle = \sqrt{n} |n-1\rangle \quad (6.130)$$

$$\hat{a}^\dagger |n\rangle = \sqrt{n+1} |n+1\rangle \quad (6.131)$$

$$\hat{n} |n\rangle = n |n\rangle \quad (6.132)$$

### 6.10.1 Theoretical Analysis

In this section, we assume the following transmission system

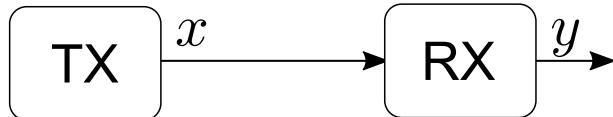


Figure 6.164: Model of a linear transmission system

We can approximate the output by the following relation [2]

$$y = tx + z \quad (6.133)$$

$y$  is the received signal,  $x$  is the transmitted signal,  $t = \sqrt{\eta T}$ , where  $\eta$  accounts for the detector efficiency and  $T$  accounts for the transmittance of the channel, and the total noise is modeled by the random variable  $z$ . We are also going to assume that the random variable  $z$  is normal distributed with mean 0 and variance  $\sigma_z^2 = \eta T \xi + N_0 + v_{el}$  which is the superposition of all noise sources assumed to be independent from each other and independent from the signal  $x$ . These sources are the shot noise characterized by a gaussian variable with mean 0 and variance  $N_0$ , the detector's electronic noise characterized by a gaussian variable with mean 0 and variance  $v_{el}$  and the excess noise, which is the equivalent noise added in the channel, accounted at the transmitter output and characterized by a gaussian variable with

mean 0 and variance  $\xi$ . We are going to assume that  $x$  is a gaussian distributed random variable with mean 0 and variance  $\sigma_x^2$ . With the previous assumptions we have

$$\langle x \rangle = \langle z \rangle = \langle y \rangle = 0 \quad (6.134)$$

with variances (and covariances)

$$\begin{aligned} \langle x^2 \rangle &= \sigma_x^2 \\ \langle xy \rangle &= t \langle x^2 \rangle + \langle xz \rangle = \sqrt{\eta T} \sigma_x^2 \\ \langle y^2 \rangle &= t^2 \langle x^2 \rangle + 2t \langle xz \rangle + \langle z^2 \rangle = \eta T \sigma_x^2 + \eta T \xi + N_0 + v_{el} \end{aligned} \quad (6.135)$$

in which  $\langle xz \rangle = 0$ , given that  $x$  and  $z$  are defined as independent variables.

### 6.10.2 Numerical Analysis

To test the previous model, we simulate an optical communication system. Diagram 6.165 shows the block setup used in the simulation

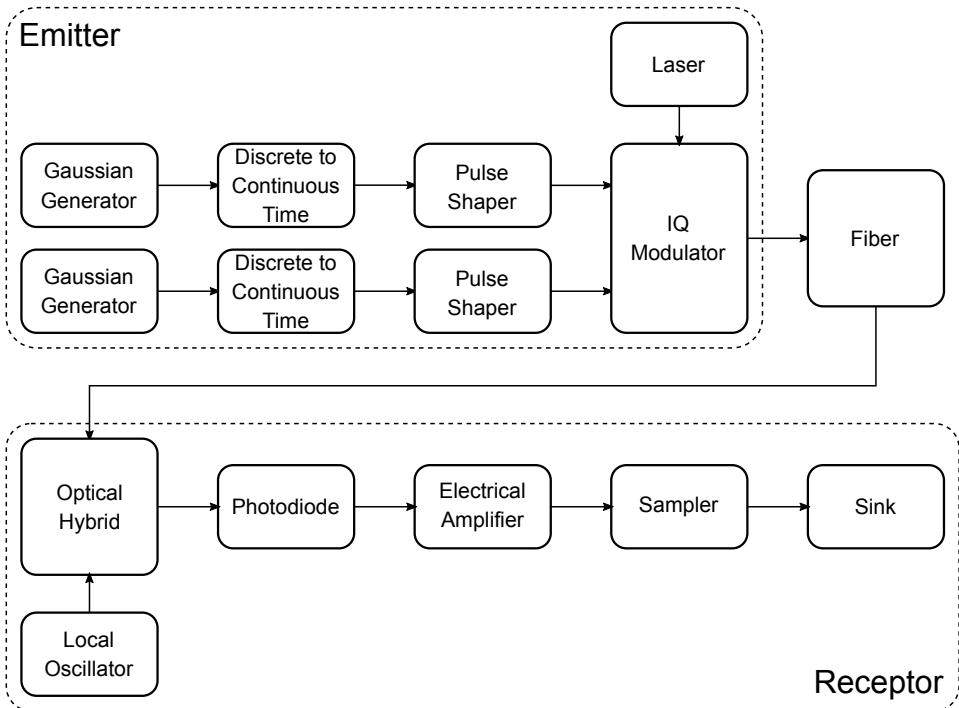


Figure 6.165: Block diagram of the simulation

The simulation starts by generating two random numbers  $x_I$  and  $x_Q$ , both following gaussian distributions with the same mean 0 and same variance  $\sigma_x^2$ . An optical signal with I and Q modulation is generated, accordantly with  $x_I$  and  $x_Q$ , and sent through the fiber to

the receptor. The receptor will perform a double balanced homodyne detection, outputing the following two currents proportional to each quadrature [3]

$$i_I(t) = R\sqrt{P_S P_{LO}} \cos [\theta_S(t) - \theta_{LO}(t)] \quad i_Q(t) = R\sqrt{P_S P_{LO}} \sin [\theta_S(t) - \theta_{LO}(t)] \quad (6.136)$$

in which  $i_I$  is the current proportional to the in-phase component,  $i_Q$  is the current proportional to the quadrature component,  $P_S$  is the signal's instantaneous power,  $P_{LO}$  is the local oscillator instantaneous power,  $R = \eta e/\hbar\omega$  is the detector responsivity,  $\theta_S$  is the signal phase and  $\theta_{LO}$  is the local oscillator phase.

Assuming a local oscillator power much higher than the signal power, the expected variance for both currents is [4]

$$(\Delta i)^2 = \frac{1}{2} R^2 P_\lambda [4\mu^2 P_{LO} N_0 + P_{LO} \mu (1 - \mu)] \quad (6.137)$$

in which  $P_\lambda = \frac{hc}{\lambda \Delta t}$  and  $\Delta t$  is the sample duration. For continuous wave coherent states,  $N_0 = \frac{1}{4}$ .

In our simulation, the input of a photodiode is a sequence of samples, corresponding to the mean amplitude of the electric field during the sample duration. Using this input, we can obtain the incident average power in the photodiode with  $\bar{P}(t) = |A(t)|^2$  which corresponds to the average number of photon in sample duration  $\bar{n} = \frac{\bar{P}(t)}{P_\lambda}$ . To model the shot noise present in a photodiode detection, we will assume that the number of photons detected during a given interval,  $n(t)$ , is a random variable following a Poisson distribution with mean  $\bar{n}$ . The output current of the photodiode will be  $i(t) = R P_\lambda n(t)$ .

The homodyne detection consists in three fundamental steps: mixing the signal with a local oscillator in a balanced beam splitter, converting each of the two output beams into a current and finally subtracting one current from the other. In our simulation, we simulate a double balanced homodyne detector, which is composed by two balanced homodyne detectors, each one for measuring one of the signals quadrature. To keep our analysis simple, we will focus on the in-phase homodyne detector, which have the following electric fields incident to each of it's photodides. Assuming that the photocurrents of the two photodiodes are stochastically independent, then:

$$(\Delta i(t))^2 = R^2 P_\lambda^2 ((\Delta n_1(t))^2 + (\Delta n_2(t))^2) = \frac{1}{2} R^2 P_\lambda^2 (\bar{n}_S + \bar{n}_{LO}) \quad (6.138)$$

These two results are in agreement with the previous obtained results 6.136 and 6.137 if  $\mu = 1$ .

Before constructing a complete continuous variable transmission system we started by simulating a simplified setup, by removing all sources of noise, to test the generation of quantum noise in the photodiodes.

The constelation was simplified to one single state  $|\sqrt{\bar{n}_S/2} + i\sqrt{\bar{n}_S/2}\rangle$ , the fiber has no loss and no noise and all the electronic elements had their noises turned off.

The simulations has two main input parameters  $\bar{n}_S$  and  $\bar{n}_{LO}$ . The following plots show

the current variance  $(\Delta i(t))^2$  in shot noise units, in function of the input signal number of photons per sample, for the  $I$  and  $Q$  quadratures. Each curve corresponds to a local oscillator number of photons per sample.

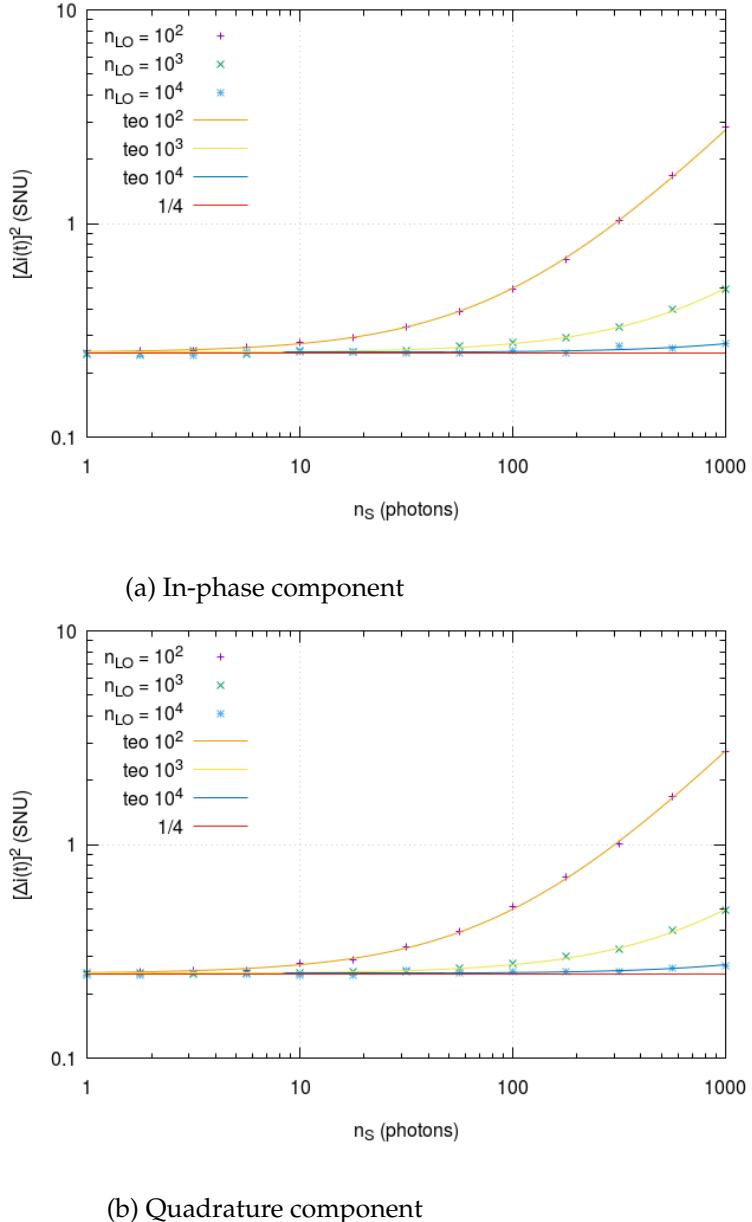


Figure 6.166: Current variance in function of the signal  $\bar{n}_S$  and the local oscillator  $\bar{n}_{LO}$  number of photons.

For each pair  $(\bar{n}_S, \bar{n}_{LO})$ , 10000 signals were used to simulate the transmission. We see that the simulation results follow the theoretical prediction. Also, for higher  $\bar{n}_{LO}$  values, the variance becomes independent of the signal's number of photons, converging to the theoretical value  $\frac{1}{4}$ .

### 6.10.3 Experimental Analysis

#### Measurement of quantum noise

The different noise sources that compose the total noise, scale differently in function of the local oscillator power. Electronic noise does not scale at all, quantum noise scales with the root square and classical noise scales linearly.

## References

- [1] Mark Fox. *Quantum Optics: An Introduction*. Oxford University Press, 2006.
- [2] Tao Wang et al. "Practical performance of real-time shot-noise measurement in continuous-variable quantum key distribution". In: *Quantum Information Processing* 17.1 (2018), p. 11.
- [3] Govind P Agrawal. *Fiber-optic communication systems*. "Third". John Wiley & Sons, 2002.
- [4] Rodney Loudon. *The Quantum Theory of Light*. Oxford University Press, 2000.

## 6.11 Frequency and Phase Recovery in CV-QC Systems

<b>Student Name</b>	: Daniel Pereira (01/05/2017 - )
<b>Goal</b>	: Theoretical and simulation study of techniques for frequency and phase recovery in CV-QC systems.
<b>Directory</b>	: sdf/cv_system
<b>Related Links</b>	: <a href="https://www.overleaf.com/14348245sjvzbkzpxwx">https://www.overleaf.com/14348245sjvzbkzpxwx</a>

Continuous Variable Quantum Communications (CV-QC) can encode information in the phase of weak coherent states. The weak nature of these states makes noise a substantial impairment, as such it is important to study the optimal techniques for noise compensation.

### 6.11.1 Classical Frequency and Phase Recovery - State of the art

The traditional method for compensating noise in classical coherent communications is to use either optical or electrical phase-locked loop, synchronizing the frequency and the phase of the local oscillator (LO) with the transmitter laser [1]. Alternatively, digital signal processing (DSP) can be used to perform frequency offset compensation and phase recovery can be performed in the digital domain [2].

#### Frequency Mismatch Compensation Techniques

Frequency mismatch occurs when the central frequencies of the transmission and reception local oscillators aren't equal, as illustrated in Figure 6.167. A frequency offset of  $2\Delta f$

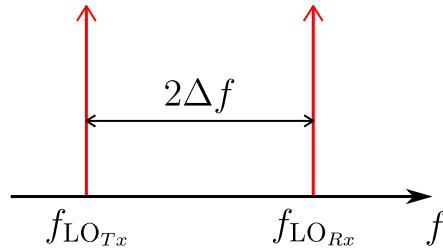


Figure 6.167: Visual representation of an optical frequency offset.

introduces a phase of  $2\pi\Delta f T n$  to the  $n$ th symbol in relation to the 0th symbol. Compensation is accomplished by first estimating  $\Delta f$  and then removing the added phase for each symbol [2]. Frequency estimation can be accomplished through a blind frequency search method [3], in which the frequency is scanned over a range, symbol decisions made and the minimum square error is used as the frequency-selection criteria. Alternatively the frequency can be estimated by evaluating the spectrum of the 4th power of the input signal [4], which exhibits a peak at the frequency  $4\Delta f$ . A third option involves the usage of training symbols inserted periodically with the data payload [5]. These training symbols

are used to remove the applied phase modulation. The resulting signal  $s(n)$  is used to obtain an estimate of the offset frequency as

$$\Delta f_{\text{est}} = \frac{1}{2\pi T} \arg \left\{ \sum_{n=1}^L s(n)s^*(n-1) \right\} \quad (6.139)$$

### Phase Mismatch Compensation Techniques

Phase mismatch occurs when the two employed oscillators have differing optical phases, as illustrated in Figure 6.168. This causes the phase modulation applied to one of the signals

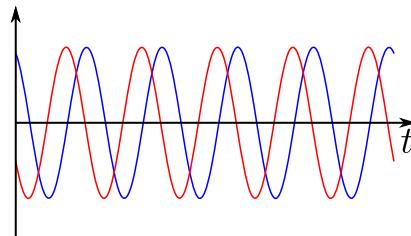


Figure 6.168: Visual representation of a phase mismatch between two cosines.

to be misread in the detection scheme. Furthermore, phase noise arises from the non-zero linewidth of the transmission and reception local oscillators, as illustrated in Figure 6.169. Phase noise consists of a time evolving phase mismatch, which can be modelled as a Weiner



Figure 6.169: Phase noise arises from the non-zero linewidth of the transmission and reception oscillators, being present even if their central frequencies match.

process described as

$$\phi(t_k) = \phi(t_{k-1}) + \Delta\phi(k), \quad (6.140)$$

where  $\Delta\phi(k)$  are phase increments, which are independent and identically distributed random Gaussian variables with zero mean and variance

$$\sigma^2 = 2\pi\Delta f|t_k - t_{k-1}|. \quad (6.141)$$

The phase mismatch can be compensated either through a blind estimation technique [6] or through a pilot-aided estimation technique [7]. We assume here that the signal at this

point is given by

$$x(n) = x_{\text{sym}}(n)e^{i\theta(n)}, \quad (6.142)$$

where  $x_{\text{sym}}(n)$  is the complex, phase-modulated signal and  $\theta(n)$  is the phase noise contribution.

In the blind estimation method a fourth-order non-linearity is used to remove the phase modulation, yielding an estimate of the phase mismatch [2]. This method requires the usage of the four state modulation in the form

$$x_{\text{sym}} = e^{i\frac{(2k+1)}{4}\pi}, \quad k \in \{0, 1, 2, 3\}. \quad (6.143)$$

The phase estimation is given by [6]

$$\theta_{\text{est}}(n) = \frac{1}{4} \arg \left\{ \frac{1}{2L+1} \sum_{l=-L}^L f(l) x^4(n+l) \right\}, \quad (6.144)$$

where  $f(l)$  is a weighting function. The 4th power effectively removes the phase modulation, explicitly

$$x_{\text{sym}}^4 = e^{i(2k+1)\pi} = -1, \quad k \in \{0, 1, 2, 3\}, \quad (6.145)$$

thus the only remaining phase is due to the phase noise contribution. The resulting sum  $-\frac{1}{2L+1} \sum_{l=-L}^L f(l) e^{i\theta(n+l)}$  is basically a weighted average of the phase noise contributions up to  $L$  symbols before and  $L$  symbols after the current one.

In the pilot aided scheme, pre-agreed on symbols are inserted, time multiplexed, with the data payload at a pilot rate of  $1/P$  (meaning one pilot symbol is inserted after  $P - 1$  data symbols). The phase mismatch is determined from the pilot symbols and this estimation is used to compensate the phase mismatch on the data carrying symbols [7].

### 6.11.2 Quantum Frequency Mismatch and Carrier Phase Noise Compensation - State of the art

High sensitivity, low thermal noise coherent receivers are necessary in order to measure quantum coherent states [8]. This limits the bandwidth of the receiver [9, 10]. To avoid the added noise from disparities between the transmitter laser and the LO laser, the first CV-QC protocols used a high intensity signal sent polarization multiplexed with the quantum signal, to be used as the LO in the detection scheme [11, 12]. This co-propagating technique presents a security risk, as tampering on the signal pulses can be hidden by altering some property of the LO [13, 14, 15, 16], as well as introduces experimental complexity [17].

Locally generated LO (LLO) CV-QC protocols were proposed simultaneously in [18] and [17]. Given their low intensity, frequency offset compensation and phase recovery methods cannot be applied directly to the quantum pulses [17]. In [18, 17] phase recovery is based on a pilot aided estimation scheme, with the pilot signal consisting of high intensity reference pulses sent time-multiplexed with the quantum pulses. In [19] a similar LLO CV-QC scheme was proposed with the reference pulses being extracted from the same wavefront as the quantum pulses, contrasting with the methods proposed in [18, 17],

in which the signal and reference pulses are carved sequentially from the continuously running transmission local oscillator. In [19, 18, 17] the frequency mismatch was assumed to be minimal, the resulting noise it introduces being treated as extra phase mismatch. Alternatively, applying blind frequency search and frequency domain methods, both similar to classical frequency estimation DSP, to the pilot signal has been proposed [20].

Despite recent works on LLO CV-QC schemes, no optimallity study of the proposed schemes in function of symbol period and/or other properties has been presented. Furthermore, despite recent studies on DSP techniques, no frequency mismatch compensation technique taking advantage of the properties of the shared pilot signal has been presented. The purpose of this text is to present both of these. This work starts by describing the novel frequency mismatch compensation technique in detail, including a comparison with previously proposed compensation schemes, followed by a validation of the method by applying it to both the sequential and self-referenced LLO CV-QC scheme. An optimality study is performed to determine the ideal scheme in multiple situations.

### 6.11.3 Novel Frequency and Phase Mismatch Compensation Algorithm

Both the sequential [18, 17] and self [19] referenced methods for LLO CV-QC return two final constellations, obtained from the signal and reference pulses, given respectively by

$$\begin{aligned} x_S(l) &= |x_S| e^{i(\omega_B t_m + \phi(t_l) - \varphi(t_l) + \theta(t_l))} & , l = 2n + 1 \\ x_R(m) &= |x_R| e^{i(\omega_B t_m + \phi(t_m) - \varphi(t_m))} & , m = 2n \end{aligned} \quad , n \in \mathbb{Z}. \quad (6.146)$$

$\omega_B$  is the offset frequency,  $\phi(t_n)$  and  $\varphi(t_n)$  are the instantaneous phase of the transmitter and receiver laser, respectively, at the moment  $t_n = (n + \frac{1}{4}) T_s$  and  $\theta(t_n)$  is the phase modulation applied to the n-th signal symbol. For the purpose of this demonstration we assume the sequential referenced method.

Laser phase noise can be modeled as a Weiner process described as

$$\theta(t_n) = \theta(t_{n-1}) + \Delta\theta(t_n, t_{n-1}), \quad (6.147)$$

where  $\Delta\theta(t_n, t_{n-1})$  are the phase increments, which are independent and identically distributed random Gaussian variables with zero mean and variance given by

$$\sigma^2 = 2\pi\Delta f |t_n - t_{n-1}|, \quad (6.148)$$

where  $\Delta f$  is the full-width at half maximum of the laser's frequency spectrum and  $t_{n/n-1}$  are two different sampling moments at which the phase is evaluated.  $\omega_B$  can be determined by multiplying an incremented reference constellation by the complex conjugate of the unincremented constellation, being in this regard similar to the training symbol aided technique observed previously [5]. The resulting signal is given by

$$x_{\text{freq}} = x_R(m+2)x_R^*(m) = |x_R|^2 e^{i(\omega_B 2T_s + \phi(t_{m+2}) - \phi(t_m) - (\varphi(t_{m+2}) - \varphi(t_m)))}. \quad (6.149)$$

The frequency estimation is accomplished by taking the complex argument of (6.149) and dividing it by  $2T_s$

$$\omega_{\text{est}} = \frac{1}{2T_s} \arg(x_{\text{freq}}) = \omega_B + \frac{\phi(t_{m+2}) - \phi(t_m) - (\varphi(t_{m+2}) - \varphi(t_m))}{2T_s} + k\pi T_s^{-1}, \quad m = 2n, \quad n \in \mathbb{Z}, \quad (6.150)$$

where the  $k\pi T_s^{-1}$  term is due to the non-injectiveness of the  $\arg$  function, forcing the value of  $\omega_{\text{est}} 2T_s$  to remain within the  $[0, 2\pi]$  interval. The value of  $k$  here is bounded by

$$0 \leq \omega_{\text{est}} 2T_s \leq 2\pi \iff -\frac{\omega_B T_s}{\pi} \leq k \leq -\frac{\omega_B T_s}{\pi} + 1. \quad (6.151)$$

$\omega_{\text{est}}$  is a normally distributed random variable with expected value

$$\mathbb{E}[\omega_{\text{est}}] = \omega_B + k\pi T_s^{-1}, \quad -\frac{\omega_B T_s}{\pi} \leq k \leq -\frac{\omega_B T_s}{\pi} + 1 \quad (6.152)$$

and variance

$$\begin{aligned} \text{Var}[\omega_B] &= \text{Var}[\omega_B + k\pi T_s^{-1}] + \text{Var} \left[ \frac{\phi(t_{m+2}) - \phi(t_m) - (\varphi(t_{m+2}) - \varphi(t_m))}{2T_s} \right] \\ &= \frac{1}{4T_s^2} (\text{Var}[\phi(t_{m+2}) - \phi(t_m)] + \text{Var}[(\varphi(t_{m+2}) - \varphi(t_m))]) \\ &= \frac{4\pi\Delta f 2T_s}{4T_s^2} = 2\pi\Delta f T_s^{-1}. \end{aligned} \quad (6.153)$$

The frequency mismatch compensated signals are obtained by multiplying the constellations in 6.154 by  $e^{-i\mathbb{E}[\omega_{\text{est}}]t_n}$ , resulting in the constellations

$$\begin{aligned} \overline{x_S(l)} &= |x_S| e^{i(-k\pi(2n+\frac{5}{4}) + \phi(t_l) - \varphi(t_l) + \theta(t_l))}, \quad l = 2n + 1, \quad n \in \mathbb{Z}, \quad -\frac{\omega_B T_s}{\pi} \leq k \leq -\frac{\omega_B T_s}{\pi} + 1. \\ \overline{x_R(m)} &= |x_R| e^{i(-k\pi(2n+\frac{1}{4}) + \phi(t_m) - \varphi(t_m))}, \quad m = 2n \end{aligned} \quad (6.154)$$

The phase modulated signal is recovered by multiplying the signal constellation with the complex conjugate of the reference constellation, yielding

$$x(n) = |x_S| |x_R| e^{i(-k\pi + \phi(t_{2n+1}) - \phi(t_{2n}) - (\varphi(t_{2n+1}) - \varphi(t_{2n})) + \theta(t_{2n+1}))}, \quad n \in \mathbb{Z}, \quad -\frac{\omega_B T_s}{\pi} \leq k \leq -\frac{\omega_B T_s}{\pi} + 1, \quad (6.155)$$

from this result we see that the uncertainty introduced by the non-injectiveness of the  $\arg$  function introduces a phase of a multiple of  $-\pi$ . Depending on the parity of  $k$  this either yields the original constellation or one rotated by  $\pi$ . This is easily solved by performing the steps after the frequency and phase recovery for both possibilities.

The exactness of  $\omega_{\text{est}}$  can be evaluated through a confidence interval. Assuming that the variance is not well known, being estimated from the experimental results, the confidence interval is given by

$$P \left( \omega_B - t \frac{s}{\sqrt{n}} < \omega_{\text{est}} < \omega_B + t \frac{s}{\sqrt{n}} \right) = 1 - \alpha, \quad (6.156)$$

where  $t$  is the  $1 - \frac{\alpha}{2}$ 'th percentile of the Student's t-distribution and  $s^2$  is the estimated value of the variance. For the a offset frequency of  $\omega_B = 10$  MHz, assuming  $n = 2 \times 10^5$  samples were used and that the variance estimate is given by  $s^2 = 2\pi\Delta f T_s^{-1}$ , the 99 % confidence interval is less than 1 % of  $\omega_B$ .

### Comparative analysis of the frequency ranging techniques

The figure of merit employed for this comparison will be the Error Vector Magnitude (EVM), where the error is evaluated as the relation between the magnitude of the error vector  $e_V$  and the magnitude of the vector of the ideal symbol position  $ref_V$

$$EVM(\%) = 100 \sqrt{\frac{|e_V|}{|ref_V|}} \quad (6.157)$$

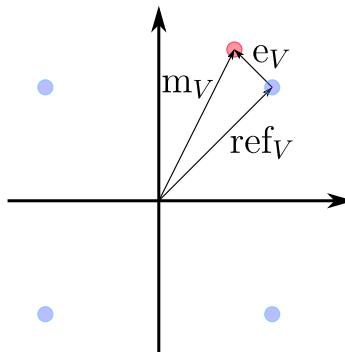


Figure 6.170: Visual representation of the EVM method in a QPSK constellation. The ideal constellation points are presented in blue, while an example for an actual measured symbol is presented in red.

The three frequency mismatch compensation techniques described in Section 6.11.1 were applied on simulated signal and reference constellations similar to the ones presented in (6.154). The blind frequency search method was applied to the pilot signal following the method described in [3], utilizing a coarse step size of 10 MHz followed by a fine step size of 1 MHz. The spectral analysis method was applied via evaluation of the spectrum of the reference constellation. A null phase noise was assumed for this study, since it would be the dominant contributor to the EVM and would not allow a good comparison between the techniques being studied.

The EVM of the recovered constellation (6.155) was computed in function of the sampling frequency. The results for this study are presented in Figure 6.171. Note that the blind frequency search and spectral methods fail to compensate for the frequency mismatch compensation at the lower sampling frequencies attempted (note the missing data points for sampling frequencies below  $10^7$  Hz for these methods). In both cases this is due to aliasing caused by the low sampling frequency when compared to the absolute value of the frequency mismatch. Our novel method can compensate for the frequency mismatch reliably even for extremely low sampling rates.

For all the tested sampling frequencies, our novel method returns consistently EVM values 4 orders of magnitude below the classical alternatives, thus delivering results of much greater precision.

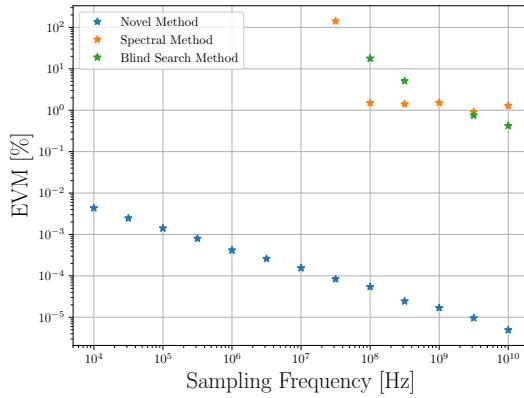


Figure 6.171: EVM in function of the sampling frequency for 3 different frequency ranging techniques, assuming no phase noise, for a frequency mismatch of 51921371 Hz (value chosen at random).

This technique has a computational complexity of  $O(n)$ . In comparison, the blind frequency estimation technique requires carrier phase recovery, bit decision and computing of the least mean square error for every test frequency, even assuming a frequency and phase recovery and decision algorithm with a total computational complexity of  $O(n)$ , the added least mean square error step will push the computational complexity above the one observed in our novel technique. The spectrum aided technique requires a Fourier Transform to be performed, which in itself has a computational complexity of  $O(n \log(n))$ , this is above the computational complexity of our novel technique for  $n > e$ .

#### 6.11.4 Validation of Proposed Method

In this section we describe in depth the Continuous Variables Quantum Key Distribution (**CV-QKD**) systems analysed here. The CV-QKD system enables two parties (Alice and Bob) to share a secret key to be employed in a symmetric encryption protocol. The following CV-QKD employs discrete modulation of coherent states, resulting in the Quadrature Phase Shift Keying (**QPSK**) constellation presented in Figure 6.172. The state to bit correspondence, agreed by Alice and Bob beforehand, also presented in Figure 6.172, is as follows:

Coherent State	Bits
$ \alpha_0\rangle =  \sqrt{n}e^{i\frac{\pi}{4}}\rangle$	01
$ \alpha_1\rangle =  \sqrt{n}e^{i\frac{3\pi}{4}}\rangle$	11
$ \alpha_2\rangle =  \sqrt{n}e^{i\frac{5\pi}{4}}\rangle$	10
$ \alpha_3\rangle =  \sqrt{n}e^{i\frac{7\pi}{4}}\rangle$	00

in which  $n$  is the average number of photons per symbol.

Two different systems will be studied, differing in the technique employed to generate the reference pulse utilized to compensate for the Phase Noise observed in the two laser

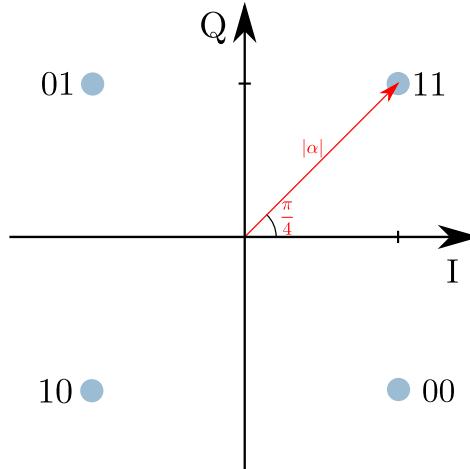


Figure 6.172: QPSK symbol constellation.

setup. The first setup extracts the reference and signal pulses from two different optical wavefronts, while the second setup extracts them from the same wavefront. A detailed diagram of the studied system is presented in Figure 6.173, which will be used in the following studies.

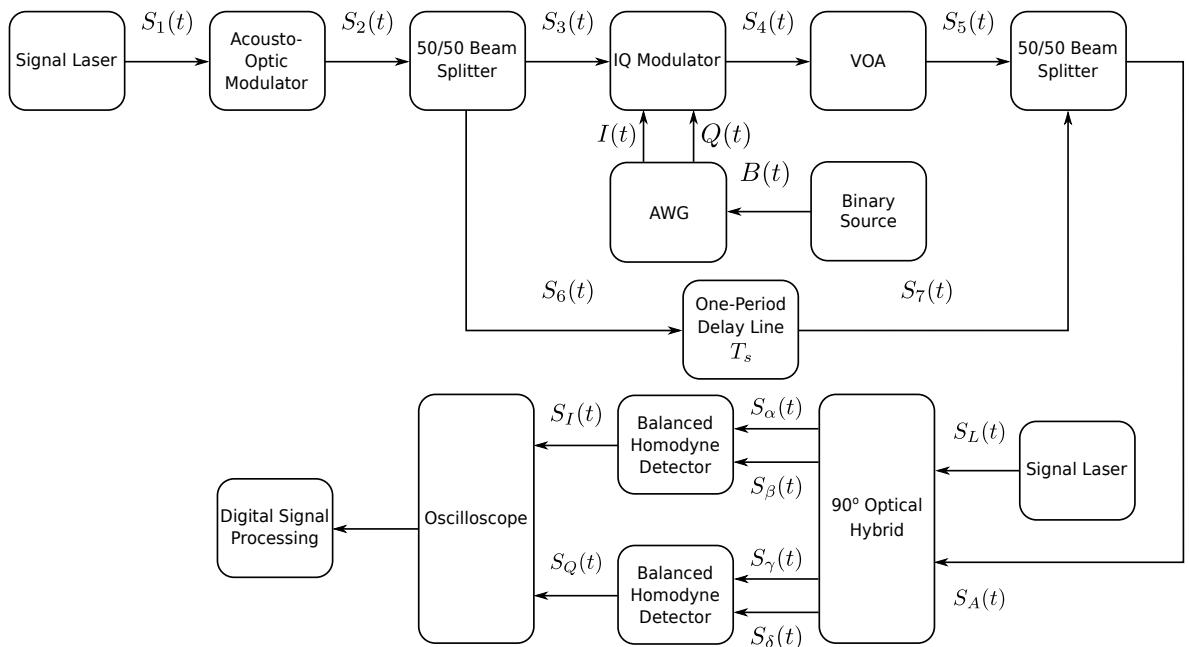


Figure 6.173: Diagram of the studied setup.

### Ideal laser without phase noise and frequency fluctuations

This first theoretical development assumes the usage of ideal lasers without phase noise nor frequency fluctuations. The signal generated by the laser is described by

$$S_1(t) = S_1 e^{i\phi}, \quad (6.158)$$

where  $\phi$  is the initial phase of the laser, assumed to take the value of 0, and  $S_1$  is its

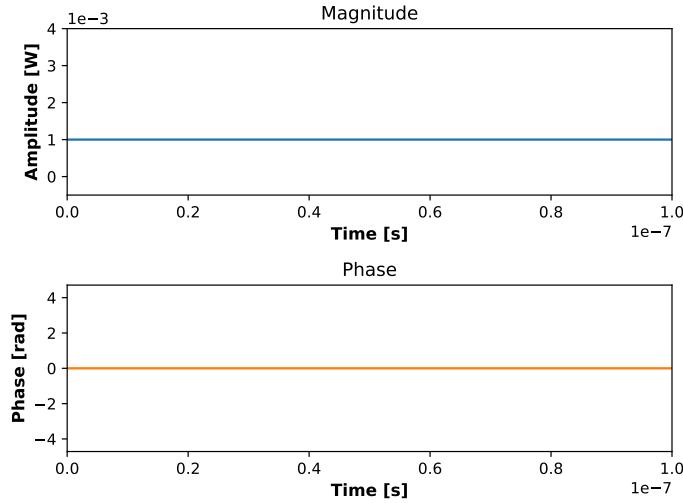


Figure 6.174: Magnitude and phase time dependence of  $S_1(t)$ .

amplitude, assumed to take the value  $\sqrt{10^{-3}} \sqrt{W}$ . This signal is phase and amplitude modulated by an IQ-Modulator with RZ signals given by

$$I(t) = \sum_{n=-\infty}^{n=+\infty} V_I(n)p(t) \quad , \quad Q(t) = \sum_{n=-\infty}^{n=+\infty} V_Q(n)p(t) \quad , \quad \text{with } p(t) = \begin{cases} 1, & 0 \leq t \leq T_h \\ 0, & \text{otherwise} \end{cases}, \quad T_h = \frac{T_s}{2} \quad (6.159)$$

where  $T_s$  is the symbol period,  $T_h$  is the high time and  $V_I(n)/V_Q(n)$  are the applied voltage amplitude for the  $n$ 'th pulse in the in-phase/in-quadrature RF electrode of the IQ-Modulator. We assume  $T_s = 10$  ns,  $T_h = 5$  ns, while the applied voltages take the values

$$V_I(n) = \begin{cases} V_{\frac{\pi}{2}}, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -13, -7, -5, 1, 3, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -3, -1, 5, 7, 13, \dots \end{cases}, \quad V_Q(n) = \begin{cases} 0, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -11, -7, -3, 1, 5, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -5, -1, 3, 7, 11, \dots \end{cases} \quad (6.160)$$

The applied encoding corresponds to a bit sequence taking the values 01001110 sequentially, following the state to bit correspondence described in Figure 6.172, interspersed with reference pulses.

$$S_2(t) = S_1 e^{i\phi} \left[ \sin \left( \frac{\pi I(t)}{V_{\pi}} \right) + i \sin \left( \frac{\pi Q(t)}{V_{\pi}} \right) \right]. \quad (6.161)$$

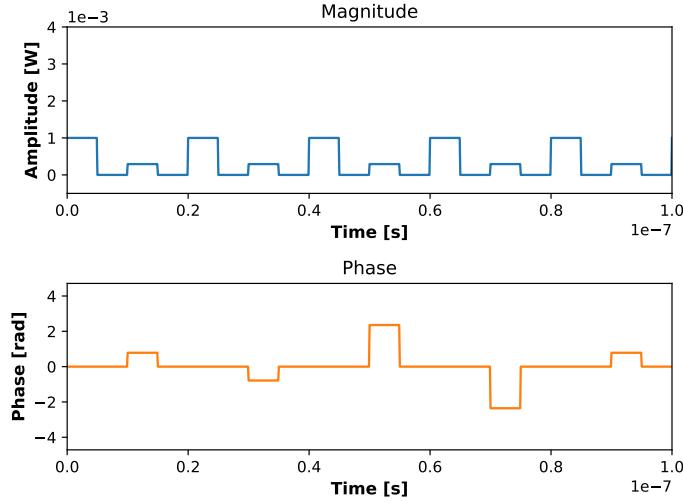
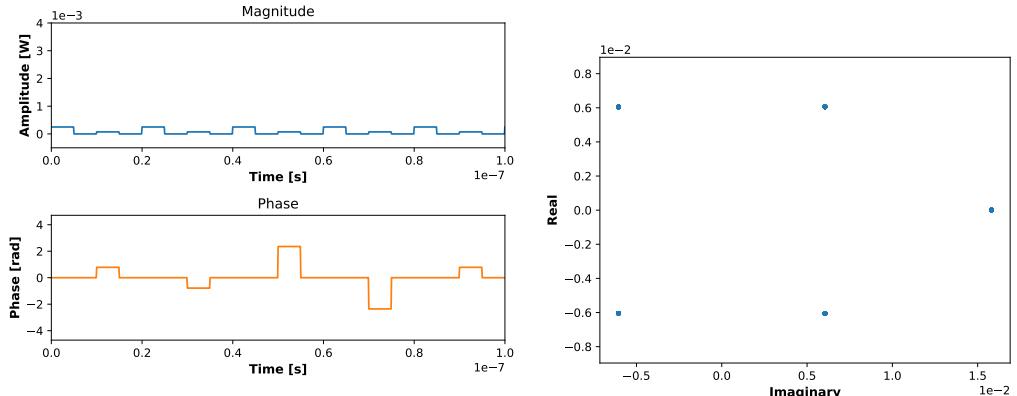


Figure 6.175: Magnitude and phase time dependence of  $S_2(t)$ .

This phase modulated signal is then attenuated by a factor of  $a$ , resulting in

$$S_A(t) = aS_1 e^{i\phi} \left[ \sin\left(\frac{\pi I(t)}{V_\pi}\right) + i \sin\left(\frac{\pi Q(t)}{V_\pi}\right) \right]. \quad (6.162)$$

Signal  $S_A(t)$  is then mixed in a  $90^\circ$  Optical Hybrid with signal  $S_L(t)$ , the latter described by



(a) Magnitude and phase time dependence of  $S_A(t)$ .

(b) Signal constellation of  $S_A(t)$ .

$$S_L(t) = S_L e^{i\varphi}, \quad (6.163)$$

where the initial phase of  $S_L(t)$ ,  $\varphi$ , differs from the initial phase of  $S_A(t)$  because of the different optical sources utilized and its amplitude,  $S_L$ , is assumed to take the value  $\sqrt{10^{-2}} \sqrt{W}$ . The signals resulting of this mixing, presented in (6.223), are then fed into two

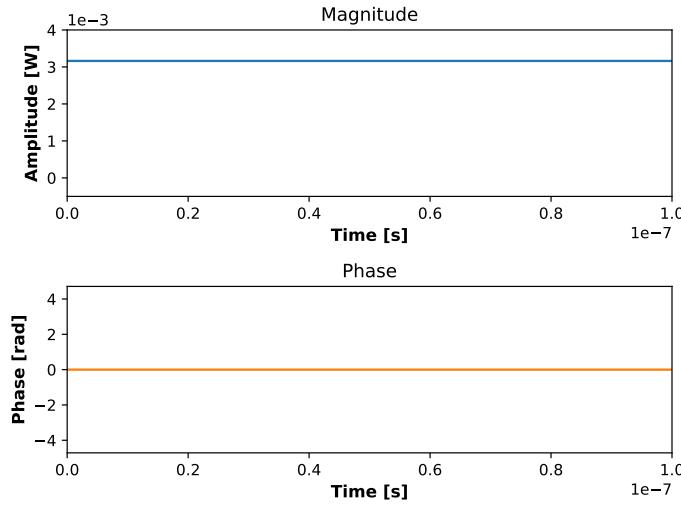


Figure 6.177: Magnitude and phase time dependence of  $S_L(t)$ .

Balanced Homodyne Receivers.

$$\begin{bmatrix} S_\alpha \\ S_\beta \\ S_\gamma \\ S_\delta \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} S_A(t) \\ S_L(t) \end{bmatrix}. \quad (6.164)$$

The internal scheme of the upper balanced homodyne receiver in Figure 6.173 is presented

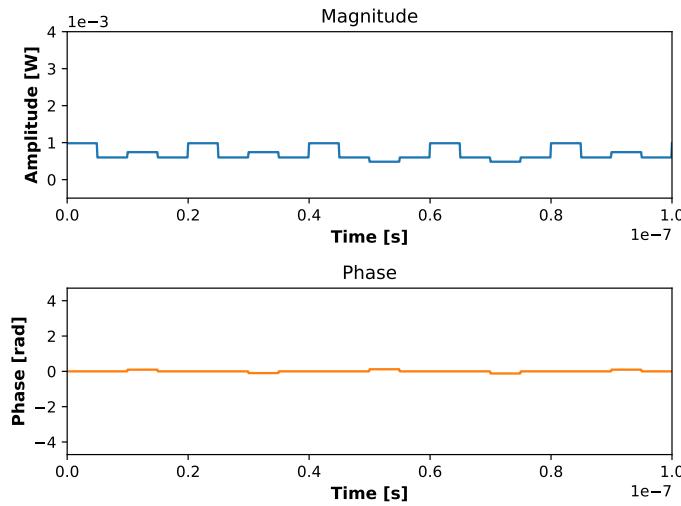


Figure 6.178: Magnitude and phase time dependence of  $S_\alpha(t)$ .

in Figure 6.182. The internal signal  $S_-(t)$  corresponds to the following subtraction current

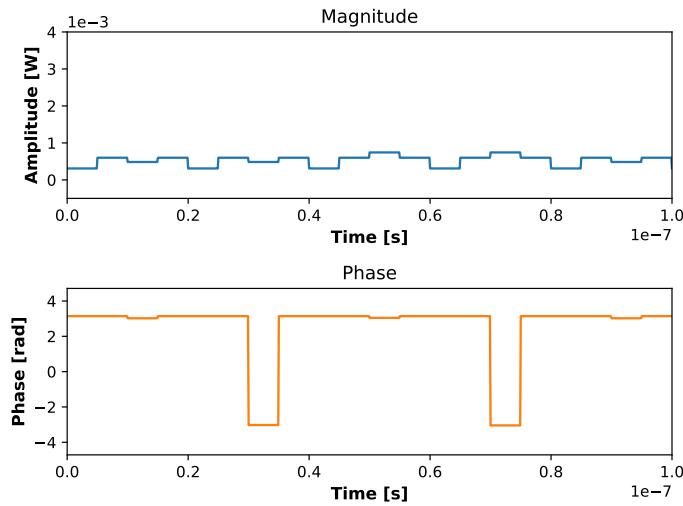


Figure 6.179: Magnitude and phase time dependence of  $S_\beta(t)$ .

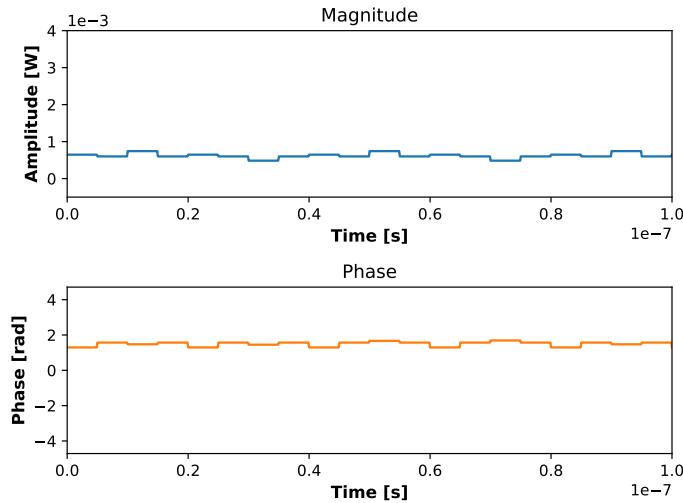


Figure 6.180: Magnitude and phase time dependence of  $S_\gamma(t)$ .

$$\begin{aligned}
 S_-(t) &= \rho (|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\
 &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \phi - \varphi + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \phi - \varphi - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi - \varphi + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \phi - \varphi - \frac{\pi Q(t)}{V_\pi} \right) \right], \tag{6.165}
 \end{aligned}$$

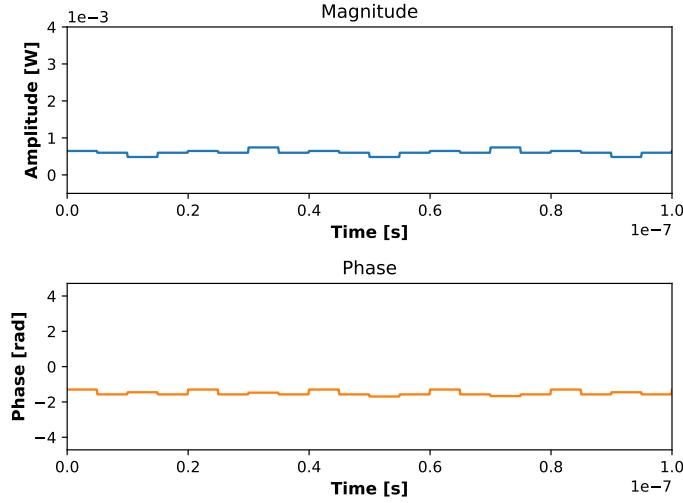


Figure 6.181: Magnitude and phase time dependence of  $S_\delta(t)$ .

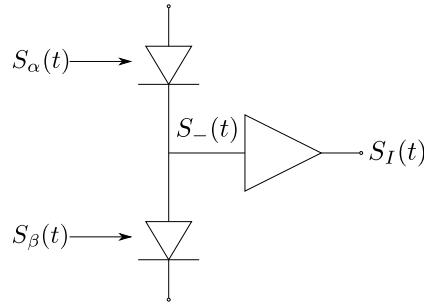


Figure 6.182: Internal diagram of the first homodyne receiver.

where  $\rho$  is the responsivity of the photodiodes. This signal is then amplified by a gain factor  $g$ , resulting in the signal

$$\begin{aligned} S_I(t) &= gS_-(t) = g\rho(|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\ &= \rho \frac{aS_1S_L}{2} \left[ \sin\left(\phi - \varphi + \frac{\pi I(t)}{V_\pi}\right) - \sin\left(\phi - \varphi - \frac{\pi I(t)}{V_\pi}\right) + \right. \\ &\quad \left. \cos\left(\phi - \varphi + \frac{\pi Q(t)}{V_\pi}\right) - \cos\left(\phi - \varphi - \frac{\pi Q(t)}{V_\pi}\right) \right]. \end{aligned} \quad (6.166)$$

The lower balanced homodyne receiver in Figure 6.173, presented in Figure 6.184, functions in a similar way to the upper homodyne receiver, with the subtraction current now given by

$$\begin{aligned} S_-(t) &= \rho(|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\ &= \rho \frac{aS_1S_L}{2} \left[ \sin\left(\phi - \varphi + \frac{\pi Q(t)}{V_\pi}\right) - \sin\left(\phi - \varphi - \frac{\pi Q(t)}{V_\pi}\right) + \right. \\ &\quad \left. \cos\left(\phi - \varphi - \frac{\pi I(t)}{V_\pi}\right) - \cos\left(\phi - \varphi + \frac{\pi I(t)}{V_\pi}\right) \right], \end{aligned} \quad (6.167)$$

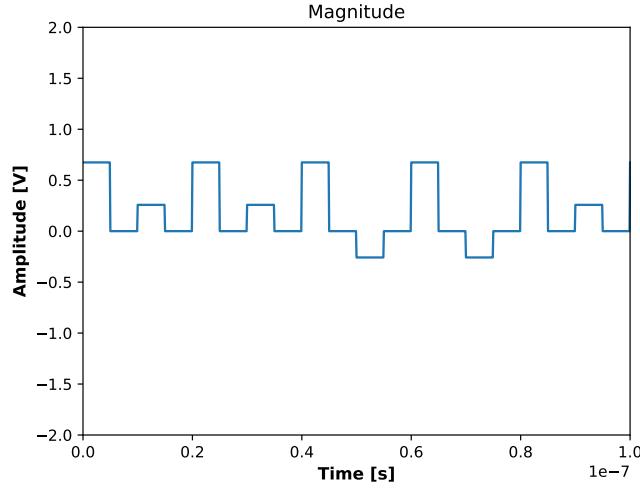
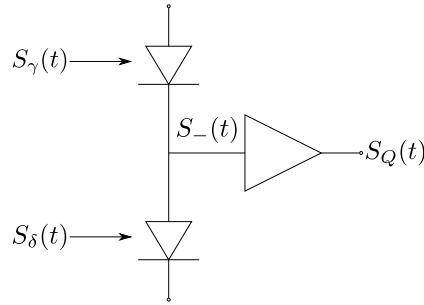
Figure 6.183: Amplitude time dependence of  $S_I(t)$ .

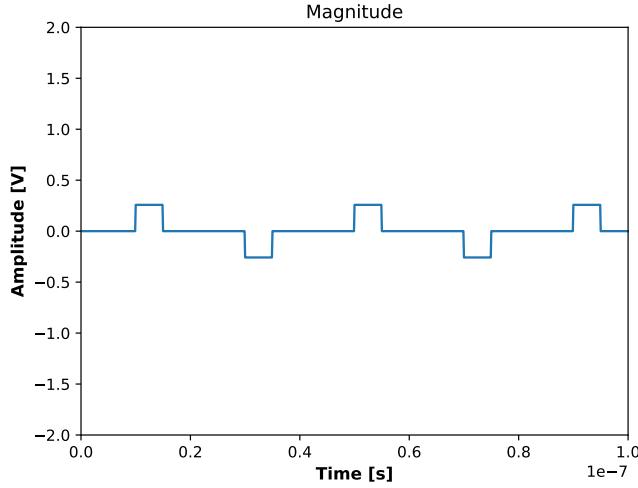
Figure 6.184: Internal diagram of the second homodyne receiver.

which is also amplified by a gain factor  $g$  and results in the signal

$$\begin{aligned}
 S_Q(t) &= g\rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \phi - \varphi + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \phi - \varphi - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi - \varphi - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \phi - \varphi + \frac{\pi I(t)}{V_\pi} \right) \right]. \tag{6.168}
 \end{aligned}$$

The  $S_I(t)$  and  $S_Q(t)$  signals are then sampled by a sampling function  $\Delta(t)$ , defined as

$$\Delta(t) = \sum_{n=-\infty}^{n=+\infty} \delta \left( t - \left( n + \frac{1}{4} \right) T_s \right), \text{ with } \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & \text{other} \end{cases}, \tag{6.169}$$

Figure 6.185: Amplitude time dependence of  $S_Q(t)$ .

resulting in the signals

$$\begin{aligned}
 S_I(n) &= \Delta(t)S_I(t) = S_I\left(\left(n + \frac{1}{4}\right)T_s\right) = S_I(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin\left(\phi - \varphi + \frac{\pi I(t_n)}{V_\pi}\right) - \sin\left(\phi - \varphi - \frac{\pi I(t_n)}{V_\pi}\right) + \right. \\
 &\quad \left. \cos\left(\phi - \varphi + \frac{\pi Q(t_n)}{V_\pi}\right) - \cos\left(\phi - \varphi - \frac{\pi Q(t_n)}{V_\pi}\right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.170}$$

$$\begin{aligned}
 S_Q(n) &= \Delta(t)S_Q(t) = S_Q\left(\left(n + \frac{1}{4}\right)T_s\right) = S_Q(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin\left(\phi - \varphi + \frac{\pi Q(t_n)}{V_\pi}\right) - \sin\left(\phi - \varphi - \frac{\pi Q(t_n)}{V_\pi}\right) + \right. \\
 &\quad \left. \cos\left(\phi - \varphi - \frac{\pi I(t_n)}{V_\pi}\right) - \cos\left(\phi - \varphi + \frac{\pi I(t_n)}{V_\pi}\right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.171}$$

The reference constellation can be recovered by isolating all the even-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_R(m) = S_I(m) + iS_Q(m) = g\rho a S_1 S_L e^{i(\phi - \varphi)}, \quad m = 2n, \quad n \in \mathbb{Z}, \tag{6.172}$$

while the phase encoded constellation can be recovered by isolating the odd-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_S(l) = S_I(l) + iS_Q(l) = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\phi - \varphi + \frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\phi - \varphi + \frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\phi - \varphi + \frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\phi - \varphi + \frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}. \tag{6.173}$$

Both  $IQ_R(m)$  and  $IQ_S(l)$  are plotted in Figure 6.186. The blue points correspond to the  $IQ_S(l)$  signal with the 01001110 sequential encoding while the orange ones correspond to the  $IQ_R(m)$  signal.

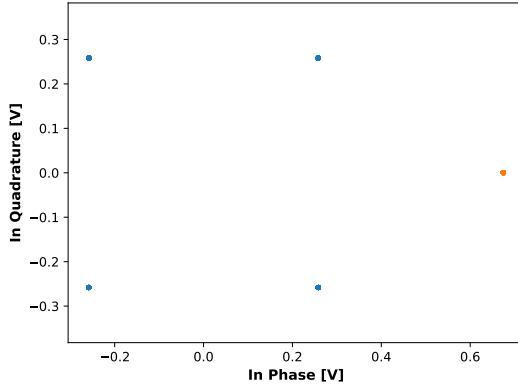


Figure 6.186: Constellation assuming  $\phi - \varphi = 0$ .

If there exists a phase mismatch between the modulated and local oscillator signal, as is the case in the constellation presented in Figure 6.187, the original constellation can be recovered by removing the phase of  $IQ_R(m)$  from  $IQ_S(l)$

$$IQ = IQ_S(l)e^{-i\arg(IQ_R(m))} \\ = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\phi - \varphi - \phi + \varphi + \frac{\pi}{4})}, & l = 8n + 1, m = l - 1 \\ e^{i(\phi - \varphi - \phi + \varphi + \frac{7\pi}{4})}, & l = 8n + 3, m = l - 1 \\ e^{i(\phi - \varphi - \phi + \varphi + \frac{3\pi}{4})}, & l = 8n + 5, m = l - 1 \\ e^{i(\phi - \varphi - \phi + \varphi + \frac{5\pi}{4})}, & l = 8n + 7, m = l - 1 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.174)$$

We thus show that the phase mismatch can be completely compensated with our scheme.

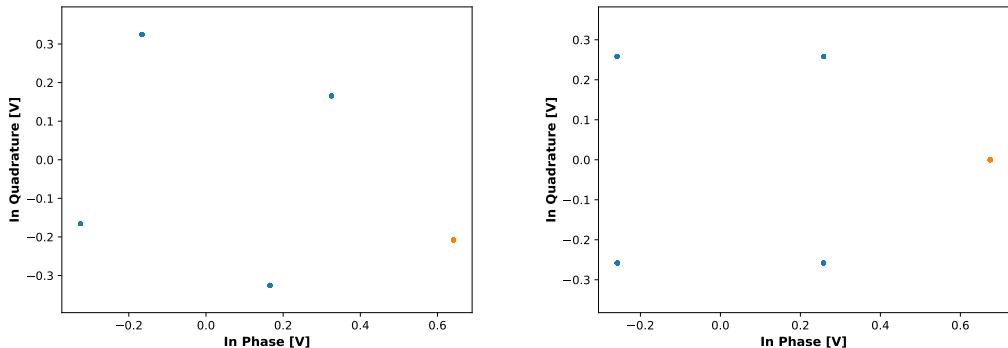


Figure 6.187: Constellation assuming a constant  $\phi - \varphi = \frac{\pi}{10}$  phase mismatch and corresponding recovered constellation.

After the phase originating from the initial phase difference between the  $S_L(t)$  and  $S_A(t)$  signals is removed, the constellation can be decoded in accordance to the state to bit correspondence in Figure 6.172.

### Laser with phase noise

This theoretical development assumes the usage of lasers with phase noise and without frequency mismatch. Laser phase noise can be modeled as a Weiner process described as

$$\phi(k) = \phi(k-1) + \Delta\phi(k), \quad (6.175)$$

where  $\Delta\phi(k)$  are the phase increments, which are independent and identically distributed random Gaussian variables with zero mean and variance given by

$$\sigma^2 = 2\pi\Delta f|t_1 - t_2|, \quad (6.176)$$

where  $\Delta f$  is the full-width at half maximum of the laser's frequency spectrum and  $t_{1/2}$  are two different sampling moments at which the phase is evaluated. The signal generated by the laser is described by

$$S_1(t) = S_1 e^{i\phi(t)}, \quad (6.177)$$

where  $\phi(t)$  is the instantaneous phase of the laser, assumed to follow the Wiener-Levy

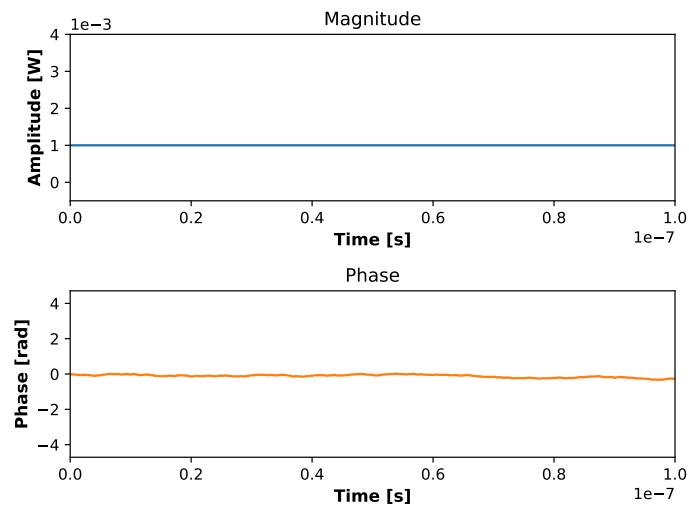


Figure 6.188: Magnitude and phase time dependence of  $S_1(t)$ .

process described above, and  $S_1$  is its amplitude, assumed to take the value  $10^{-3} \sqrt{W}$ . This signal is phase and amplitude modulated by an IQ-Modulator with RZ signals given by

$$I(t) = \sum_{n=-\infty}^{n=+\infty} V_I(n)p(t) \quad , \text{ with } p(t) = \begin{cases} 1, & 0 \leq t \leq T_h \\ 0, & \text{otherwise} \end{cases}, \quad T_h = \frac{T_s}{2} \quad (6.178)$$

$$Q(t) = \sum_{n=-\infty}^{n=+\infty} V_Q(n)p(t)$$

where  $T_s$  is the symbol period,  $T_h$  is the high time and  $V_I(n)/V_Q(n)$  are the applied voltage amplitude for the  $n$ 'th pulse in the in-phase/in-quadrature RF electrode of the IQ-Modulator. We assume  $T_s = 10$  ns,  $T_h = 5$  ns, while the applied voltages take the values

$$V_I(n) = \begin{cases} V_{\frac{\pi}{2}}, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -13, -7, -5, 1, 3, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -3, -1, 5, 7, 13, \dots \end{cases}, \quad V_Q(n) = \begin{cases} 0, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -11, -7, -3, 1, 5, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -5, -1, 3, 7, 11, \dots \end{cases} \quad . \quad (6.179)$$

The applied encoding corresponds to a bit sequence taking the values 01001110 sequentially, following the state to bit correspondence described in Figure 6.172, interspaced with reference pulses.

$$S_2(t) = S_1 e^{i\phi(t)} \left[ \sin \left( \frac{\pi I(t)}{V_\pi} \right) + i \sin \left( \frac{\pi Q(t)}{V_\pi} \right) \right]. \quad (6.180)$$

This phase modulated signal is then attenuated by a factor of  $a$ , resulting in

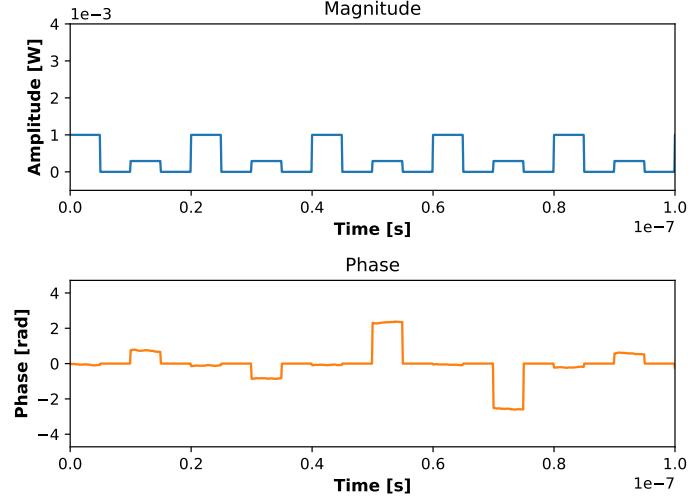


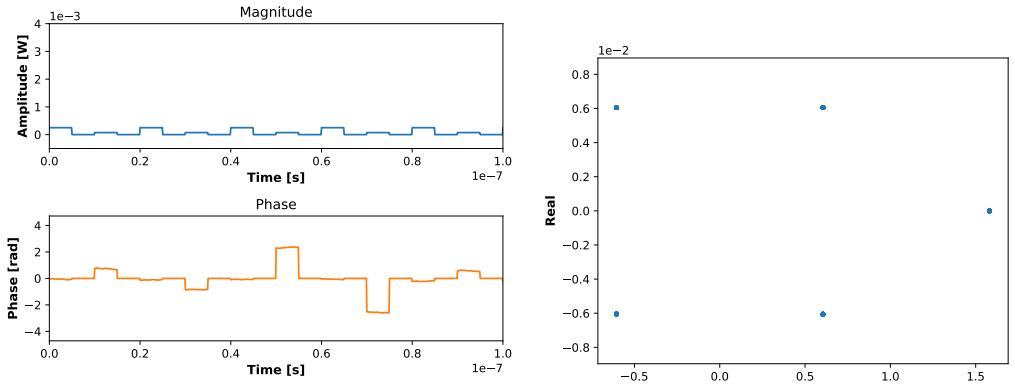
Figure 6.189: Magnitude and phase time dependence of  $S_2(t)$ .

$$S_A(t) = a S_1 e^{i\phi(t)} \left[ \sin \left( \frac{\pi I(t)}{V_\pi} \right) + i \sin \left( \frac{\pi Q(t)}{V_\pi} \right) \right]. \quad (6.181)$$

Signal  $S_A(t)$  is then mixed in a  $90^\circ$  Optical Hybrid with signal  $S_L(t)$ , the latter described by

$$S_L(t) = S_L e^{i\varphi(t)}, \quad (6.182)$$

where the initial phase of  $S_L(t)$ ,  $\varphi$ , follows a Wiener-Levy process independent from, but similar to, the one that governs the evolution of  $\phi(t)$ . The amplitude of the signal,  $S_L$ , is assumed to take the value  $\sqrt{10^{-2} W}$ . The signals resulting of this mixing, presented



(a) Magnitude and phase time dependence of  $S_A(t)$ .

(b) Signal constellation of  $S_A(t)$ .

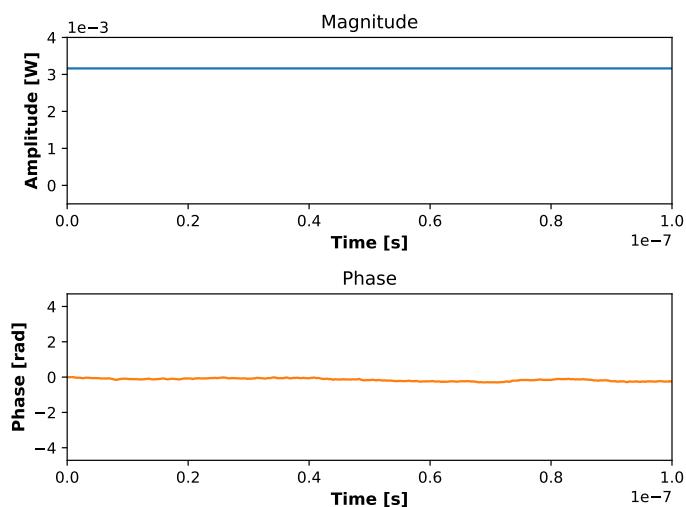


Figure 6.191: Magnitude and phase time dependence of  $S_L(t)$ .

in (6.223), are then fed into two Balanced Homodyne Receivers.

$$\begin{bmatrix} S_\alpha \\ S_\beta \\ S_\gamma \\ S_\delta \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} S_A(t) \\ S_L(t) \end{bmatrix}. \quad (6.183)$$

The internal scheme of the upper balanced homodyne receiver in Figure 6.173 is presented in Figure 6.182. The internal signal  $S_-(t)$  corresponds to the following subtraction current

$$\begin{aligned} S_-(t) &= \rho (|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\ &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) + \right. \\ &\quad \left. \cos \left( \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) \right], \end{aligned} \quad (6.184)$$

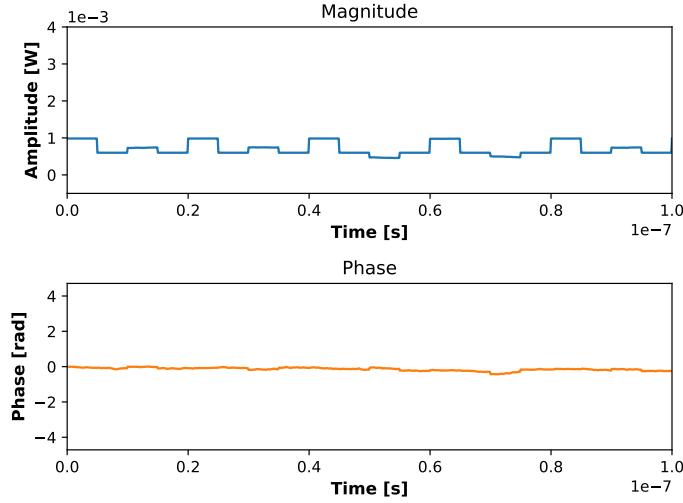


Figure 6.192: Magnitude and phase time dependence of  $S_\alpha(t)$ .

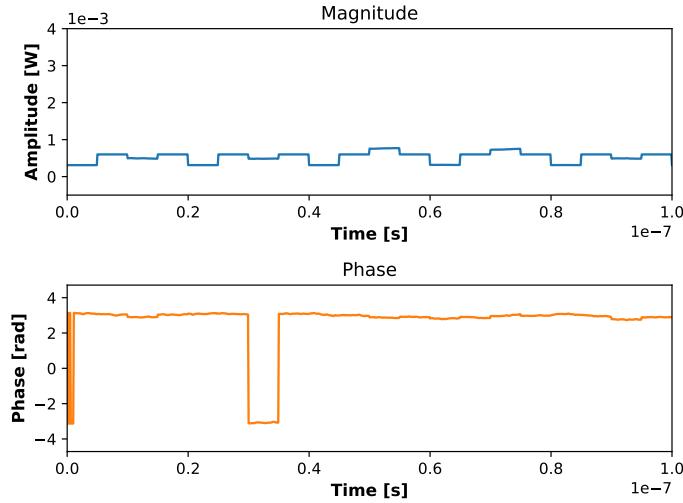


Figure 6.193: Magnitude and phase time dependence of  $S_\beta(t)$ .

where  $\rho$  is the responsivity of the photodiodes. This signal is then amplified by a gain factor  $g$ , resulting in the signal

$$\begin{aligned}
 S_I(t) &= gS_-(t) = g\rho \left( |S_\alpha(t)|^2 - |S_\beta(t)|^2 \right) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) \right]. \tag{6.185}
 \end{aligned}$$

The lower balanced homodyne receiver in Figure 6.173, presented in Figure 6.184, functions

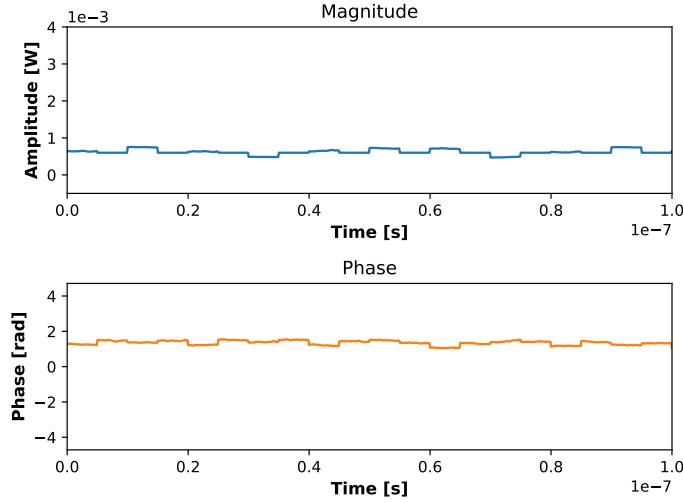


Figure 6.194: Magnitude and phase time dependence of  $S_\gamma(t)$ .

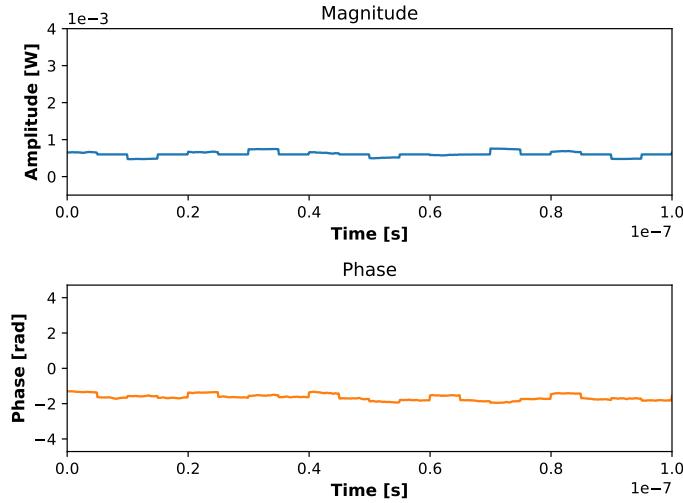


Figure 6.195: Magnitude and phase time dependence of  $S_\delta(t)$ .

in a similar way to the upper homodyne receiver, with the subtraction current now given by

$$\begin{aligned}
 S_-(t) &= \rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) \right], \quad (6.186)
 \end{aligned}$$

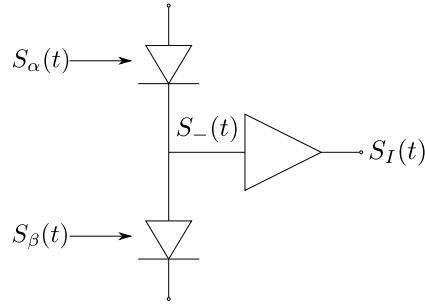
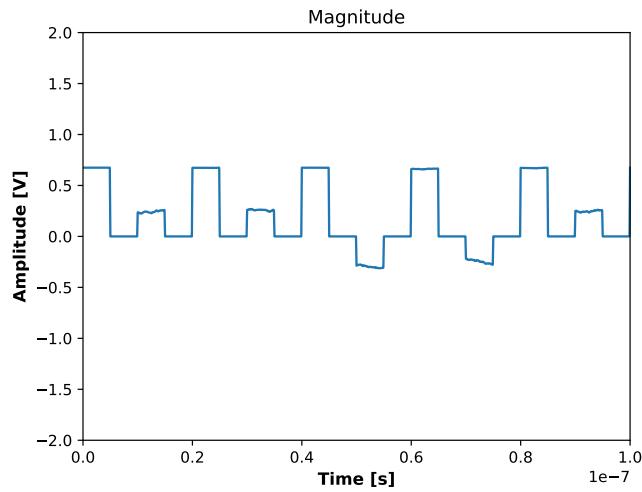


Figure 6.196: Internal diagram of the first homodyne receiver.

Figure 6.197: Amplitude time dependence of  $S_I(t)$ .

which is also amplified by a gain factor  $g$  and results in the signal

$$\begin{aligned}
 S_Q(t) &= g\rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) \right]. \quad (6.187)
 \end{aligned}$$

The  $S_I(t)$  and  $S_Q(t)$  signals are then sampled by a sampling function  $\Delta(t)$ , defined as

$$\Delta(t) = \sum_{n=-\infty}^{n=+\infty} \delta \left( t - \left( n + \frac{1}{4} \right) T_s \right), \text{ with } \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & \text{other} \end{cases}, \quad (6.188)$$

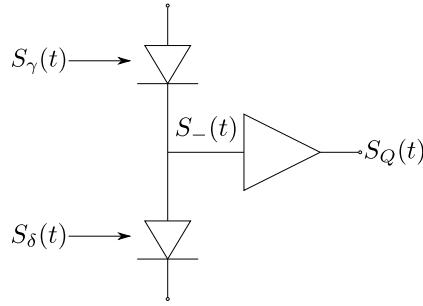
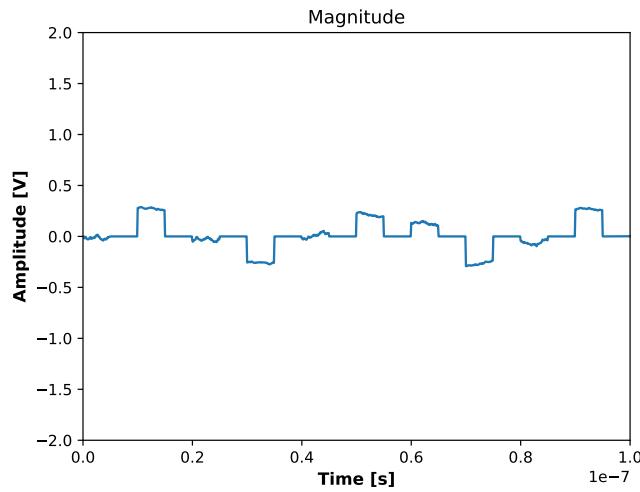


Figure 6.198: Internal diagram of the second homodyne receiver.

Figure 6.199: Amplitude time dependence of  $S_Q(t)$ .

resulting in the signals

$$\begin{aligned}
 S_I(n) &= \Delta(t)S_I(t) = S_I \left( \left( n + \frac{1}{4} \right) T_s \right) = S_I(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin \left( \phi(t_n) - \varphi(t_n) + \frac{\pi I(t_n)}{V_\pi} \right) - \sin \left( \phi(t_n) - \varphi(t_n) - \frac{\pi I(t_n)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi(t_n) - \varphi(t_n) + \frac{\pi Q(t_n)}{V_\pi} \right) - \cos \left( \phi(t_n) - \varphi(t_n) - \frac{\pi Q(t_n)}{V_\pi} \right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.189}$$

$$\begin{aligned}
 S_Q(n) &= \Delta(t)S_Q(t) = S_Q \left( \left( n + \frac{1}{4} \right) T_s \right) = S_Q(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin \left( \phi(t_n) - \varphi(t_n) + \frac{\pi Q(t_n)}{V_\pi} \right) - \sin \left( \phi(t_n) - \varphi(t_n) - \frac{\pi Q(t_n)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \phi(t_n) - \varphi(t_n) - \frac{\pi I(t_n)}{V_\pi} \right) - \cos \left( \phi(t_n) - \varphi(t_n) + \frac{\pi I(t_n)}{V_\pi} \right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.190}$$

The reference constellation can be recovered by isolating all the even-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_R(m) = S_I(m) + iS_Q(m) = g\rho a S_1 S_L e^{i(\phi(t_m) - \varphi(t_m))}, \quad m = 2n, \quad n \in \mathbb{Z}, \quad (6.191)$$

while the phase encoded constellation can be recovered by isolating the odd-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_S(l) = S_I(l) + iS_Q(l) = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\phi(t_l) - \varphi(t_l) + \frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\phi(t_l) - \varphi(t_l) + \frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\phi(t_l) - \varphi(t_l) + \frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\phi(t_l) - \varphi(t_l) + \frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.192)$$

Both  $IQ_R(m)$  and  $IQ_S(l)$  are plotted in Figure 6.200. The blue points correspond to the  $IQ_S(l)$  signal with the 01001110 sequential encoding while the orange ones correspond to the  $IQ_R(m)$  signal. The effect of the phase noise is visible in the dragging rotation of the 4 state constellation, which results in the circular constellation visible in Figure 6.200.

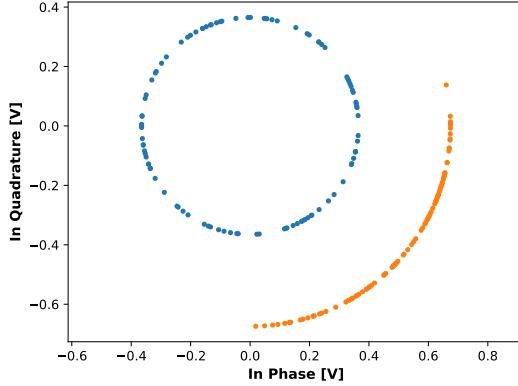


Figure 6.200: Recovered constellation assuming a large acquisition time.

The original constellation can be recovered by removing the phase of  $IQ_R(m)$  from  $IQ_S(l)$

$$IQ = IQ_S(l) e^{-i\arg(IQ_R(m))} \\ = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{\pi}{4})}, & l = 8n + 1, \quad m = l - 1 \\ e^{i(\phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{7\pi}{4})}, & l = 8n + 3, \quad m = l - 1 \\ e^{i(\phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{3\pi}{4})}, & l = 8n + 5, \quad m = l - 1 \\ e^{i(\phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{5\pi}{4})}, & l = 8n + 7, \quad m = l - 1 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.193)$$

The constellation obtained after this phase drift compensation scheme is presented in Figure 6.201. The remaining phase drift observed is due to the laser's phase drifting

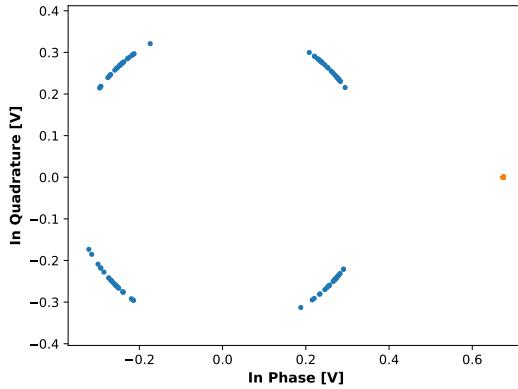


Figure 6.201: Recovered constellation after the phase drift compensation step for a symbol period of  $T_s = 1 \mu\text{s}$ .

during the timing difference between the signal and reference pulses, thus the quality of the phase drift compensation scheme improves with reduced symbol periods. From the

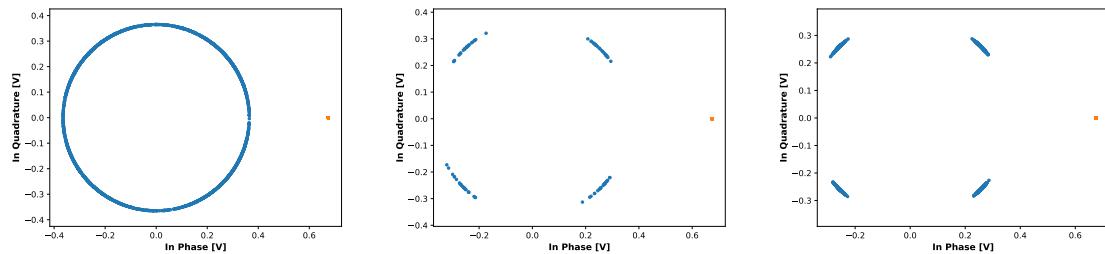


Figure 6.202: Recovered constellations after the phase drift compensation step for symbol periods of 100, 10 and 1 ns.

results in Figure 6.202 we see that the remaining phase noise can be reduced by increasing the repetition rate. The variance of the remaining phase drift is expected to take the value

$$\sigma^2 = 2\pi\Delta f T_s, \quad (6.194)$$

to test this we isolate each state and calculate their phase variance, the results thus obtained are plotted against the expected values, calculated with (6.194), in Figure 6.203. We thus show that the phase noise can be sufficiently compensated with our scheme. After the phase originating from the initial phase difference between the  $S_L(t)$  and  $S_A(t)$  signals is removed, the constellation can be decoded in accordance to the state to bit correspondence in Figure 6.172.

### Laser with frequency mismatch

This theoretical development assumes the usage of lasers with a constant frequency mismatch and without phase noise. Laser frequency mismatch arises normally from the

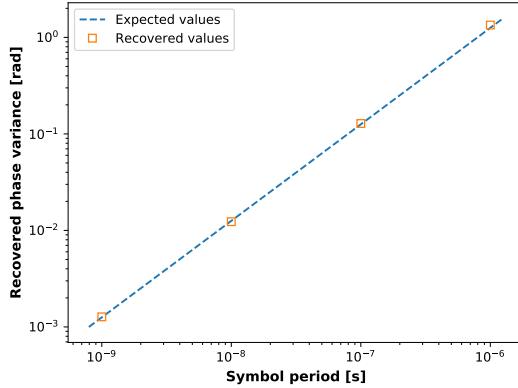


Figure 6.203: Variance of the remaining phase noise.

usage of different laser sources. The signal generated by the laser is described by

$$S_1(t) = S_1 e^{i\phi}, \quad (6.195)$$

where  $\phi(t)$  is the instantaneous phase of the laser, assumed to follow the Wiener-Levy

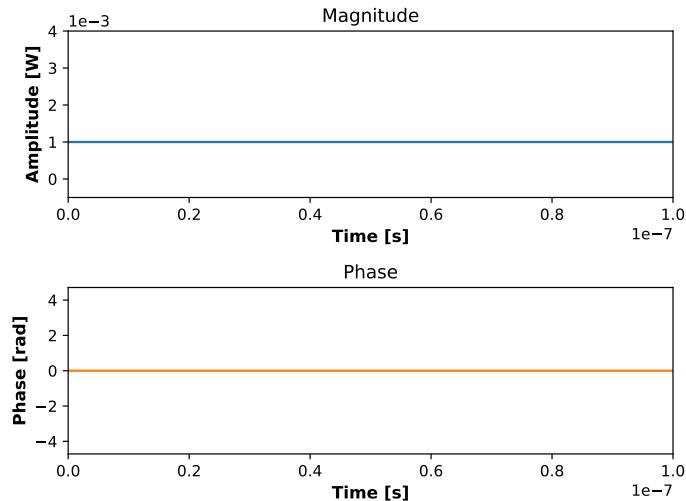


Figure 6.204: Magnitude and phase time dependence of  $S_1(t)$ .

process described above, and  $S_1$  is its amplitude, assumed to take the value  $10^{-3} \sqrt{W}$ . This signal is phase and amplitude modulated by an IQ-Modulator with RZ signals given by

$$I(t) = \sum_{n=-\infty}^{n=+\infty} V_I(n)p(t) \quad , \quad Q(t) = \sum_{n=-\infty}^{n=+\infty} V_Q(n)p(t) \quad , \quad \text{with } p(t) = \begin{cases} 1, & 0 \leq t \leq T_h \\ 0, & \text{otherwise} \end{cases} \quad , \quad T_h = \frac{T_s}{2} \quad (6.196)$$

where  $T_s$  is the symbol period,  $T_h$  is the high time and  $V_I(n)/V_Q(n)$  are the applied voltage amplitude for the n'th pulse in the in-phase/in-quadrature RF electrode of the IQ-

Modulator. We assume  $T_s = 10$  ns,  $T_h = 5$  ns, while the applied voltages take the values

$$V_I(n) = \begin{cases} V_{\frac{\pi}{2}}, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -13, -7, -5, 1, 3, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -3, -1, 5, 7, 13, \dots \end{cases}, \quad V_Q(n) = \begin{cases} 0, & n \text{ is even} \\ \frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -11, -7, -3, 1, 5, 9, \dots \\ -\frac{V_{\frac{\pi}{2}}}{k}, & n = \dots, -9, -5, -1, 3, 7, 11, \dots \end{cases}. \quad (6.197)$$

The applied encoding corresponds to a bit sequence taking the values 01001110 sequentially, following the state to bit correspondence described in Figure 6.172, interspaced with reference pulses.

$$S_2(t) = S_1 e^{i\phi} \left[ \sin \left( \frac{\pi I(t)}{V_\pi} \right) + i \sin \left( \frac{\pi Q(t)}{V_\pi} \right) \right]. \quad (6.198)$$

This phase modulated signal is then attenuated by a factor of  $a$ , resulting in

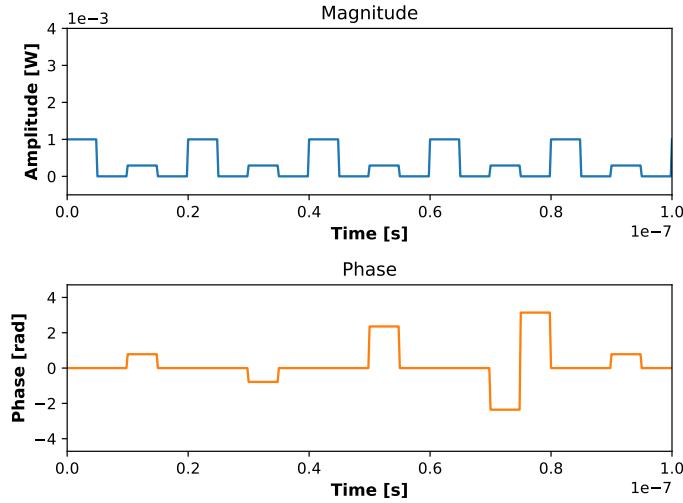


Figure 6.205: Magnitude and phase time dependence of  $S_2(t)$ .

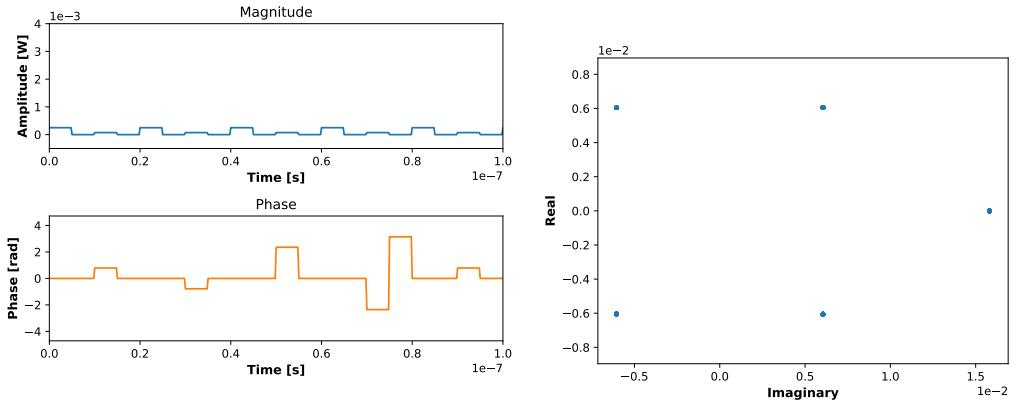
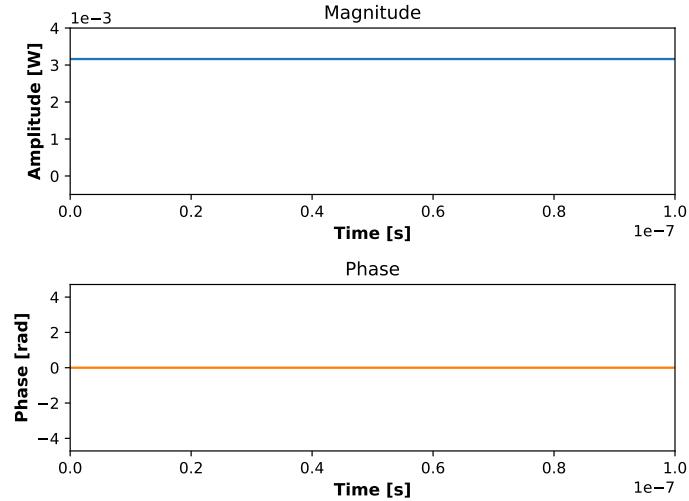
$$S_A(t) = a S_1 e^{i\phi} \left[ \sin \left( \frac{\pi I(t)}{V_\pi} \right) + i \sin \left( \frac{\pi Q(t)}{V_\pi} \right) \right]. \quad (6.199)$$

Signal  $S_A(t)$  is then mixed in a  $90^\circ$  Optical Hybrid with signal  $S_L(t)$ , the latter described by

$$S_L(t) = S_L e^{i\varphi}, \quad (6.200)$$

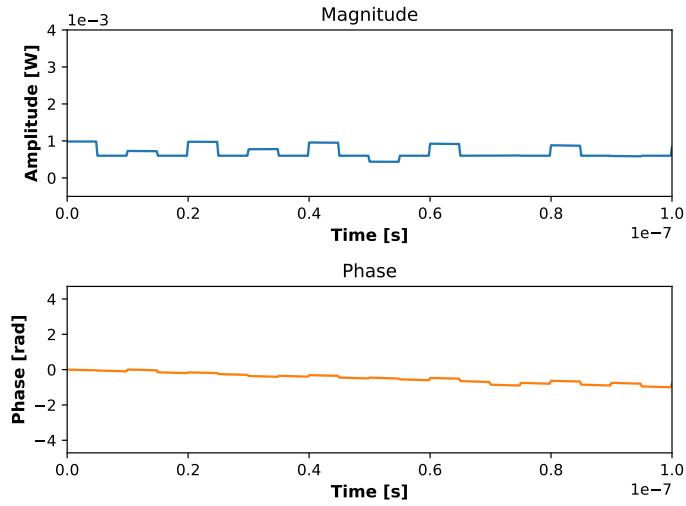
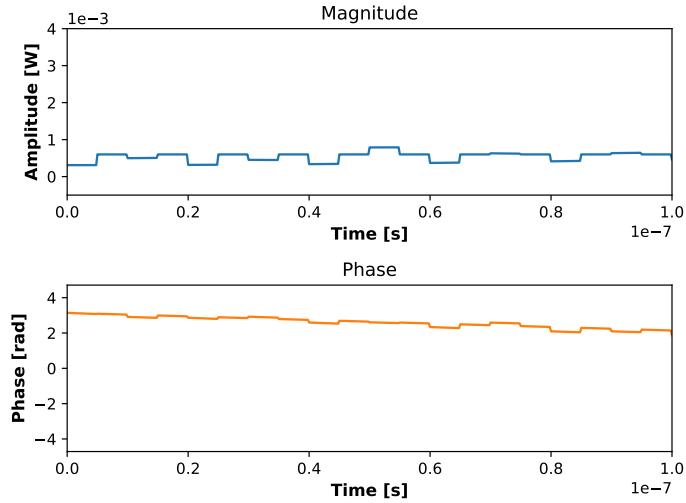
The amplitude of the signal,  $S_L$ , is assumed to take the value  $\sqrt{10^{-2}} \sqrt{W}$ . The signals resulting of this mixing, presented in (6.223), are then fed into two Balanced Homodyne Receivers. At this point the frequency mismatch becomes noticeable, due to the introduction of a beat frequency. In this study the beat frequency was assumed to take the value  $\omega_B = 10$  MHz.

$$\begin{bmatrix} S_\alpha \\ S_\beta \\ S_\gamma \\ S_\delta \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} S_A(t) \\ S_L(t) \end{bmatrix}. \quad (6.201)$$

(a) Magnitude and phase time dependence of  $S_A(t)$ .(b) Signal constellation of  $S_A(t)$ .Figure 6.207: Magnitude and phase time dependence of  $S_L(t)$ .

The internal scheme of the upper balanced homodyne receiver in Figure 6.173 is presented in Figure 6.182. The internal signal  $S_-(t)$  corresponds to the following subtraction current

$$\begin{aligned}
 S_-(t) &= \rho (|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\
 &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \omega_B t + \phi - \varphi + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi - \varphi - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi - \varphi + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi - \varphi - \frac{\pi Q(t)}{V_\pi} \right) \right], \quad (6.202)
 \end{aligned}$$

Figure 6.208: Magnitude and phase time dependence of  $S_\alpha(t)$ .Figure 6.209: Magnitude and phase time dependence of  $S_\beta(t)$ .

where  $\rho$  is the responsivity of the photodiodes. This signal is then amplified by a gain factor  $g$ , resulting in the signal

$$\begin{aligned}
 S_I(t) &= gS_-(t) = g\rho (|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t + \phi - \varphi + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi - \varphi - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi - \varphi + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi - \varphi - \frac{\pi Q(t)}{V_\pi} \right) \right]. \quad (6.203)
 \end{aligned}$$

The lower balanced homodyne receiver in Figure 6.173, presented in Figure 6.184, functions

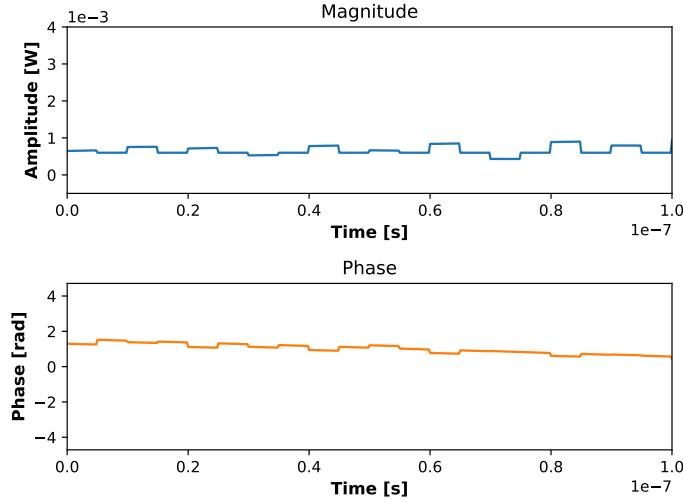


Figure 6.210: Magnitude and phase time dependence of  $S_\gamma(t)$ .

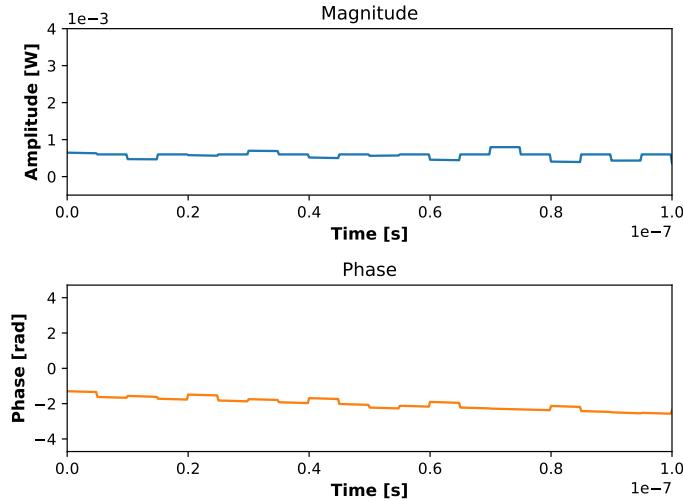


Figure 6.211: Magnitude and phase time dependence of  $S_\delta(t)$ .

in a similar way to the upper homodyne receiver, with the subtraction current now given by

$$\begin{aligned}
 S_-(t) &= \rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \omega_B t + \phi - \varphi + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi - \varphi - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi - \varphi - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi - \varphi + \frac{\pi I(t)}{V_\pi} \right) \right], \quad (6.204)
 \end{aligned}$$

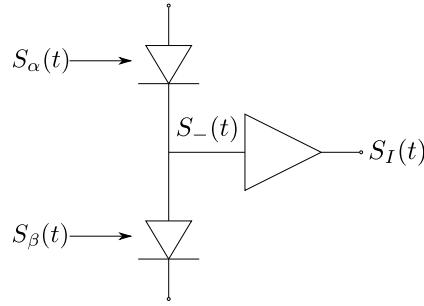
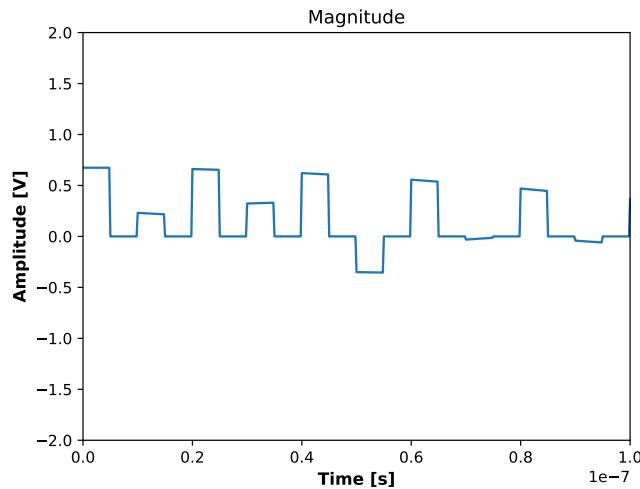


Figure 6.212: Internal diagram of the first homodyne receiver.

Figure 6.213: Amplitude time dependence of  $S_I(t)$ .

which is also amplified by a gain factor  $g$  and results in the signal

$$\begin{aligned}
 S_Q(t) &= g\rho(|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin\left(\omega_B t + \phi - \varphi + \frac{\pi Q(t)}{V_\pi}\right) - \sin\left(\omega_B t + \phi - \varphi - \frac{\pi Q(t)}{V_\pi}\right) + \right. \\
 &\quad \left. \cos\left(\omega_B t + \phi - \varphi - \frac{\pi I(t)}{V_\pi}\right) - \cos\left(\omega_B t + \phi - \varphi + \frac{\pi I(t)}{V_\pi}\right) \right]. \quad (6.205)
 \end{aligned}$$

The  $S_I(t)$  and  $S_Q(t)$  signals are then sampled by a sampling function  $\Delta(t)$ , defined as

$$\Delta(t) = \sum_{n=-\infty}^{n=+\infty} \delta\left(t - \left(n + \frac{1}{4}\right)T_s\right), \text{ with } \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & \text{otherwise} \end{cases}, \quad (6.206)$$

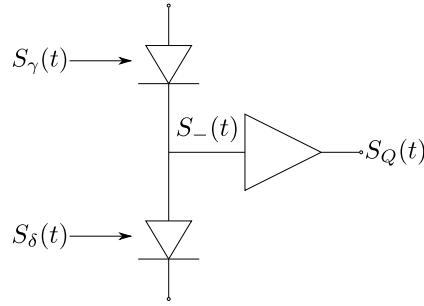
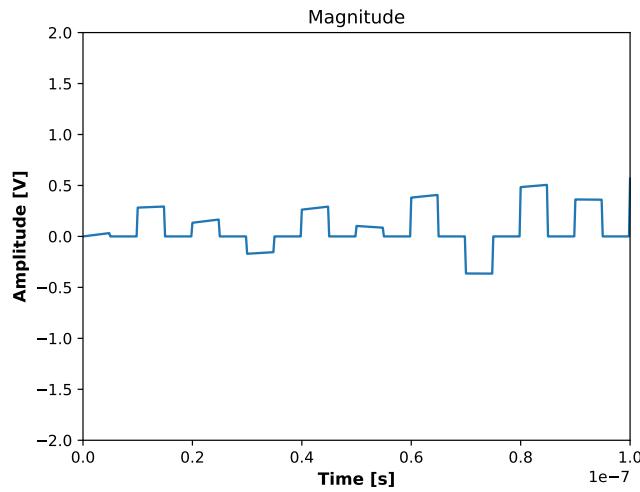


Figure 6.214: Internal diagram of the second homodyne receiver.

Figure 6.215: Amplitude time dependence of  $S_Q(t)$ .

resulting in the signals

$$\begin{aligned}
 S_I(n) &= \Delta(t)S_I(t) = S_I \left( \left( n + \frac{1}{4} \right) T_s \right) = S_I(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t_n + \phi - \varphi + \frac{\pi I(t_n)}{V_\pi} \right) - \sin \left( \omega_B t_n + \phi - \varphi - \frac{\pi I(t_n)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t_n + \phi - \varphi + \frac{\pi Q(t_n)}{V_\pi} \right) - \cos \left( \omega_B t_n + \phi - \varphi - \frac{\pi Q(t_n)}{V_\pi} \right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.207}$$

$$\begin{aligned}
 S_Q(n) &= \Delta(t)S_Q(t) = S_Q \left( \left( n + \frac{1}{4} \right) T_s \right) = S_Q(t_n) \\
 &= g\rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t_n + \phi - \varphi + \frac{\pi Q(t_n)}{V_\pi} \right) - \sin \left( \omega_B t_n + \phi - \varphi - \frac{\pi Q(t_n)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t_n + \phi - \varphi - \frac{\pi I(t_n)}{V_\pi} \right) - \cos \left( \omega_B t_n + \phi - \varphi + \frac{\pi I(t_n)}{V_\pi} \right) \right], \quad n \in \mathbb{Z}
 \end{aligned} \tag{6.208}$$

The reference constellation can be recovered by isolating all the even-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_R(m) = S_I(m) + iS_Q(m) = g\rho a S_1 S_L e^{i(\omega_B t_m + \phi - \varphi)}, \quad m = 2n, \quad n \in \mathbb{Z}, \quad (6.209)$$

while the phase encoded constellation can be recovered by isolating the odd-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_S(l) = S_I(l) + iS_Q(l) = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\omega_B t_l + \phi - \varphi + \frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\omega_B t_l + \phi - \varphi + \frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\omega_B t_l + \phi - \varphi + \frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\omega_B t_l + \phi - \varphi + \frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.210)$$

Both  $IQ_R(m)$  and  $IQ_S(l)$  are plotted in Figure 6.216. The blue points correspond to the  $IQ_S(l)$  signal with the 01001110 sequential encoding while the orange ones correspond to the  $IQ_R(m)$  signal. The effect of the frequency mismatch is visible in the dragging rotation of the 4 state constellation, which results in the circular constellation visible in Figure 6.216. The  $\omega_B t_m$  and  $\omega_B t_l$  elements in the equations for the reference and signal constellations are

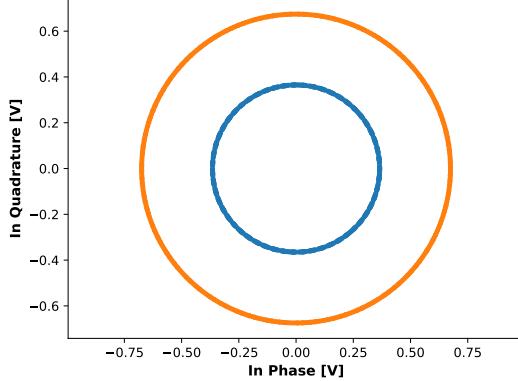


Figure 6.216: Recovered constellation assuming a frequency mismatch.

responsible for the dragging rotation visible in Figure 6.216. This effect can be removed by estimating the value of  $\omega_B$  and multiplying the reference and signal constellations by  $e^{-i\omega_B t_j}$  and  $e^{-i\omega_B(t_j + T_s)}$ , where  $t_j$  attempts to estimate the values of  $t_m$ . The beat frequency  $\omega_B$  can be determined by multiplying an incremented reference constellation by the complex conjugate of the unincremented constellation, resulting in

$$IQ_{\text{beat}} = IQ_R(m+2)IQ_R^*(m) = (g\rho a S_1 S_L)^2 e^{i\omega_B 2T_s} \quad (6.211)$$

and taking the phase of the resulting signal. The beat frequency compensated constellations are then given by

$$IQ_r(m) = IQ_R(m)e^{-i\omega_B t_j} = g\rho a S_1 S_L e^{i(\omega_B(t_m - t_j) + \phi - \varphi)}, \quad m = 2n, \quad n \in \mathbb{Z}, \quad (6.212)$$

and

$$IQ_s(l) = IQ_s(l)e^{-i\omega_B(t_j+T_s)} = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\omega_B(t_l-t_j-T_s)+\phi-\varphi+\frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\omega_B(t_l-t_j-T_s)+\phi-\varphi+\frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\omega_B(t_l-t_j-T_s)+\phi-\varphi+\frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\omega_B(t_l-t_j-T_s)+\phi-\varphi+\frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.213)$$

The constellations for  $IQ_r(m)$  and  $IQ_s(l)$  are presented in Figure 6.217, where it was assumed that  $t_j$  misestimates  $t_m$  by  $10T_s$ . Taking into account that  $t_l = t_m - T_s$ , the

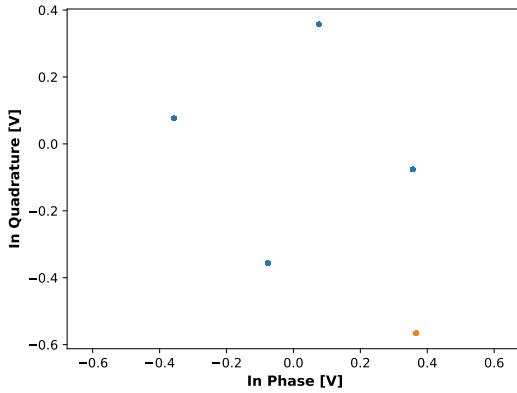


Figure 6.217: Recovered constellation after the phase drift compensation step for a symbol period of  $T_s = 10$  ns assuming  $t_j$  misestimates  $t_m$  by  $10T_s$ .

original constellation can be easily recovered by removing the phase of  $IQ_r(m)$  from  $IQ_s(l)$ , compensating both the initial phase difference and the one introduced by the misestimation of  $t_m$ .

$$IQ = IQ_s(l)e^{-i\arg(IQ_r(m))} = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i\frac{\pi}{4}}, & l = 8n + 1, m = l - 1 \\ e^{i\frac{7\pi}{4}}, & l = 8n + 3, m = l - 1 \\ e^{i\frac{3\pi}{4}}, & l = 8n + 5, m = l - 1 \\ e^{i\frac{5\pi}{4}}, & l = 8n + 7, m = l - 1 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.214)$$

The constellation obtained after both the frequency and phase difference compensation scheme is presented in Figure 6.201. We thus show that the frequency mismatch can be completely compensated with our scheme. After the phase originating from the initial phase difference between the  $S_L(t)$  and  $S_A(t)$  signals is removed, the constellation can be decoded in accordance to the state to bit correspondence in Figure 6.172.

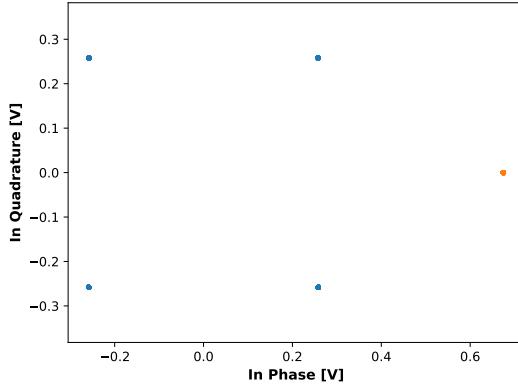


Figure 6.218: Recovered constellation after the phase drift compensation step for a symbol period of  $T_s = 10$  ns.

#### Realistic laser with phase difference, phase noise and frequency mismatch

This theoretical development assumes the usage of lasers with both phase noise and a constant frequency mismatch. As before, the laser phase noise can be modeled as a Weiner process described as

$$\phi(k) = \phi(k-1) + \Delta\phi(k), \quad (6.215)$$

where  $\Delta\phi(k)$  are the phase increments, which are independent and identically distributed random Gaussian variables with zero mean and variance given by

$$\sigma^2 = 2\pi\Delta f|t_1 - t_2|, \quad (6.216)$$

where  $\Delta f$  is the full-width at half maximum of the laser's frequency spectrum and  $t_{1/2}$  are two different sampling moments at which the phase is evaluated. The signal generated by the laser is described by

$$S_1(t) = S_1 e^{i\phi(t)}, \quad (6.217)$$

where  $\phi(t)$  is the instantaneous phase of the laser, assumed to follow the Wiener-Levy process described above, and  $S_1$  is its amplitude, assumed to take the value  $10^{-3} \sqrt{W}$ . This signal is phase and amplitude modulated by an IQ-Modulator with RZ signals given by

$$I(t) = \sum_{n=-\infty}^{n=+\infty} V_I(n)p(t) \quad Q(t) = \sum_{n=-\infty}^{n=+\infty} V_Q(n)p(t), \text{ with } p(t) = \begin{cases} 1, & 0 \leq t \leq T_h \\ 0, & \text{otherwise} \end{cases}, \quad T_h = \frac{T_s}{2} \quad (6.218)$$

where  $T_s$  is the symbol period,  $T_h$  is the high time and  $V_I(n)/V_Q(n)$  are the applied voltage amplitude for the  $n$ 'th pulse in the in-phase/in-quadrature RF electrode of the IQ-Modulator. We assume  $T_s = 10$  ns,  $T_h = 5$  ns, while the applied voltages take the values

$$V_I(n) = \begin{cases} \frac{V_\pi}{2}, & n \text{ is even} \\ \frac{V_\pi}{k}, & n = \dots, -13, -7, -5, 1, 3, 9, \dots \\ -\frac{V_\pi}{k}, & n = \dots, -9, -3, -1, 5, 7, 13, \dots \end{cases}, \quad V_Q(n) = \begin{cases} 0, & n \text{ is even} \\ \frac{V_\pi}{2}, & n = \dots, -11, -7, -3, 1, 5, 9, \dots \\ -\frac{V_\pi}{2}, & n = \dots, -9, -5, -1, 3, 7, 11, \dots \end{cases}. \quad (6.219)$$

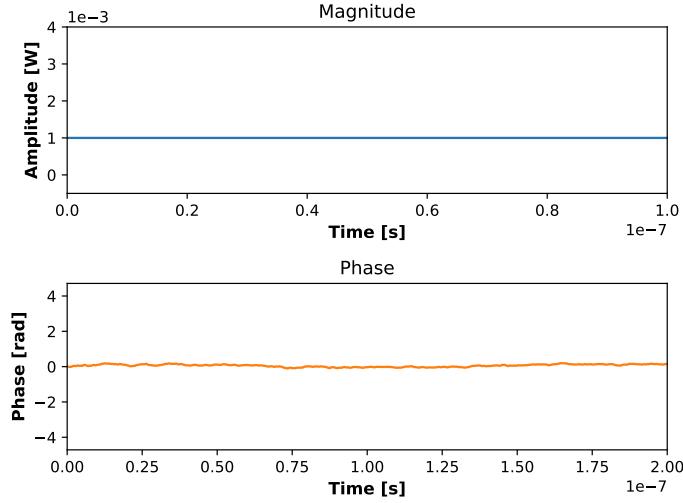


Figure 6.219: Magnitude and phase time dependence of  $S_1(t)$ .

The applied encoding corresponds to a bit sequence taking the values 01001110 sequentially, following the state to bit correspondence described in Figure 6.172, interspaced with reference pulses.

$$S_2(t) = S_1 e^{i\phi(t)} \left[ \sin\left(\frac{\pi I(t)}{V_\pi}\right) + i \sin\left(\frac{\pi Q(t)}{V_\pi}\right) \right]. \quad (6.220)$$

This phase modulated signal is then attenuated by a factor of  $a$ , resulting in

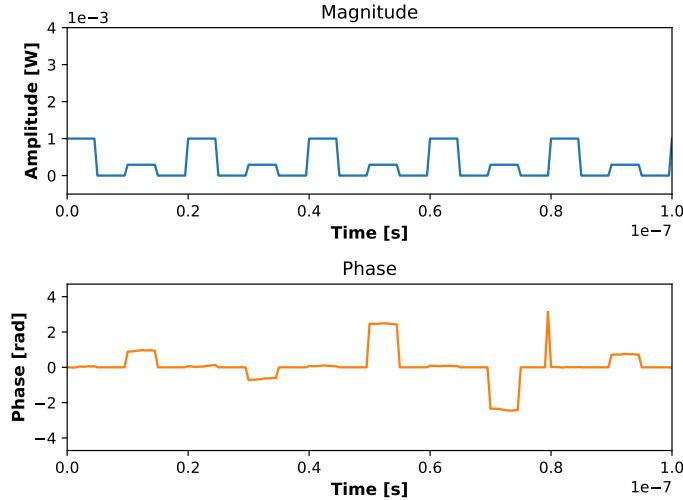
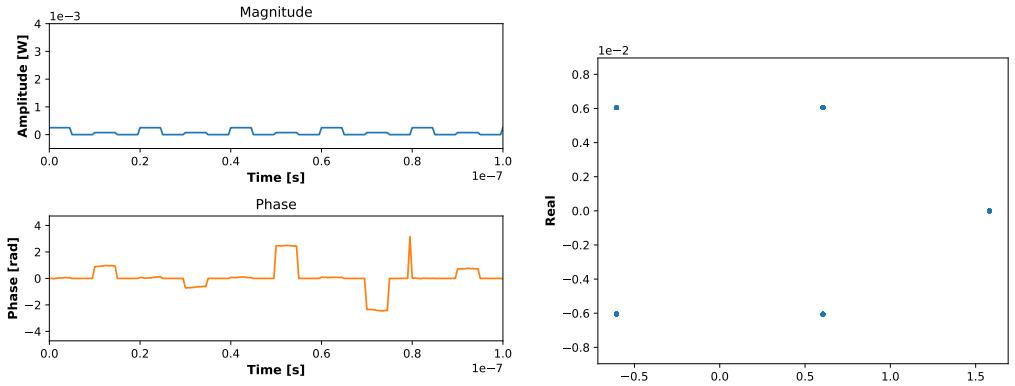


Figure 6.220: Magnitude and phase time dependence of  $S_2(t)$ .

$$S_A(t) = a S_1 e^{i\phi(t)} \left[ \sin\left(\frac{\pi I(t)}{V_\pi}\right) + i \sin\left(\frac{\pi Q(t)}{V_\pi}\right) \right]. \quad (6.221)$$

Signal  $S_A(t)$  is then mixed in a  $90^\circ$  Optical Hybrid with signal  $S_L(t)$ , the latter described by



(a) Magnitude and phase time dependence of  $S_A(t)$ .

(b) Signal constellation of  $S_A(t)$ .

$$S_L(t) = S_L e^{i\varphi(t)}, \quad (6.222)$$

where the initial phase of  $S_L(t)$ ,  $\varphi$ , follows a Wiener-Levy process independent from, but

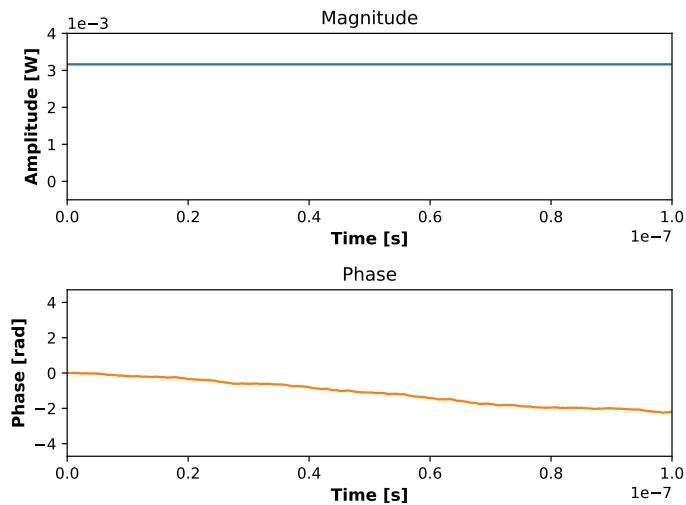


Figure 6.222: Magnitude and phase time dependence of  $S_L(t)$ .

similar to, the one that governs the evolution of  $\phi(t)$ . The amplitude of the signal,  $S_L$ , is assumed to take the value  $\sqrt{10^{-2}} \sqrt{W}$ . The signals resulting of this mixing, presented in (6.223), are then fed into two Balanced Homodyne Receivers. At this point the frequency mismatch becomes noticeable, due to the introduction of a beat frequency. In this study the beat frequency was assumed to take the value  $\omega_B = 10$  MHz.

$$\begin{bmatrix} S_\alpha \\ S_\beta \\ S_\gamma \\ S_\delta \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & i \\ 1 & -i \end{bmatrix} \begin{bmatrix} S_A(t) \\ S_L(t) \end{bmatrix}. \quad (6.223)$$

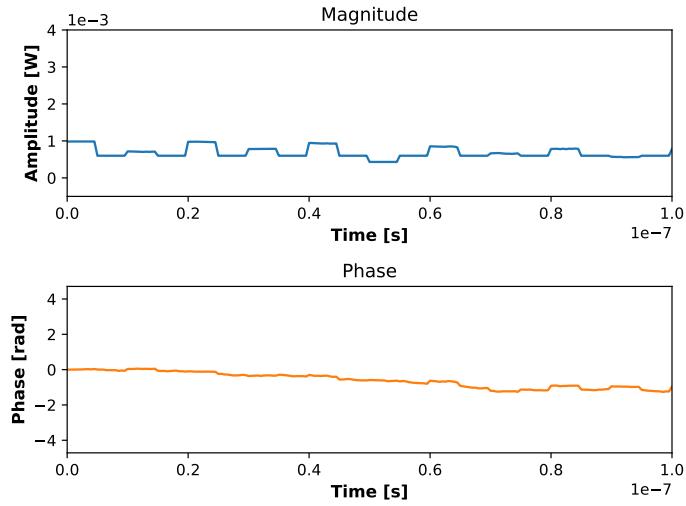


Figure 6.223: Magnitude and phase time dependence of  $S_\alpha(t)$ .

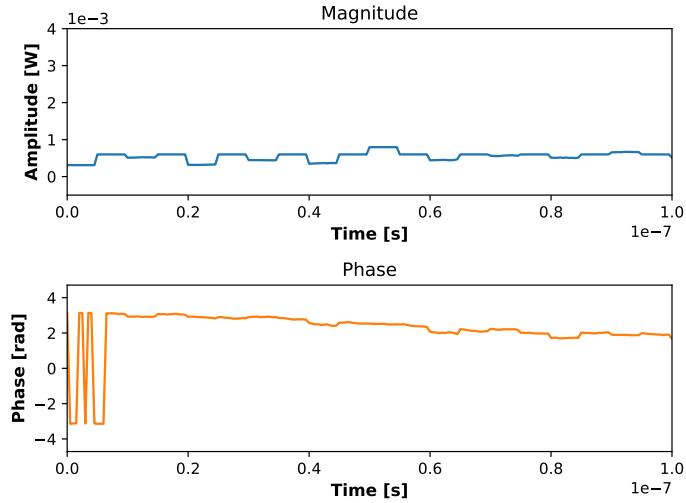
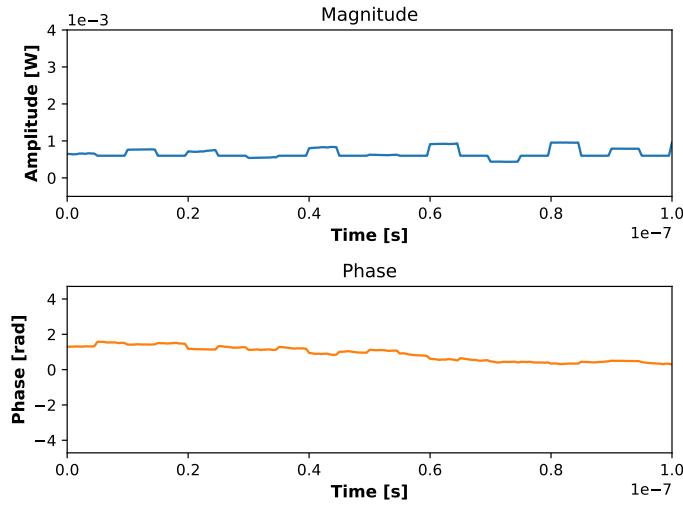
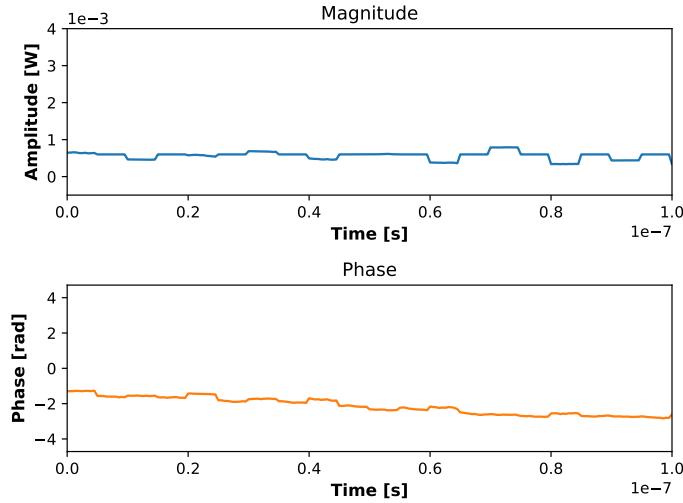


Figure 6.224: Magnitude and phase time dependence of  $S_\beta(t)$ .

The internal scheme of the upper balanced homodyne receiver in Figure 6.173 is presented in Figure 6.182. The internal signal  $S_-(t)$  corresponds to the following subtraction current

$$\begin{aligned}
 S_-(t) &= \rho (|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\
 &= \rho \frac{a S_1 S_L}{2} \left[ \sin \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) \right], \tag{6.224}
 \end{aligned}$$

Figure 6.225: Magnitude and phase time dependence of  $S_\gamma(t)$ .Figure 6.226: Magnitude and phase time dependence of  $S_\delta(t)$ .

where  $\rho$  is the responsivity of the photodiodes. This signal is then amplified by a gain factor  $g$ , resulting in the signal

$$\begin{aligned}
 S_I(t) &= gS_-(t) = g\rho(|S_\alpha(t)|^2 - |S_\beta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) \right]. \tag{6.225}
 \end{aligned}$$

The lower balanced homodyne receiver in Figure 6.173, presented in Figure 6.184, functions

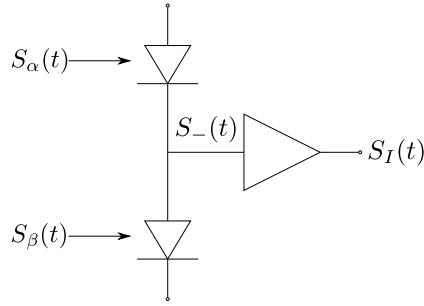
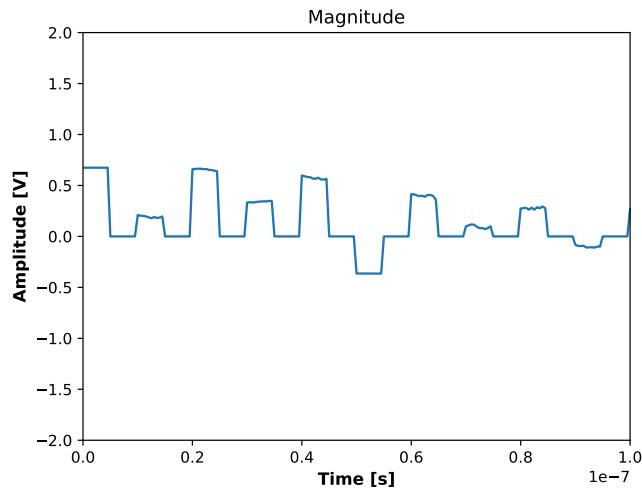


Figure 6.227: Internal diagram of the first homodyne receiver.

Figure 6.228: Amplitude time dependence of  $S_I(t)$ .

in a similar way to the upper homodyne receiver, with the subtraction current now given by

$$\begin{aligned}
 S_-(t) &= \rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) \right], \tag{6.226}
 \end{aligned}$$

which is also amplified by a gain factor  $g$  and results in the signal

$$\begin{aligned}
 S_Q(t) &= g\rho (|S_\gamma(t)|^2 - |S_\delta(t)|^2) \\
 &= \rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi Q(t)}{V_\pi} \right) - \sin \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi Q(t)}{V_\pi} \right) + \right. \\
 &\quad \left. \cos \left( \omega_B t + \phi(t) - \varphi(t) - \frac{\pi I(t)}{V_\pi} \right) - \cos \left( \omega_B t + \phi(t) - \varphi(t) + \frac{\pi I(t)}{V_\pi} \right) \right]. \tag{6.227}
 \end{aligned}$$

The  $S_I(t)$  and  $S_Q(t)$  signals are then sampled by a sampling function  $\Delta(t)$ , defined as

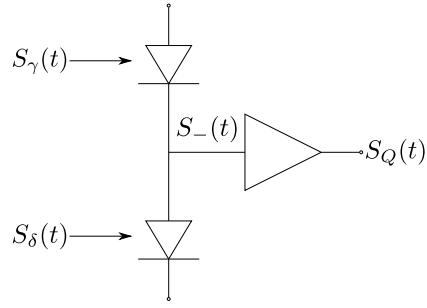
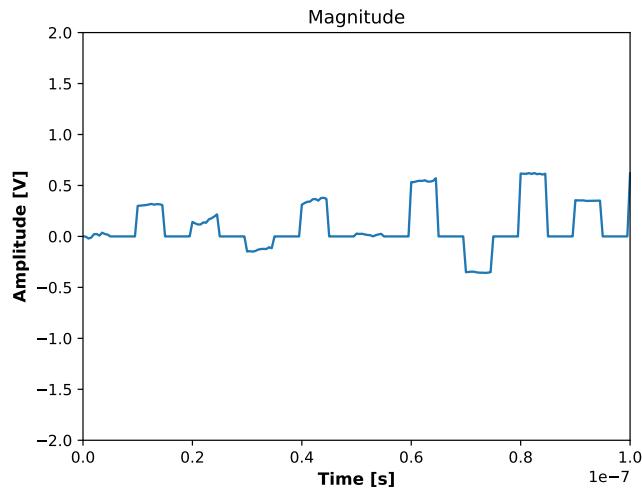


Figure 6.229: Internal diagram of the second homodyne receiver.

Figure 6.230: Amplitude time dependence of  $S_Q(t)$ .

$$\Delta(t) = \sum_{n=-\infty}^{n=+\infty} \delta \left( t - \left( n + \frac{1}{4} \right) T_s \right), \text{ with } \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & \text{otherwise} \end{cases}, \quad (6.228)$$

resulting in the signals

$$\begin{aligned} S_I(n) &= \Delta(t) S_I(t) = S_I \left( \left( n + \frac{1}{4} \right) T_s \right) = S_I(t_n) \\ &= g\rho \frac{aS_1S_L}{2} \left[ \sin \left( \omega_B t_n + \phi(t_n) - \varphi(t_n) + \frac{\pi I(t_n)}{V_\pi} \right) - \sin \left( \omega_B t_n + \phi(t_n) - \varphi(t_n) - \frac{\pi I(t_n)}{V_\pi} \right) + \right. \\ &\quad \left. \cos \left( \omega_B t_n + \phi(t_n) - \varphi(t_n) + \frac{\pi Q(t_n)}{V_\pi} \right) - \cos \left( \omega_B t_n + \phi(t_n) - \varphi(t_n) - \frac{\pi Q(t_n)}{V_\pi} \right) \right], \quad n \in \mathbb{Z} \end{aligned} \quad (6.229)$$

$$\begin{aligned}
S_Q(n) &= \Delta(t)S_Q(t) = S_Q\left(\left(n + \frac{1}{4}\right)T_s\right) = S_Q(t_n) \\
&= g\rho \frac{aS_1S_L}{2} \left[ \sin\left(\omega_B t_n + \phi(t_n) - \varphi(t_n) + \frac{\pi Q(t_n)}{V_\pi}\right) - \sin\left(\omega_B t_n + \phi(t_n) - \varphi(t_n) - \frac{\pi Q(t_n)}{V_\pi}\right) + \right. \\
&\quad \left. \cos\left(\omega_B t_n + \phi(t_n) - \varphi(t_n) - \frac{\pi I(t_n)}{V_\pi}\right) - \cos\left(\omega_B t_n + \phi(t_n) - \varphi(t_n) + \frac{\pi I(t_n)}{V_\pi}\right) \right], \quad n \in \mathbb{Z}
\end{aligned} \tag{6.230}$$

The reference constellation can be recovered by isolating all the even-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$IQ_R(m) = S_I(m) + iS_Q(m) = g\rho a S_1 S_L e^{i(\omega_B t_m + \phi(t_m) - \varphi(t_m))}, \quad m = 2n, \quad n \in \mathbb{Z}, \tag{6.231}$$

while the phase encoded constellation can be recovered by isolating the odd-numbered results of  $S_I(n)$  and  $S_Q(n)$

$$\begin{aligned}
IQ_S(l) &= S_I(l) + iS_Q(l) \\
&= g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\omega_B t_l + \phi(t_l) - \varphi(t_l) + \frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\omega_B t_l + \phi(t_l) - \varphi(t_l) + \frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\omega_B t_l + \phi(t_l) - \varphi(t_l) + \frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\omega_B t_l + \phi(t_l) - \varphi(t_l) + \frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}.
\end{aligned} \tag{6.232}$$

Both  $IQ_R(m)$  and  $IQ_S(l)$  are plotted in Figure 6.231. The blue points correspond to the  $IQ_S(l)$  signal with the 01001110 sequential encoding while the orange ones correspond to the  $IQ_R(m)$  signal. The effect of the phase noise is visible in the dragging rotation of the 4 state constellation, which results in the circular constellation visible in Figure 6.231. The

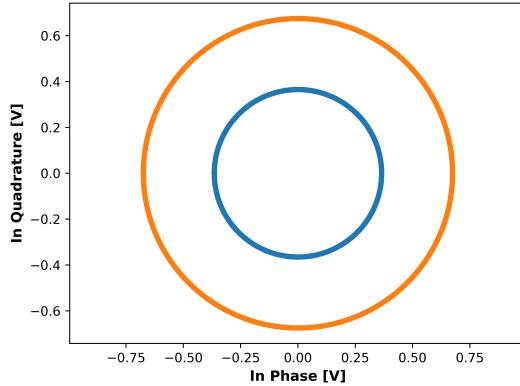


Figure 6.231: Recovered constellation assuming a large acquisition time.

$\omega_B t_m$  and  $\omega_B t_l$  elements in the equations for the reference and signal constellations are responsible for the dragging rotation visible in Figure 6.216. This effect can be removed by estimating the value of  $\omega_B$  and multiplying the reference and signal constellations by  $e^{-i\omega_B t_j}$  and  $e^{-i\omega_B(t_j + T_s)}$ , where  $t_j$  attempts to estimate the values of  $t_m$ . The beat frequency

$\omega_B$  can be determined by multiplying an incremented reference constellation by the complex conjugate of the unincremented constellation, resulting in

$$IQ_{\text{beat}} = IQ_R(m+2)IQ_R^*(m) = (g\rho a S_1 S_L)^2 e^{i(\omega_B 2T_s + \phi(t_{m+2}) - \phi(t_m) - (\varphi(t_{m+2}) - \varphi(t_m)))}, \quad (6.233)$$

dividing the complex phase of this signal by  $2T_s$  will yield a normally distributed random variable with mean  $\omega_B$  and variance  $2\pi\Delta f T_s^{-1}$ . The exacteness of this estimate can be evaluated through a confidence interval. Assuming that the variance is not well known, being estimated from the experimental results, the confidence interval is given by

$$P\left(\omega_B - t \frac{s}{\sqrt{n}} < \omega_B < \omega_B + t \frac{s}{\sqrt{n}}\right) = 1 - \alpha, \quad (6.234)$$

where  $t$  is the  $1 - \frac{\alpha}{2}$ 'th percentile of the Student's t-distribution and  $s^2$  is the estimated value of the variance. For the utilized beat frequency of  $\omega_B = 10$  MHz, assuming  $n = 2 \times 10^5$  samples were used and that the variance estimate is given by  $s^2 = 2\pi\Delta f T_s^{-1}$ , the 99 % confidence interval is less than 1 % of  $\omega_B$ . We thus see that this method for estimating the beat frequency utilizes an acceptably low number of samples, does not require a high sampling rate (only one sample per pulse is necessary) and has a low computational complexity.

The beat frequency compensated constellations are then given by

$$IQ_r(m) = IQ_R(m)e^{-i\omega_B t_j} = g\rho a S_1 S_L e^{i(\omega_B(t_m - t_j) + \phi - \varphi)}, \quad m = 2n, \quad n \in \mathbb{Z}, \quad (6.235)$$

and

$$IQ_s(l) = IQ_s(l)e^{-i\omega_B(t_j + T_s)} = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\omega_B(t_l - t_j - T_s) + \phi - \varphi + \frac{\pi}{4})}, & l = 8n + 1 \\ e^{i(\omega_B(t_l - t_j - T_s) + \phi - \varphi + \frac{7\pi}{4})}, & l = 8n + 3 \\ e^{i(\omega_B(t_l - t_j - T_s) + \phi - \varphi + \frac{3\pi}{4})}, & l = 8n + 5 \\ e^{i(\omega_B(t_l - t_j - T_s) + \phi - \varphi + \frac{5\pi}{4})}, & l = 8n + 7 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.236)$$

The constellations for  $IQ_r(m)$  and  $IQ_s(l)$  are presented in Figure 6.217, where it was assumed that  $t_j$  misestimates  $t_m$  by  $10T_s$ . The original constellation can be recovered by removing the phase of  $IQ_R(m)$  from  $IQ_s(l)$

$$IQ = IQ_S(l)e^{-i\arg(IQ_R(m))} = g\rho a S_1 S_L \sqrt{2} \sin\left(\frac{\pi}{k}\right) \begin{cases} e^{i(\omega_B(t_l - t_m) + \phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{\pi}{4})}, & l = 8n + 1, \quad m = l - 1 \\ e^{i(\omega_B(t_l - t_m) + \phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{7\pi}{4})}, & l = 8n + 3, \quad m = l - 1 \\ e^{i(\omega_B(t_l - t_m) + \phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{3\pi}{4})}, & l = 8n + 5, \quad m = l - 1 \\ e^{i(\omega_B(t_l - t_m) + \phi(t_l) - \varphi(t_l) - \phi(t_m) + \varphi(t_m) + \frac{5\pi}{4})}, & l = 8n + 7, \quad m = l - 1 \end{cases}, \quad n \in \mathbb{Z}. \quad (6.237)$$

The constellation obtained after this phase drift compensation scheme is presented in Figure 6.233. The remaining phase drift observed is due to the laser's phase drifting during the timing difference between the signal and reference pulses, thus the quality of the phase

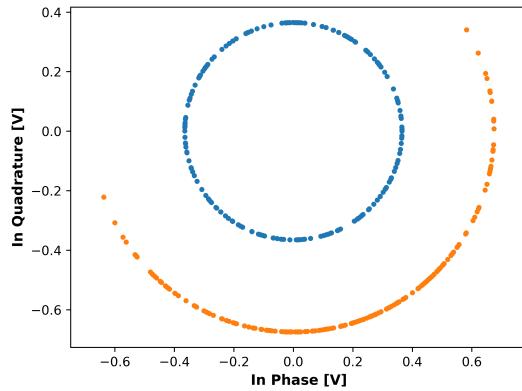


Figure 6.232: Recovered constellation after the phase drift compensation step for a symbol period of  $T_s = 10$  ns assuming  $t_j$  misestimates  $t_m$  by  $10T_s$ .

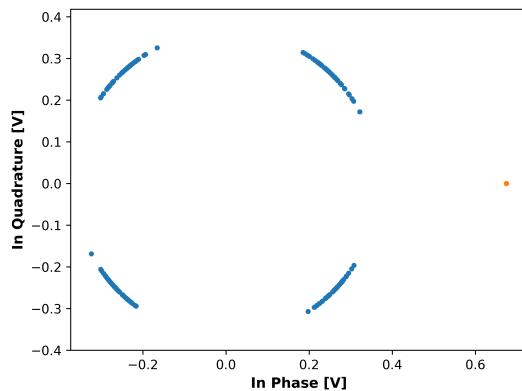


Figure 6.233: Recovered constellation after the phase drift compensation step for a symbol period of  $T_s = 1 \mu\text{s}$ .

drift compensation scheme improves with reduced symbol periods. From Figure 6.233 it becomes apparent that a faster repetition rate is necessary for the laser properties assumed in this study. We thus show that both the frequency mismatch and the phase noise can be compensated for with our scheme with an acceptable number of samples. After the phase originating from the initial phase difference between the  $S_L(t)$  and  $S_A(t)$  signals is removed, the constellation can be decoded in accordance to the state to bit correspondence in Figure 6.172.

### 6.11.5 Simulation Analysis

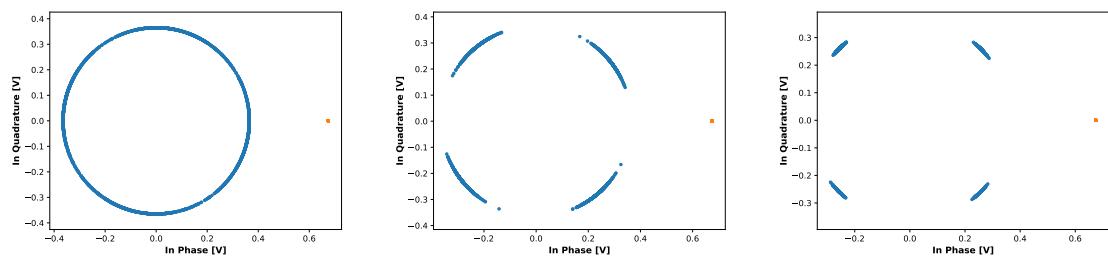


Figure 6.234: Recovered constellations after the phase drift compensation step for symbol periods of 0.1, 0.01, and 0.001  $\mu$ s.

## References

- [1] Ezra Ip and Joseph M Kahn. "Feedforward carrier recovery for coherent optical communications". In: *Journal of Lightwave Technology* 25.9 (2007), pp. 2675–2692.
- [2] Md Saifuddin Faruk and Seb J Savory. "Digital signal processing for coherent transceivers employing multilevel formats". In: *Journal of Lightwave Technology* 35.5 (2017), pp. 1125–1141.
- [3] Xiang Zhou et al. "64-Tb/s, 8 b/s/Hz, PDM-36QAM transmission over 320 km using both pre-and post-transmission digital signal processing". In: *Journal of Lightwave Technology* 29.4 (2011), pp. 571–577.
- [4] Mehrez Selmi, Yves Jaouën, and Philippe Ciblat. "Accurate digital frequency offset estimator for coherent PolMux QAM transmission systems". In: *Optical Communication, 2009. ECOC'09. 35th European Conference on*. IEEE. 2009, pp. 1–2.
- [5] Xian Zhou, Xue Chen, and Keping Long. "Wide-range frequency offset estimation algorithm for optical coherent systems using training sequence". In: *IEEE Photonics Technology Letters* 24.1 (2012), pp. 82–84.
- [6] Michael G Taylor. "Phase estimation methods for optical coherent detection using digital signal processing". In: *Journal of Lightwave Technology* 27.7 (2009), pp. 901–914.
- [7] Maurizio Magarini et al. "Pilot-symbols-aided carrier-phase recovery for 100-G PM-QPSK digital coherent receivers". In: *IEEE Photonics Technology Letters* 24.9 (2012), p. 739.
- [8] MB Costa Silva et al. "Homodyne detection for quantum key distribution: an alternative to photon counting in BB84 protocol". In: *Photonics North 2006*. Vol. 6343. International Society for Optics and Photonics. 2006, 63431R.
- [9] John Bertrand Johnson. "Thermal agitation of electricity in conductors". In: *Physical review* 32.1 (1928), p. 97.
- [10] Harry Nyquist. "Thermal agitation of electric charge in conductors". In: *Physical review* 32.1 (1928), p. 110.
- [11] T. C. Ralph. "Continuous variable quantum cryptography". In: *Phys. Rev. A* 61 (1 Dec. 1999), p. 010303. DOI: [10.1103/PhysRevA.61.010303](https://doi.org/10.1103/PhysRevA.61.010303). URL: <https://link.aps.org/doi/10.1103/PhysRevA.61.010303>.
- [12] Mark Hillery. "Quantum cryptography with squeezed states". In: *Physical Review A* 61.2 (2000), p. 022309.
- [13] Xiang-Chun Ma et al. "Wavelength attack on practical continuous-variable quantum-key-distribution system with a heterodyne protocol". In: *Physical Review A* 87.5 (2013), p. 052309.
- [14] Jing-Zheng Huang et al. "Quantum hacking on quantum key distribution using homodyne detection". In: *Physical Review A* 89.3 (2014), p. 032304.

- [15] Paul Jouguet, Sébastien Kunz-Jacques, and Eleni Diamanti. "Preventing calibration attacks on the local oscillator in continuous-variable quantum key distribution". In: *Physical Review A* 87.6 (2013), p. 062313.
- [16] Jing-Zheng Huang et al. "Quantum hacking of a continuous-variable quantum-key-distribution system using a wavelength attack". In: *Physical Review A* 87.6 (2013), p. 062329.
- [17] Bing Qi et al. "Generating the local oscillator "locally" in continuous-variable quantum key distribution based on coherent detection". In: *Physical Review X* 5.4 (2015), p. 041009.
- [18] Daniel BS Soh et al. "Self-referenced continuous-variable quantum key distribution protocol". In: *Physical Review X* 5.4 (2015), p. 041010.
- [19] Adrien Marie and Romain Alléaume. "Self-coherent phase reference sharing for continuous-variable quantum key distribution". In: *Physical Review A* 95.1 (2017), p. 012316.
- [20] S Kleis and CG Schaeffer. "Comparison of frequency estimation methods for heterodyne quantum communications". In: *Photonic Networks; 18. ITG-Symposium; Proceedings of*. VDE. 2017, pp. 1–3.

## 6.12 BPSK Transmission System

<b>Student Name</b>	:	André Mourato (2018/01/28 - 2018/02/27) Daniel Pereira (2017/09/01 - 2017/11/16)
<b>Goal</b>	:	Estimate the BER in a Binary Phase Shift Keying optical transmission system with additive white Gaussian noise. Comparison with theoretical results.
<b>Directory</b>	:	sdf/bpsk_system

Binary Phase Shift Keying (BPSK) is the simplest form of Phase Shift Keying (PSK), in which binary information is encoded into a two state constellation with the states being separated by a phase shift of  $\pi$  (see Figure 6.235).

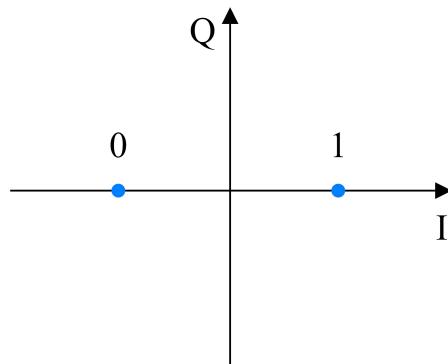


Figure 6.235: BPSK symbol constellation.

White noise is a random signal with equal intensity at all frequencies, having a constant power spectral density. White noise is said to be Gaussian (WGN) if its samples follow a normal distribution with zero mean and a certain variance  $\sigma^2$ . For WGN its spectral density equals its variance. For the purpose of this work, additive WGN is used to model thermal noise at the receivers.

The purpose of this system is to simulate BPSK transmission in back-to-back configuration with additive WGN at the receiver and to perform an accurate estimation of the BER and validate the estimation using theoretical values.

### 6.12.1 Theoretical Analysis

The output of the system with added gaussian noise follows a normal distribution, whose first probabilistic moment can be readily obtained by knowledge of the optical power of the received signal and local oscillator,

$$m_i = 2\sqrt{P_L P_S} G_{ele} \cos(\Delta\theta_i), \quad (6.238)$$

where  $P_L$  and  $P_S$  are the optical powers, in watts, of the local oscillator and signal, respectively,  $G_{ele}$  is the gain of the trans-impedance amplifier in the coherent receiver and

$\Delta\theta_i$  is the phase difference between the local oscillator and the signal, for BPSK this takes the values  $\pi$  and 0, in which case (6.238) can be reduced to,

$$m_i = (-1)^{i+1} 2 \sqrt{P_L P_S} G_{ele}, \quad i = 0, 1. \quad (6.239)$$

The second moment is directly chosen by inputting the spectral density of the noise  $\sigma^2$ , and thus is known *a priori*.

Both probabilist moments being known, the probability distribution of measurement results is given by a simple normal distribution,

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m_i)^2}{2\sigma^2}}. \quad (6.240)$$

The BER is calculated in the following manner,

$$BER = \frac{1}{2} \int_0^{+\infty} f(x|\Delta\theta = \pi) dx + \frac{1}{2} \int_{-\infty}^0 f(x|\Delta\theta = 0) dx, \quad (6.241)$$

given the symmetry of the system, this can be simplified to,

$$BER = \int_0^{+\infty} f(x|\Delta\theta = \pi) dx = \frac{1}{2} \operatorname{erfc} \left( \frac{-m_0}{\sqrt{2}\sigma} \right) \quad (6.242)$$

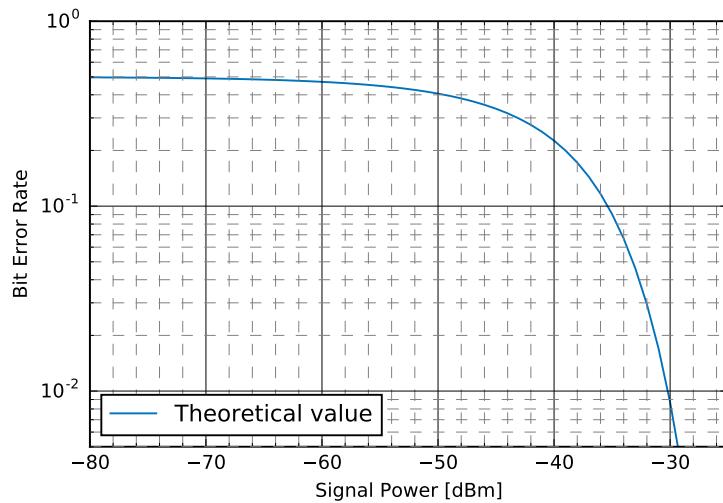


Figure 6.236: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm.

### 6.12.2 Simulation Analysis

A diagram of the system being simulated is presented in the Figure 6.237. A random binary sequence is generated and encoded in an optical signal using BPSK modulation. The decoding of the optical signal is accomplished by an homodyne receiver, which combines the

signal with a local oscillator. The received binary signal is compared with the transmitted binary signal in order to estimate the Bit Error Rate (BER). The simulation is repeated for multiple signal power levels. Each corresponding BER is recorded and plotted against the expectation value.

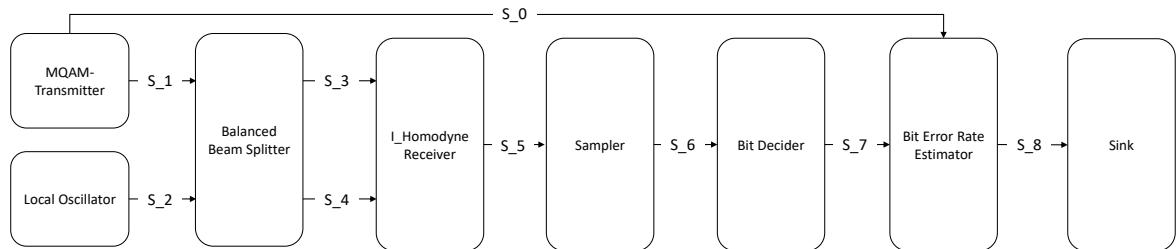


Figure 6.237: Overview of the BPSK system being simulated.

## Required files

Header Files		
File	Comments	Status
add_20171116.h		✓
balanced_beam_splitter_20180124.h		✓
binary_source_20180118.h		✓
bit_decider_20170818.h		✓
bit_error_rate_20171810.h		✓
discrete_to_continuous_time_20180118.h		✓
i_homodyne_receiver_20180124.h		✓
ideal_amplifier_20180118.h		✓
iq_modulator_20180130.h		✓
local_oscillator_20180130.h		✓
m_qam_mapper_20180118.h		✓
m_qam_transmitter_20180118.h		✓
netxpto_20180418.h		✓
photodiode_old_20180118.h		✓
pulse_shaper_20180118.h		✓
sampler_20171116.h		✓
sink_20180118.h		✓
super_block_interface_20180118.h		✓
ti_amplifier_20180102.h		✓
white_noise_20180118.h		✓

Source Files		
File	Comments	Status
add_20171116.cpp		✓
balanced_beam_splitter_20180124.cpp		✓
binary_source_20180118.cpp		✓
bit_decider_20170818.cpp		✓
bit_error_rate_20171810.cpp		✓
discrete_to_continuous_time_20180118.cpp		✓
i_homodyne_receiver_20180124.cpp		✓
ideal_amplifier_20180118.cpp		✓
iq_modulator_20180130.cpp		✓
local_oscillator_20180130.cpp		✓
m_qam_mapper_20180118.cpp		✓
m_qam_transmitter_20180118.cpp		✓
netxpto_20180418.cpp		✓
photodiode_old_20180118.cpp		✓
pulse_shaper_20180118.cpp		✓
sampler_20171116.cpp		✓
sink_20180118.cpp		✓
super_block_interface_20180118.cpp		✓
ti_amplifier_20180102.cpp		✓
white_noise_20180118.cpp		✓

### System Input Parameters

This system takes into account the following input parameters:

System Input Parameters		
Parameter	Default Value	Comments
numberOfBitsReceived	-1	
numberOfBitsGenerated	1000	
samplesPerSymbol	16	
pLength	5	
bitPeriod	$20 \times 10^{-12}$	
rollOffFactor	0.3	
signalOutputPower_dBm	-20	
localOscillatorPower_dBm	0	
localOscillatorPhase	0	
iqAmplitudesValues	$\{ \{-1, 0\}, \{1, 0\} \}$	
transferMatrix	$\{ \{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}\} \}$	
responsivity	1	
amplification	$10^6$	
electricalNoiseAmplitude	$5 \times 10^{-4}\sqrt{2}$	
samplesToSkip	$8 \times \text{samplesPerSymbol}$	
bufferLength	20	
shotNoise	false	

## Inputs

This system takes no inputs.

## Outputs

This system outputs the following objects:

System Output Signals	
Signal	Associated File
Initial Binary String ( $S_0$ )	S0.sgn
Optical Signal with coded Binary String ( $S_1$ )	S1.sgn
Local Oscillator Optical Signal ( $S_2$ )	S2.sgn
Beam Splitter Outputs ( $S_3, S_4$ )	S3.sgn & S4.sgn
Homodyne Receiver Electrical Output ( $S_5$ )	S5.sgn
Sampler Output ( $S_6$ )	S6.sgn
Decoded Binary String ( $S_7$ )	S7.sgn
BER Result String ( $S_8$ )	S8.sgn
Report	Associated File
Bit Error Rate Report	BER.txt

## Bit Error Rate - Simulation Results

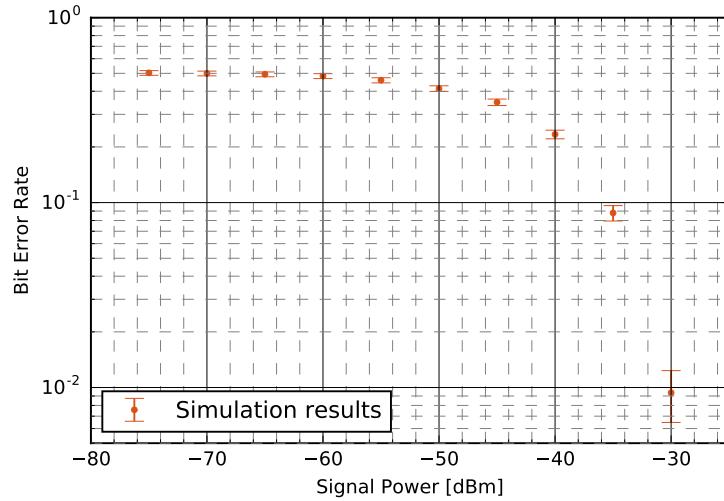


Figure 6.238: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm.

### 6.12.3 Comparative Analysis

The following results show the dependence of the error rate with the signal power assuming a constant Local Oscillator power of 0 dBm, the signal power was evaluated at levels between -70 and -25 dBm, in steps of 5 dBm between each. The simulation results are presented in orange with the computed lower and upper bounds, while the expected value, obtained from (6.242), is presented as a full blue line. A close agreement is observed between the simulation results and the expected value. The noise spectral density was set at  $5 \times 10^{-4}\sqrt{2} \text{ V}^2$  [1].

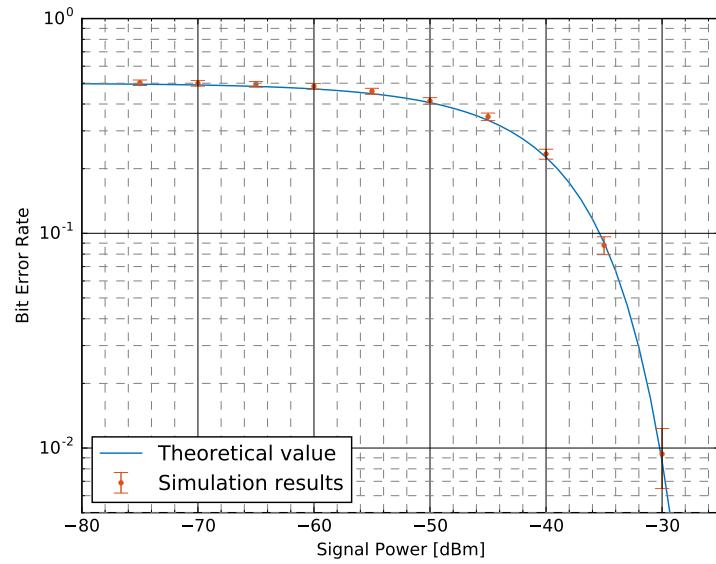


Figure 6.239: Bit Error Rate in function of the signal power in dBm at a constant local oscillator power level of 0 dBm. Theoretical values are presented as a full blue line while the simulated results are presented as a errorbar plot in orange, with the upper and lower bound computed in accordance with the method described in 7.4

## References

- [1] Thorlabs. *Thorlabs Balance Amplified Photodetectors: PDB4xx Series Operation Manual*. 2014.

## 6.13 Intradyne Continuous Variables QKD Transmission System

<b>Student Name</b>	:	Nuno Silva (11/01/2018 - )
<b>Goal</b>	:	Simulation and experimental validation of a CV-QKD transmission system with a real local oscillator.
<b>Directory</b>	:	sdf/cv_intradyne_system

The aim of Continuous Variable Quantum Key Distribution (CV-QKD) is to encode information in observables whose measurements take continuous values, such as the phase and amplitude of a electromagnetic wave, or the stokes parameters. In this work we are going to analyze a CV-QKD system which uses the phase and amplitude degrees of freedom of a coherent light field to generate a secret key. Two protocols are going to be analyzed for implement a CV-QKD system: (1) using a continuous Gaussian-modulated coherent state (continuous constellation) at transmitter side; (2) using a discrete-modulated coherent state (M-ary PSK) at transmitter side. In both protocols, the receiver will implemented a double homodyne detection scheme with an optical hybrid, and locally generated local oscillator, i.e. intradyne detection. Due to phase/frequency uncorrelation between quantum signals and local oscillator, the transmitter will send a strong (classical) reference signal for phase/frequency reference.

### 6.13.1 Theoretical Analysis

In this section we are going to describe detailed the CV-QKD setup, using continuous- and discrete-modulated coherent states. The continuous Gaussian-modulation of a coherent state in principle leads to a higher achievable information rate between the two legitimate users of a CV-QKD system. However, in practice it is preferable to use the discrete-modulation of a coherent state (such as the QPSK constellation) since for this case there exist high efficient error correction codes even at very low SNR values. These two approaches to the implementation of a CV-QKD system will be analyzed in the next two sections.

#### Gaussian-Modulated Coherent States

In the Gaussian-modulated coherent state CV-QKD protocol, Alice randomly generates a coherent state  $|\alpha\rangle$  with  $\alpha = |\alpha|e^{i\theta}$  where  $|\alpha|$  is the amplitude of the coherent-state and  $\theta$  is its phase. In this notation the coherent state can be written

$$|\alpha\rangle = \exp\left(-\frac{1}{2}|\alpha|^2\right) \sum_{n=0}^{+\infty} \frac{\alpha^n}{(n!)^{\frac{1}{2}}} |n\rangle \quad (6.243)$$

$$= \exp\left(-\frac{1}{2}|\alpha|^2\right) \sum_{n=0}^{+\infty} \frac{(\alpha\hat{a}^\dagger)^n}{(n!)} |0\rangle, \quad (6.244)$$

where  $|n\rangle$  represent the number state, and  $\hat{a}^\dagger$  ( $\hat{a}$ ) represents the creation (annihilation) operator, respectively, which obey to the commutation relation

$$[\hat{a}, \hat{a}^\dagger] = \hat{a}\hat{a}^\dagger - \hat{a}^\dagger\hat{a} = 1. \quad (6.245)$$

The coherent state obeys to the following properties

$$\hat{a}|\alpha\rangle = \alpha|\alpha\rangle, \quad (6.246)$$

$$\langle\alpha|\hat{a}^\dagger = \langle\alpha|\alpha^*, \quad (6.247)$$

$$\langle\alpha|\alpha\rangle = 1, \quad (6.248)$$

$$\langle\alpha|\beta\rangle = \exp\left(-\frac{1}{2}|\alpha|^2 - \frac{1}{2}|\beta|^2 + \alpha^*\beta\right), \quad (6.249)$$

$$|\langle\alpha|\beta\rangle|^2 = \exp(-|\alpha - \beta|^2). \quad (6.250)$$

$$(6.251)$$

Note that the coherent states  $|\alpha\rangle$  and  $|\beta\rangle$  become orthogonal when  $|\alpha - \beta|^2 \gg 1$ .

The coherent-state expectation value for the number operator and its variance are given by

$$\bar{n} = \langle\alpha|\hat{n}|\alpha\rangle = |\alpha|^2 \quad (6.252)$$

$$(\Delta n)^2 = \langle\alpha|(\hat{n})^2|\alpha\rangle = \langle\alpha|\hat{a}^\dagger\hat{a}^\dagger\hat{a}\hat{a}|\alpha\rangle + \langle\alpha|\hat{n}|\alpha\rangle - (\langle\alpha|\hat{n}|\alpha\rangle)^2 = |\alpha|^2 \quad (6.253)$$

where  $\hat{n} = \hat{a}^\dagger\hat{a}$ . The probability of finding  $n$  photons in the coherent-state  $|\alpha\rangle$  is

$$P(n) = |\langle n|\alpha\rangle|^2 = \exp(-|\alpha|^2) \frac{|\alpha|^{2n}}{n!} = e^{-\bar{n}} \frac{\bar{n}^n}{n!}, \quad (6.254)$$

which is a Poisson probability distribution. For large values of  $\bar{n}$ , the Poisson distribution approaches a Gaussian distribution, by using  $\bar{n}^n = \exp(n \ln(\bar{n}))$

$$P(n) \approx \frac{1}{\sqrt{2\pi\bar{n}}} \exp\left(-\frac{(n - \bar{n})^2}{2\bar{n}}\right) \quad (6.255)$$

For a coherent-state  $|\alpha\rangle$  with  $\alpha = |\alpha_A|e^{i\theta}$ , the corresponding phase distribution is

$$P(\varphi) = |\langle\varphi|\alpha\rangle|^2 = \frac{1}{2\pi} e^{-\frac{1}{2}|\alpha|^2} \left| \sum_{n=0}^{+\infty} e^{in(\varphi-\theta)} \frac{|\alpha|^n}{\sqrt{n!}} \right|^2, \quad (6.256)$$

with  $\bar{\varphi} = \theta$ . For  $|\alpha|^2 \gg 1$ ,  $P(\varphi)$  tends to a Gaussian distribution

$$P(\varphi) \approx \left(\frac{2|\alpha|^2}{\pi}\right)^{\frac{1}{2}} \exp(-2\bar{n}(\varphi - \theta)^2). \quad (6.257)$$

In this case  $(\Delta\varphi)^2 = 1/(4\bar{n})$ . The number-phase uncertainty is therefore

$$(\Delta n)^2 (\Delta\varphi)^2 = 1/4. \quad (6.258)$$

An important point to analyze now is the quantization of the electromagnetic field. The electromagnetic field is quantized by the association of a quantum-mechanical harmonic-oscillator with each mode of radiation. We assume a one-dimensional harmonic oscillator for this work. In that case the Hamiltonian is

$$\hat{H} = \frac{1}{2} (\hat{p}^2 + \omega^2 \hat{q}^2), \quad (6.259)$$

where  $\omega$  is the frequency of the mode,  $\hat{p}$  and  $\hat{q}$  are Hermitian operators and therefore correspond to observable quantities, with  $\hat{p}$  being the momentum operator and  $\hat{q}$  the position operator which obey to the commutation relation

$$[\hat{q}, \hat{p}] = i\hbar. \quad (6.260)$$

It is convenient non-Hermitian (and therefore non-observable) annihilation and creation operators

$$\hat{a} = (2\hbar\omega)^{-1/2} (\omega\hat{q} + i\hat{p}), \quad (6.261)$$

$$\hat{a}^\dagger = (2\hbar\omega)^{-1/2} (\omega\hat{q} - i\hat{p}). \quad (6.262)$$

Then the electric and the magnetic field operators can be written as

$$\hat{E}_x(z, t) = \hat{E}_x^+(z, t) + \hat{E}_x^-(z, t) \quad (6.263)$$

$$= \sqrt{\frac{\hbar\omega}{2\epsilon_0 V}} (\hat{a}e^{-i\chi} + \hat{a}^\dagger e^{i\chi}), \quad (6.264)$$

$$\hat{B}_y(z, t) = \hat{B}_y^+(z, t) + \hat{B}_y^-(z, t) \quad (6.265)$$

$$= -i \frac{\epsilon_0 \mu_0}{k} \sqrt{\frac{\hbar\omega}{2\epsilon_0 V}} (\hat{a}e^{-i\chi} + \hat{a}^\dagger e^{i\chi}), \quad (6.266)$$

where  $\chi = \omega t - kz - \pi/2$  is the phase of the electromagnetic field, with  $k = \omega/c$  is the wave number,  $V$  is the efective volume of the cavity used for quantization,  $\epsilon_0$  is the vacuum permitivity,  $\mu_0$  is the vacuum permaability,  $c$  is the speed of light in vacuum, and the plus and minus symbols are known as the positive and negative frequency parts of the electromagnetic field. From the electromagnetic field operators we can right the radiation Hamiltonian

$$\hat{H}_R = \frac{1}{2} \int dV \left( \epsilon_0 \hat{E}_x(z, t) \hat{E}_x(z, t) + \frac{1}{\mu_0} \hat{B}_y(z, t) \hat{B}_y(z, t) \right) \quad (6.267)$$

$$= \frac{1}{2} (\hat{p}^2 + \omega^2 \hat{q}^2). \quad (6.268)$$

We give particular attention to the electric field operator because of its important role in the the definitions of degrees of coherence. We now introduce the so-called quadrature operators

$$\hat{X} = \frac{1}{2} (\hat{a}^\dagger + \hat{a}), \quad (6.269)$$

$$\hat{Y} = \frac{i}{2} (\hat{a}^\dagger - \hat{a}), \quad (6.270)$$

with

$$[\hat{X}, \hat{Y}] = i/2. \quad (6.271)$$

The electric field can be written using these quadrature operators

$$\hat{E}_x(\chi) = \sqrt{\frac{2\hbar\omega}{\epsilon_0 V}} (\hat{X} \cos(\chi) + \hat{Y} \sin(\chi)) \quad (6.272)$$

The electric field operator obey to the commutation relation

$$[\hat{E}_x(\chi_1), \hat{E}_x(\chi_2)] = -\frac{i}{2} \sin(\chi_1 - \chi_2). \quad (6.273)$$

Let us now consider the expectation values for those operators considering the coherent-state

$$\langle \alpha | \hat{n} | \alpha \rangle = \frac{1}{2\hbar\omega} (\omega^2 \langle \alpha | \hat{q}^2 | \alpha \rangle + \langle \alpha | \hat{p}^2 | \alpha \rangle - \hbar\omega) = |\alpha_A|^2 \quad (6.274)$$

with

$$\langle \alpha | \hat{q}^2 | \alpha \rangle = \frac{\hbar}{2\omega} (2|\alpha|^2 + 1 + \alpha^2 + (\alpha^*)^2), \quad (6.275)$$

$$\langle \alpha | \hat{p}^2 | \alpha \rangle = \frac{\hbar\omega}{2} (2|\alpha|^2 + 1 - \alpha^2 - (\alpha^*)^2). \quad (6.276)$$

The variance of the momentum and position operators is

$$(\Delta q)^2 = \langle \alpha | \hat{q}^2 | \alpha \rangle - (\langle \alpha | \hat{q} | \alpha \rangle)^2 = \frac{\hbar}{2\omega}, \quad (6.277)$$

$$(\Delta p)^2 = \langle \alpha | \hat{p}^2 | \alpha \rangle - (\langle \alpha | \hat{p} | \alpha \rangle)^2 = \frac{\hbar\omega}{2}, \quad (6.278)$$

with

$$\langle \alpha | \hat{q} | \alpha \rangle = \sqrt{\frac{\hbar}{2\omega}} (\alpha + \alpha^*), \quad (6.279)$$

$$\langle \alpha | \hat{p} | \alpha \rangle = i\sqrt{\frac{\hbar\omega}{2}} (\alpha^* - \alpha). \quad (6.280)$$

In terms of quadrature operators we have

$$\langle \alpha | \hat{X} | \alpha \rangle = \frac{1}{2} (\alpha + \alpha^*) = |\alpha| \cos(\theta), \quad (6.281)$$

$$\langle \alpha | \hat{Y} | \alpha \rangle = \frac{i}{2} (\alpha^* - \alpha) = |\alpha| \sin(\theta), \quad (6.282)$$

$$\langle \alpha | \hat{X}^2 | \alpha \rangle = \frac{1}{4} (2|\alpha|^2 + \alpha^2 + 1 + (\alpha^*)^2) = \frac{1}{4} + |\alpha|^2 \cos^2(\theta), \quad (6.283)$$

$$\langle \alpha | \hat{Y}^2 | \alpha \rangle = \frac{1}{4} (2|\alpha|^2 + 1 - \alpha^2 - (\alpha^*)^2) = \frac{1}{4} + |\alpha|^2 \sin^2(\theta). \quad (6.284)$$

The variance of the quadrature operators is

$$(\Delta X)^2 = \langle \alpha | \hat{X}^2 | \alpha \rangle - \left( \langle \alpha | \hat{X} | \alpha \rangle \right)^2 = \frac{1}{4}, \quad (6.285)$$

$$(\Delta Y)^2 = \langle \alpha | \hat{Y}^2 | \alpha \rangle - \left( \langle \alpha | \hat{Y} | \alpha \rangle \right)^2 = \frac{1}{4}, \quad (6.286)$$

thus the coherent-state is a quadrature minimum uncertainty state for all mean photon numbers  $|\alpha_A|^2$ .

The expectation values for the electric field operator is given by

$$\langle \alpha | \hat{E}_x(\chi) | \alpha \rangle = |\alpha| \sqrt{\frac{2\hbar\omega}{\epsilon_0 V}} \cos(\chi - \theta), \quad (6.287)$$

$$\left\langle \alpha \left| \left( \hat{E}_x(\chi) \right)^2 \right| \alpha \right\rangle = \frac{2\hbar\omega}{\epsilon_0 V} \left( \frac{1}{4} + \frac{|\alpha|^2}{2} \{ 1 + \cos(2\theta) \cos(2\chi) + 2 \sin(2\theta) \sin(\chi) \cos(\chi) \} \right). \quad (6.288)$$

The variance of the electric field is given by

$$(\Delta E_x(\chi))^2 = \left\langle \alpha \left| \left( \hat{E}_x(\chi) \right)^2 \right| \alpha \right\rangle - \left( \langle \alpha | \hat{E}_x(\chi) | \alpha \rangle \right)^2 = \frac{1}{4} \left( \frac{2\hbar\omega}{\epsilon_0 V} \right), \quad (6.289)$$

The coherent-states  $|\alpha\rangle$  are quantum states very close to classical states because: (1) the expectation value of the electric field has the form of the classical expression; (2) the fluctuations in the electric field variables are the same as for a vacuum; (3) the fluctuation in the fractional uncertainty for the photon number decrease with the increasing average photon number; (4) the states become well localized in phase with the increasing average photon number. Moreover, the coherent state may be generated by a classical oscillating current, such as laser radiation which is produced in a resonant cavity where the resonant frequency of the cavity is the same as the frequency associated with the atomic electron transitions providing energy flow into the field.

In the Gaussian-modulated coherent state CV-QKD protocol, Alice randomly generates a coherent state  $|\alpha\rangle$  with  $\alpha = |\alpha_A| e^{i\theta_A} = x_A + i p_A$ . In this protocol for CV-QKD,  $x_A$  and  $p_A$  represents two independent Gaussian variables. In this protocol Alice prepares displaced coherent states with quadrature components  $x_A = |\alpha_A| \cos(\theta)$  and  $p_A = \sin(\theta)$  which are realizations of two independent and identically distributed (i.i.d.) random variables  $Q$  and  $P$ . The random variables  $Q$  and  $P$  obey to zero-centered normal distribution

$$Q \sim N(0, V_{\text{mod}_q}), \quad (6.290)$$

$$P \sim N(0, V_{\text{mod}_p}), \quad (6.291)$$

where  $V_{\text{mod}_q}$  and  $V_{\text{mod}_p}$  represents the modulation variance of  $x_A$  and  $p_A$ . This means that

$$\langle \alpha | \hat{X} | \alpha \rangle = E \{ x_A \} = 0, \quad (6.292)$$

$$\langle \alpha | \hat{Y} | \alpha \rangle = E \{ p_A \} = 0, \quad (6.293)$$

$$\langle \alpha | \hat{X}^2 | \alpha \rangle = \frac{1}{4} + E \{ x_A^2 \}, \quad (6.294)$$

$$\langle \alpha | \hat{Y}^2 | \alpha \rangle = \frac{1}{4} + E \{ p_A^2 \}, \quad (6.295)$$

$$\bar{n} = \langle \alpha | \hat{X}^2 | \alpha \rangle + \langle \alpha | \hat{Y}^2 | \alpha \rangle - \frac{1}{2} = E \{ x_A^2 \} + E \{ p_A^2 \} = V_{\text{mod}_q} + V_{\text{mod}_p}, \quad (6.296)$$

where  $E \{ \cdot \}$  represents the expectation value. Note that, the variance of a coherent state is equal to  $\bar{n}$ . This leads to

$$(\Delta X)^2 = \langle \alpha | \hat{X}^2 | \alpha \rangle = \frac{1}{4} + V_{\text{mod}_q}, \quad (6.297)$$

$$(\Delta Y)^2 = \langle \alpha | \hat{Y}^2 | \alpha \rangle = \frac{1}{4} + V_{\text{mod}_p}. \quad (6.298)$$

If we define  $N_0 = 1/4$  which represents the shot-noise variance of a single mode coherent state (identical to the fluctuation of a vacuum state), we have

$$(\Delta X)^2 = \langle \alpha | \hat{X}^2 | \alpha \rangle = N_0 \left( 1 + \frac{V_{\text{mod}_q}}{N_0} \right), \quad (6.299)$$

$$(\Delta Y)^2 = \langle \alpha | \hat{Y}^2 | \alpha \rangle = N_0 \left( 1 + \frac{V_{\text{mod}_p}}{N_0} \right). \quad (6.300)$$

Assuming that  $V_{\text{mod}_q}/N_0 = V_{\text{mod}_p}/N_0 = V_A$ , which leads to

$$Q \sim P \sim N(0, V_A), \quad (6.301)$$

$$\bar{n} = 2V_A. \quad (6.302)$$

The variance of Alice's field quadratures is

$$V = \langle x_A^2 \rangle = \langle p_A^2 \rangle = (V_A + 1)N_0 \quad (6.303)$$

The  $x_A$  and  $p_A$  components can be generated experimentally by using an amplitude modulator followed by a phase modulator. We are going now relate  $x_A$  and  $p_A$  with the transfer functions of amplitude and phase modulators. According to the Box-Muller transform,  $x_A$  and  $p_A$  can be generated from a pair of uniformly distributed random numbers ( $U_1$  and  $U_2$ ) over the interval  $[0, 1]$

$$x_A = \sqrt{-2V_A \ln(U_1)} \cos(2\pi U_2), \quad (6.304)$$

$$p_A = \sqrt{-2V_A \ln(U_1)} \sin(2\pi U_2), \quad (6.305)$$

with

$$|\alpha_A| = \sqrt{x_A^2 + p_A^2} = \sqrt{-2V_A \ln(U_1)}, \quad (6.306)$$

$$\theta_A = \arccos \left( \frac{x_A}{|\alpha_A|} \right) = 2\pi U_2, \quad (6.307)$$

where  $\theta_A$  has an uniform distribution on  $[0, 2\pi]$ , and  $|\alpha_A|$  obeys the Rayleigh distribution

$$P(|\alpha_A|) = \frac{|\alpha_A|}{V_A} e^{-\frac{|\alpha_A|^2}{2V_A}}. \quad (6.308)$$

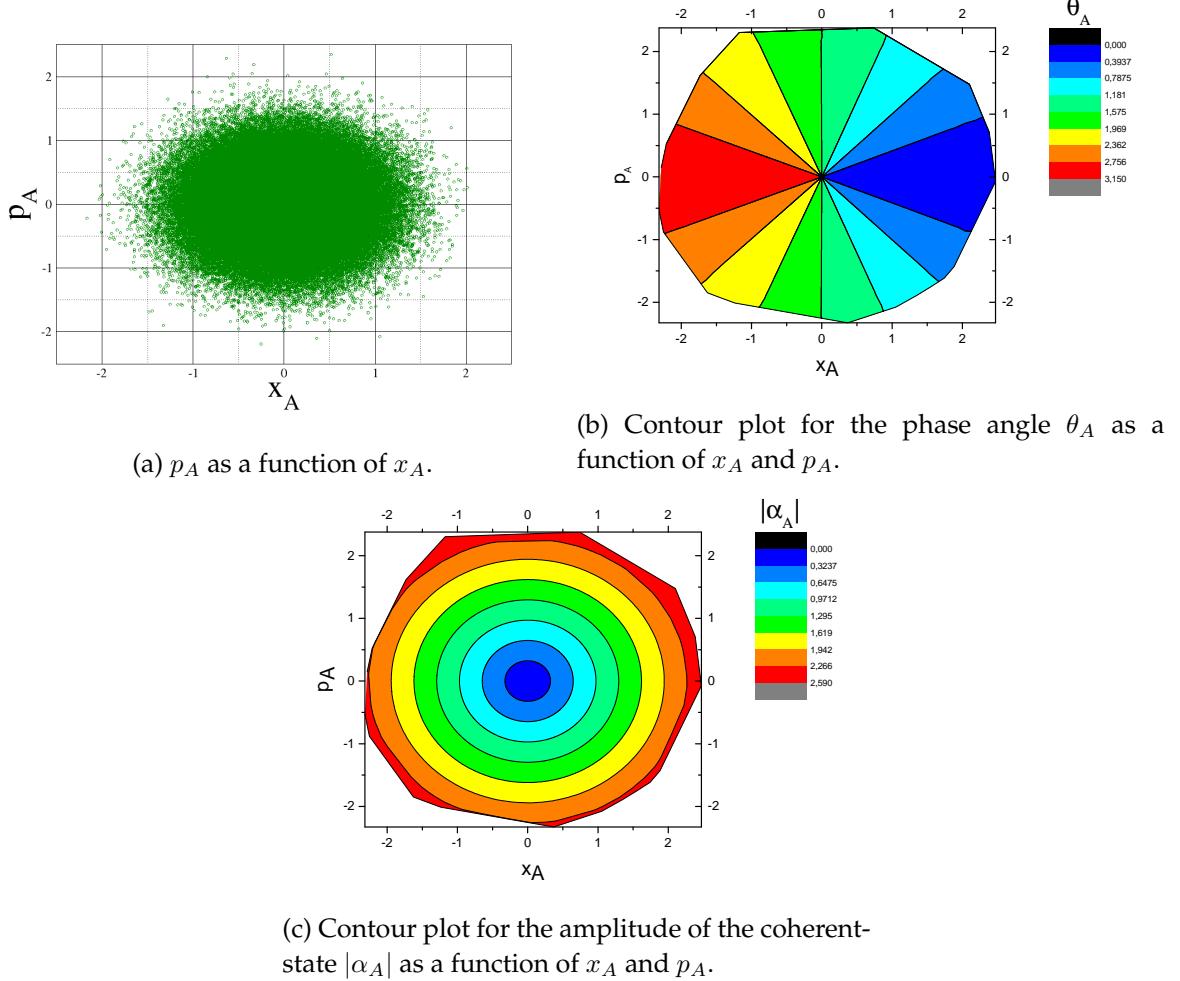


Figure 6.240: (a): Quadrature values in the phase space for a bivariate Gaussian distribution obtained using (6.304); (b) and (c): Contour plots for the phase and amplitude of the coherent-state given by (6.306). In the figure we have used  $V_A = \bar{n}/2 = 0.25$ , and we have used  $10^6$  points for generate each of the uniform distributions in (6.304).

Therefore, an amplitude modulator and a phase modulator are sufficient to achieve above bivariate Gaussian distribution.

The transfer function of a Mach-Zhender (MZ) intensity modulator followed by a phase modulator (PM) can be written as

$$T = t_{MZ} t_{PM} \sin \left( \frac{\pi}{2} \frac{V_{\min} - V_{MZ}}{V_{\pi}^{MZ}} \right) e^{i\pi V_{PM}/V_{\pi}^{PM}}, \quad (6.309)$$

where  $(1 - t_{MZ}^2)$  is the insertion loss coefficient of the MZ,  $(1 - t_{PM}^2)$  is the insertion loss coefficient of the PM,  $V_{MZ}$  is the modulation signal voltage impinged in the MZ,  $V_{\min}$  is the voltage in the MZ which corresponds to a minimum transmission of the modulator,  $V_{\pi}^{MZ}$  is the half-wave voltage. Moreover,  $V_{PM}$  is the modulation voltage impinged in the phase modulator, and  $V_{\pi}^{PM}$  is the half-voltage of the PM. From (6.304) and (6.309) the voltages impinged in the MZ and phase modulators can be written as

$$V_{PM} = 2U_2 V_{\pi}^{PM}, \quad (6.310)$$

$$V_{MZ} = V_{\min} - \frac{2}{\pi} V_{\pi}^{MZ} \arcsin \left( \frac{\sqrt{-2V_A \ln(U_1)}}{t_{MZ} t_{PM} |\alpha_{in}|} \right), \quad (6.311)$$

where  $\alpha_{in}$  is the amplitude of the coherent-state at the modulators input, with  $|\alpha_A|$  its output amplitude. The implementation of the Gaussian modulated quadrature transmitter (Alice) can be seen in Fig. 6.241

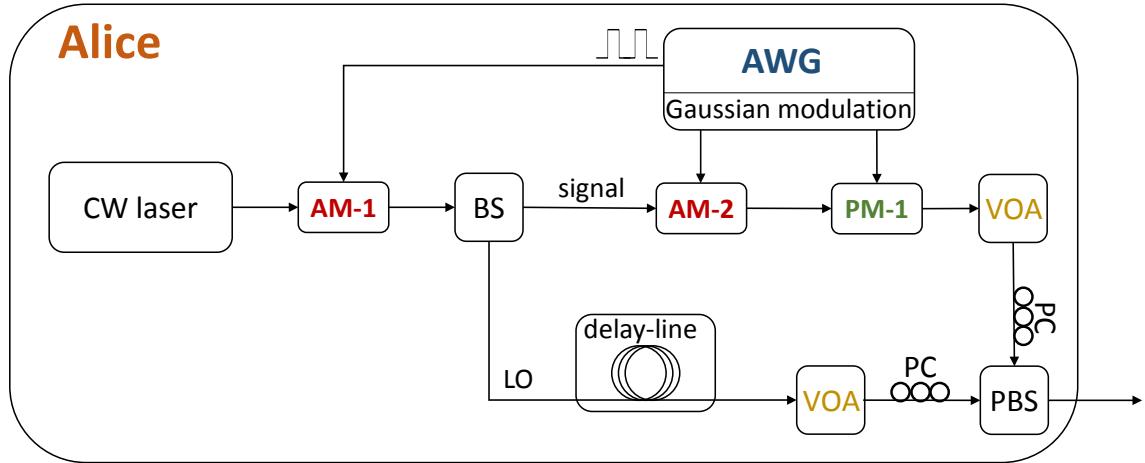


Figure 6.241: Scheme for implement a Gaussian modulated transmitter for being used in CV-QKD systems. In the figure: AM, amplitude modulator; BS, beam-splitter; PM, phase modulator; VOA, variable optical attenuator; PC, polarization controller; PBS, polarization beam-splitter; AWG, arbitrary waveform generator.

Figure 6.241 depicts the scheme for implement a Gaussian modulated coherent state transmitter. Alice uses an AM-1 with a continuous-wave laser to produce a coherent state pulse. Then she uses a beam splitter (BS) to split the coherent state into two portions, where one serves as signal and the other as local oscillator. A combination of MZ amplitude modulator (AM-2) and phase modulator (PM-1) is used to archived a bivariate Gaussian modulation, such as in 6.240. Then the Gaussian modulated signal is adjusted to an optimized variance of  $V_A$  by tuning a variable optical attenuator (VOA). The modulated signal is then combined with the local oscillator by means of a polarization beam-splitter (PBS), and both are sent to the quantum channel. Note that, by using the polarization-multiplexing and time-multiplexing techniques, the signal together with LO are sent to Bob avoiding leakage of photons from LO to the quantum signal

After propagation in the quantum channel channel the Gaussian modulated signal and local oscillator reaches the Bob receiver. Bob can perform homodyne or intradyne detection to extract information from the incoming signal and local oscillator. We are going to start with the Homodyne detection, see Fig. 6.242.

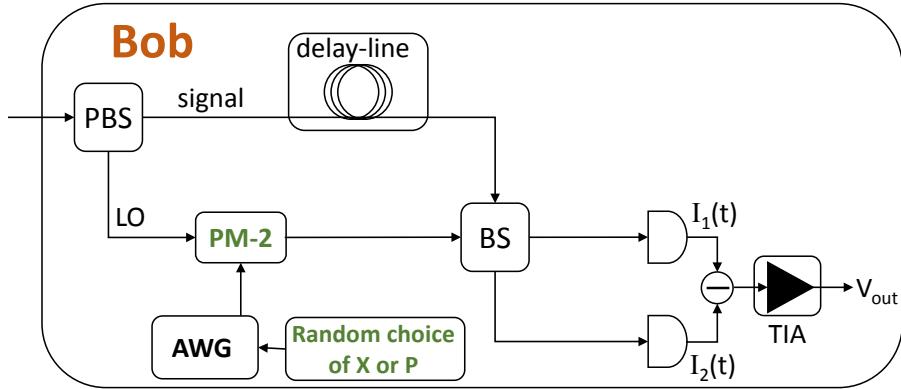


Figure 6.242: Scheme for implementing a Homodyne detection scheme for being used in CV-QKD systems. In the figure: PM, amplitude modulator; BS, beam-splitter; PM, phase modulator; VOA, variable optical attenuator; PBS, polarization beam-splitter; TIA, transimpedance amplifier.

Figure 6.242 shows a Homodyne receiver that can be used to extract the information sent by Alice. At Bob's side, the LO and the signal are polarization and time demultiplexed by using a PBS and a delay line, respectively. The LO passes through a phase modulator (PM-2) which is used to measure either  $\hat{X}$  or  $\hat{Y}$  quadratures generated by Alice. The signal and the LO will further be launched in a beam-splitter where they will interfere and after measured by two balanced photo-diodes. The photo-diodes currents  $I_1(t)$  and  $I_2(t)$  will be subtracted and amplified by means of a transimpedance amplifier, where the current intensity is amplified and converted to voltage  $V_{\text{out}}$ . We consider that the LO is a high intensity classical optical signal

$$E_{\text{LO}}(t) = \frac{1}{2} \left( A_{\text{LO}}(t) e^{i(\omega_{\text{LO}} t + \phi_{\text{LO}}(t))} + A_{\text{LO}}^*(t) e^{-i(\omega_{\text{LO}} t + \phi_{\text{LO}}(t))} \right), \quad (6.312)$$

where  $A_{\text{LO}}(t)$  is the amplitude of the LO electric field,  $\omega_{\text{LO}}$  is the optical frequency of the LO, and  $\phi_{\text{LO}}(t)$ , represents the phase of the LO. After the phase modulator PM-2 in Fig. 6.242, the LO electric field is given by

$$E_{\text{LO}}^{\text{out}}(t) = E_{\text{LO}}(t) t_{\text{PM-2}} e^{i\pi V_{\text{PM-2}}/V_{\pi}^{\text{PM-2}}} \quad (6.313)$$

### Pilot-aided locally generated LO

The detection of the Gaussian modulated quadrature can be performed using an intradyne coherent receiver, assuming that the local oscillator (LO) is generated in the coherent receiver by using an independent laser source at the receiver's end. In that case, the main challenge

is how to effectively establish a reliable phase reference between Alice and Bob, since the quantum signal contains only few photons. While various techniques, such as feedforward carrier recovery, optical phase-locked loops, and optical injection phase-locked loops, have been developed in classical coherent communication, these techniques are not suitable in QKD where the quantum signal is extremely weak and the tolerable phase noise is low. In that scenario, we can solve the above problem by implement a pilot-aided feedforward data recovery scheme, see Fig. 6.243.

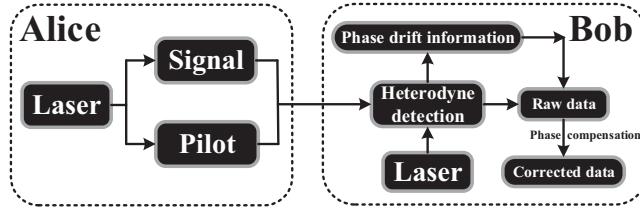


Figure 6.243: Pilot-aided locally generated LO for CV-QKD. Alice first generates the signal and the pilot, both of which are then transmitted to Bob. On the other side, Bob produces the local oscillator to interfere with the signal and the pilot. After the heterodyne (i.e. intradyne) detection, Bob obtains the raw data as well as the phase drift information, which is then utilized to execute the phase compensation yielding the corrected data.

In Fig. 6.243 it is presented the basic idea beyond the pilot-aided feedforward data recovery scheme to be used in CV-QKD systems. Nevertheless, in those systems two conditions need to be met: (1) the LO must be generated at the reception; (2) a large number of pilot pulses carrying the phase information need to be transmitted. However, the production, transmission and detection ways of the pilot pulses are optional and a different arrangement may finally affect the system performance. Three different schemes were proposed to implement a pilot-aided CV-QKD setup.

The first one is the pilot-sequential scheme, see Fig. 6.244.

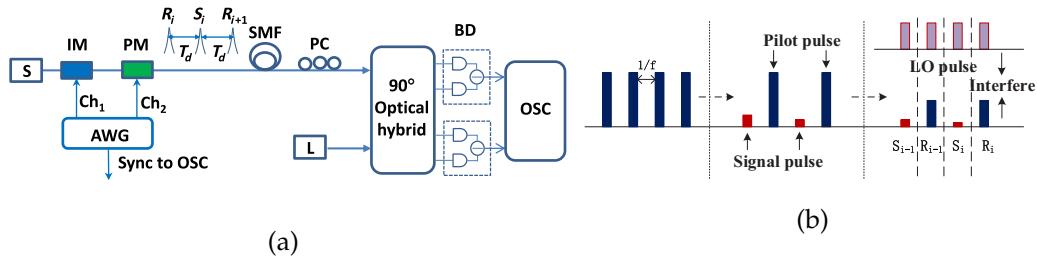


Figure 6.244: The scheme and procedure to implement a pilot-sequential setup.

To begin with, Alice produces a coherent state pulse chain, whose pulse interval is  $1/f$ . Then the odd sequence number pulses are taken as the signal and modulated according to a centered Gaussian distribution, while the even sequence number pulses are regarded as the pilot and passed without any change. At the reception, Bob produces a LO pulse

chain, whose pulse interval is also  $1/f$ . Through adjusting the optical length, the signal and pilot pulses are accurately aligned with the LO pulses. After the heterodyne (i.e. intradyne) detection, the Gaussian key information is obtained from the interference measurement of the signal pulse, while the phase drift information is contained in the interference measurement of the pilot pulse. After the data acquisition of the interference measurement, one can use the adjacent pilot phase to estimate the signal's phase drift. Since the signal pulse  $S_i$  is in the middle of two pilot pulses  $R_{i-1}$  and  $R_i$ , the relative phase drift can be estimated from the phase on  $R_{i-1}$  and  $R_i$ . However, due to the rapid phase fluctuation during the time interval, the phase estimator has an intrinsic estimation error yielding a constant phase noise.

The second one is the pilot-delay-line scheme, see Fig. 6.245.

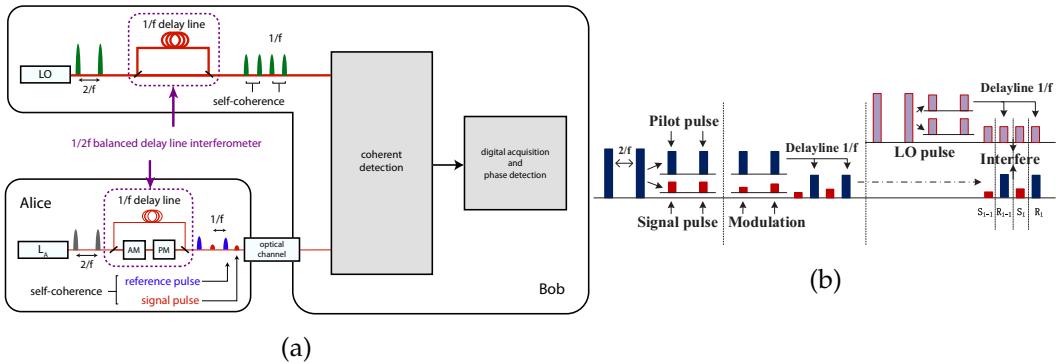


Figure 6.245: The scheme and procedure to implement a pilot-delay-line setup.

The pilot-delayline scheme essential idea is generating the signal pulse and the pilot pulse simultaneously to eliminate the phase fluctuation error. The procedure of the pilot-delayline scheme is illustrated in Fig. 6.245. First, Alice generates a coherent state pulse sequence with a  $2/f$  interval. Then she splits the coherent state into the signal pulse and the pilot pulse. The signal pulse is modulated according to a centered Gaussian distribution, while the pilot pulse is delayed for  $1/f$ , inserted into the signal pulse sequence, and transmitted to Bob. At the reception, the LO pulse sequence is also split into two identical sequences. One is delayed with a time  $1/f$  and then inserted into the other. Through aligning the LO pulses with the signal as well as the pilot pulses, the Gaussian key information and the corresponding phase information can be obtained from the interference measurement of the signal and the pilot, respectively. In the pilot-delayline scheme, since the pilot pulse is generated simultaneously with the corresponding signal pulse, both of them have the same optical phase. Therefore, one can utilize this pilot pulse as a phase monitor to execute an accurate compensation. However, due to the inherent unbalanced interferometer structure, such design requires a stable interferometric setup. Unfortunately, in the practical scenario, the transmitted optical length may fluctuate with a relatively slow rate, making this desired stability hard to maintain.

Recently, a new scheme was proposed to implement the pilot-aided CV-QKD system that solves the problems in the pilot-sequential and pilot-delay-line setups. The pilot-

multiplexed scheme introduces time- and polarization multiplexing techniques and uses two heterodyne (i.e. intradyne) receivers to detect the signal and pilot separately, see Fig. 6.246.

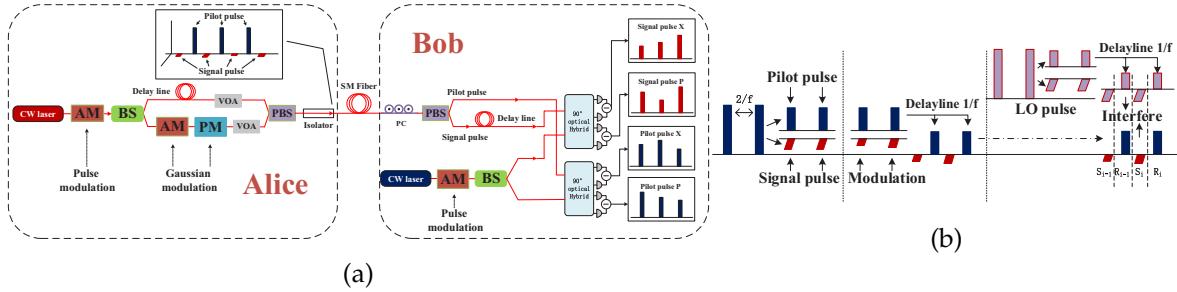


Figure 6.246: The scheme and procedure to implement a pilot-multiplexed setup.

The procedure of the pilot-multiplexed scheme is illustrated in Fig. 6.246. Specifically, the signal pulse and the pilot pulse are also derived from a single pulse to make their optical phases identical. After the signal modulation, Alice delays the reference path for  $1/f$  to realize the time-multiplexing. Besides, orthogonal polarization channels are occupied to transmit the signal pulse and the pilot pulse realizing the polarization multiplexing. At the reception, these polarization-multiplexing pulses are divided and sent to different heterodyne (i.e. intradyne) detectors. Meanwhile, the LO pulse is split into two identical pulses, aligned with its corresponding signal pulse and pilot pulse, and interfered with them. The interference measurements of the signal as well as the pilot contain the key information and the phase drift information, respectively. In this scheme, we can divide the phase drift into the fast-drift part and slow-drift part, thus one can execute two remaps to compensate them separately.

The pilot-multiplexed scheme illustrated in Fig. 6.246 can also be used to implement a CV-QKD system with homodyne detection instead of intradyne, see Fig. 6.247

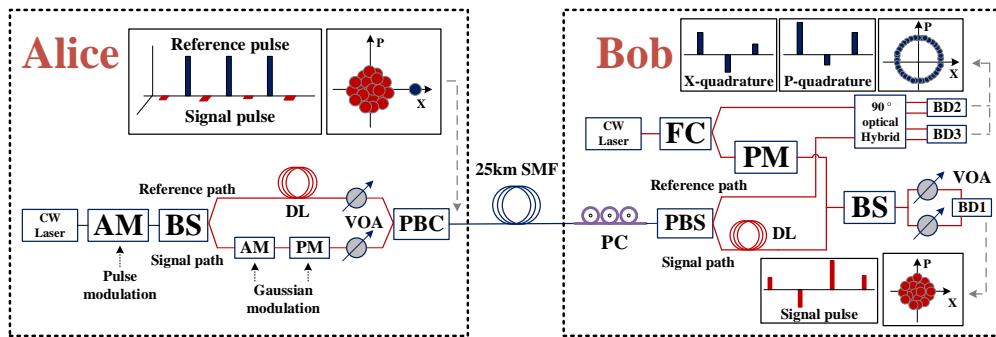


Figure 6.247: The scheme to implement a pilot-multiplexed setup using homodyne detection for the quantum signal.

When compared with 6.246, the scheme in Fig. 6.247 uses a beam-splitter and a pair of photodiodes instead an optical hybrid and four photodiodes. Nevertheless, to random basis

switch scheme in Fig. 6.247 uses a phase modulator in the LO arm.

We are going now to analyze the performance and the security of the pilot-aided CV-QKD system for the Gaussian modulated protocol. After modulation, Alice transmit to Bob the coherent state  $|\alpha_A\rangle = |x_A + ip_A\rangle$  to Bob through a quantum channel with transmission efficiency

$$T_{\text{ch}} = 10^{-\frac{\alpha_{\text{ch}} L_f}{10}}, \quad (6.314)$$

with  $\alpha_{\text{ch}}$  being the channel loss coefficient (in dB/km), and  $L_f$  the channel length (in km). The channel loss gives rise to a quantum noise coefficient  $(1 - T_{\text{ch}})/T_{\text{ch}}$ . Moreover, after propagation in the quantum channel the field quadratures sent by Alice can contain other noise sources, for instance from other classical or quantum signals in the same fiber, or from imperfections on Alice side (modulators and laser source). All these noise source are grouped in the same noise parameter  $\xi_A$  (typically known as excess of noise), such that at the end of the transmission channel the total added noise added by the channel is given by

$$\chi_{\text{ch}} = \frac{1 - T_{\text{ch}}}{T_{\text{ch}}} + \xi_A. \quad (6.315)$$

In addition to the obligatory shot-noise and the propagation noise induced by the quantum channel, various experimental imperfections will contribute to the total noise and undermine the system's performance. The constituents of  $\xi_A$  may originate from Alice modulation imperfections, from nonlinear out-of-band linear and nonlinear effects that generate noise at signal band (such ASE and Raman noise), from phase fluctuations due to the use of two laser sources, from quantisation at the ADCs, from detection noise (referred typically as receiver shot-noise which in this case represents the current fluctuations due to the discreteness of the electrical charge), from the imperfections on the balancing the photodiodes (known as common-mode rejection rate), from imperfections on the optical hybrid, among others. Assuming that all of these noise sources to be stochastically independent which makes the variances they cause to the quadratures additive

$$\xi_A = \xi_{\text{Mod}} + \xi_{\text{ON}} + \xi_{\text{Ph}} + \xi_{\text{ADC}} + \xi_{\text{det}} + \xi_{\text{CMMR}} + \xi_{\text{OH}} + \dots \quad (6.316)$$

Note that, the subscript  $A$  in  $\xi_A$  is referring to the channel input. We refer to this additional noise as excess noise  $\chi_{\text{det}}$ .

Finally, the homodyne or intradyne receiver it self introduces noise due to the fact that is not ideal due to the non-zero temperature, thermal fluctuations are an unavoidable source of noise

$$\chi_{\text{Hom}} = \frac{1 - \eta_{\text{det}}}{\eta_{\text{det}}} + \frac{v_{\text{el}}}{\eta_{\text{det}}}, \quad (6.317)$$

$$\chi_{\text{Het}} = \frac{2 - \eta_{\text{det}}}{\eta_{\text{det}}} + \frac{2v_{\text{el}}}{\eta_{\text{det}}}, \quad (6.318)$$

where  $\eta_{\text{det}}$  represents the homodyne or intradyne detector efficiency, and  $v_{\text{el}}$  represents the receiver electronic noise. Note that  $\chi_{\text{Hom}}$  or  $\chi_{\text{Het}}$  are referring to the channel output. The

total noise is given by

$$\chi = \chi_{\text{ch}} + \frac{\chi_{\text{det}}}{T_{\text{ch}}}, \quad (6.319)$$

with  $\chi_{\text{det}} = \chi_{\text{Hom}}$  for homodyne detection, and  $\chi_{\text{det}} = \chi_{\text{Het}}$  for heterodyne (intradyne) detection. Note that  $\chi_{\text{det}}$  is divided by  $T_{\text{ch}}$  in order to refer to the channel input.

After transmission over the loss and noisy channel, Bob will measure a quadrature variance

$$V_B = T_{\text{ch}}\eta_{\text{det}}(V + \chi N_0), \quad (6.320)$$

where  $V = V_A + 1$  as previously defined. This means that Bob will measure an attenuated and noise low intensity quantum signal sent by Alice.

In order to access to the performance of the CV-QKD system, we first calculate the secure key rate per pulse under the finite-size analysis

$$K_p = \left(1 - \frac{m}{N}\right) [\beta I_{\text{BA}} - I_{\text{BE}} - \Delta(n)], \quad (6.321)$$

where  $N$  is total number of signals exchanged to establish the key, and  $m$  are the number of signals used to estimate  $T_{\text{ch}}$  and  $\xi_A$ . This indicates that  $n = N - m$  represents are the number of signals used for establishment of the key. In (6.321),  $\beta$  is the efficiency of the reverse reconciliation protocol used for Alice and Bob extract their mutual information (including error correction). Moreover,  $I_{\text{BA}}$  represents the Shannon mutual information between Bob and Alice, and  $I_{\text{BE}}$  is the Shannon mutual information between Bob and Eve. Finally in (6.321),  $\Delta(n)$  is related with the security of the privacy amplification procedure. The mutual information in (6.321) depends on the kind of receiver that Bob uses, i.e.  $I_{\text{BA}} = I_{\text{BA}}^{\text{Hom}}$  if he uses a homodyne receiver such as in Fig. 6.247, and  $I_{\text{BA}} = I_{\text{BA}}^{\text{Het}}$  if Bob uses an intradyne detector such as in Fig. 6.246. The mutual information for those detection schemes between Bob and Alice are given by

$$I_{\text{BA}}^{\text{Hom}} = \frac{1}{2} \log_2 \left( \frac{V_B^+}{V_{A|B}^{\pm}} \right), \quad (6.322)$$

$$I_{\text{BA}}^{\text{Het}} = \frac{1}{2} \log_2 \left( \frac{V_B^+}{V_{A|B}^+} \right) + \frac{1}{2} \log_2 \left( \frac{V_B^-}{V_{A|B}^-} \right), \quad (6.323)$$

where we assume  $V_B^+$  represents the Bob variance associate to the  $x_A$  quadrature component sent by Alice,  $V_B^-$  represents the Bob variance associate to the  $p_A$  quadrature component sent by Alice. Moreover,  $V_{A|B}^{\pm}$  is the Alice conditional variance of Bob measurement, given by

$$V_{A|B}^{\pm} = V_B^{\pm} - \frac{|\langle S_A^{\pm} X_B^{\pm} \rangle|^2}{V_S^{\pm}}, \quad (6.324)$$

where  $V_S^{\pm}$  is the quadrature variance generate by Alice for the  $x_A$  and  $p_A$  quadrature components, and  $\langle S_A^{\pm} X_B^{\pm} \rangle$  is correlation coefficient between the Alice prepared state and the Bob measurement state. The coefficients in (6.324) are slightly different in the case of homodyne or intradyne detection.

## 1. Homodyne detection

$$S_A^+ = x_A = \left( \hat{X} - \hat{X}_v^+ \right), \quad (6.325)$$

$$S_A^- = p_A = \left( \hat{Y} - \hat{X}_v^- \right), \quad (6.326)$$

$$V_S^\pm = \langle (S_A^\pm)^2 \rangle = V_A N_0, \quad (6.327)$$

$$X_B^+ = \sqrt{T_{\text{ch}} \eta_{\text{det}}} (x_A + X_N^+), \quad (6.328)$$

$$X_B^- = \sqrt{T_{\text{ch}} \eta_{\text{det}}} (p_A + X_N^-), \quad (6.329)$$

$$V_B^\pm = \langle (X_B^\pm)^2 \rangle = T_{\text{ch}} \eta_{\text{det}} (V + \chi N_0), \quad (6.330)$$

$$\langle x_A^2 \rangle = \langle p_A^2 \rangle = (V_A + 1) N_0, \quad (6.331)$$

$$\langle (X_N^\pm)^2 \rangle = \chi N_0, \quad (6.332)$$

$$\langle X_A^\pm X_B^\pm \rangle = \sqrt{T_{\text{ch}} \eta_{\text{det}}} V_A N_0, \quad (6.333)$$

where  $X_v^\pm$  represents the quadrature operators for the vacuum state, and  $X_N^\pm$  represents noise added to the channel. In this case, the mutual information between Bob and Alice is

$$I_{\text{BA}}^{\text{Hom}} = \frac{1}{2} \log_2 \left( \frac{T_{\text{ch}} \eta_{\text{det}} (V_A + 1 + \chi) N_0}{T_{\text{ch}} \eta_{\text{det}} (1 + \chi) N_0} \right) \quad (6.334)$$

$$= \frac{1}{2} \log_2 \left( 1 + \frac{V_A}{1 + \chi} \right), \quad (6.335)$$

with  $\chi = \chi_{\text{ch}} + \chi_{\text{Hom}}/T_{\text{ch}}$ , and  $\chi_{\text{Hom}} = (1 - \eta_{\text{det}})/\eta_{\text{det}} + v_{\text{el}}/\eta_{\text{det}}$ .

2. Intradyne detection. In this case, the quantum signal entering at Bob detection system, passes through an extra beam-splitter before being measured by two homodyne detectors. This extra beam-splitter leads to a decrease in the detection efficiency of  $1/\sqrt{2}$  and an extra noise factor  $\hat{N}_B^\pm$ . This leads to

$$S_A^+ = x_A = \left( \hat{X} - \hat{X}_v^+ \right), \quad (6.336)$$

$$S_A^- = p_A = \left( \hat{Y} - \hat{X}_v^- \right), \quad (6.337)$$

$$V_S^\pm = \langle (S_A^\pm)^2 \rangle = V_A N_0, \quad (6.338)$$

$$X_B^+ = \sqrt{\frac{T_{\text{ch}} \eta_{\text{det}}}{2}} (x_A + X_N^+ + \hat{N}_B^+), \quad (6.339)$$

$$X_B^- = \sqrt{\frac{T_{\text{ch}} \eta_{\text{det}}}{2}} (p_A + X_N^- + \hat{N}_B^-), \quad (6.340)$$

$$V_B^\pm = \langle (X_B^\pm)^2 \rangle = \frac{T_{\text{ch}} \eta_{\text{det}}}{2} (V + \chi N_0), \quad (6.341)$$

$$\langle x_A^2 \rangle = \langle p_A^2 \rangle = (V_A + 1) N_0, \quad (6.342)$$

$$\langle (X_N^\pm + \hat{N}_B^\pm)^2 \rangle = \chi N_0, \quad (6.343)$$

$$\langle X_A^\pm X_B^\pm \rangle = \sqrt{\frac{T_{\text{ch}} \eta_{\text{det}}}{2}} V_A N_0. \quad (6.344)$$

The mutual information between Bob and Alice for the intradyne detection is

$$I_{BA}^{\text{Het}} = \log_2 \left( \frac{T_{\text{ch}}\eta_{\text{det}}(V_A + 1 + \chi)N_0}{T_{\text{ch}}\eta_{\text{det}}(1 + \chi)N_0} \right) \quad (6.345)$$

$$= \log_2 \left( 1 + \frac{V_A}{1 + \chi} \right), \quad (6.346)$$

with  $\chi = \chi_{\text{ch}} + \chi_{\text{det}}/T_{\text{ch}}$  with  $\chi_{\text{Het}} = (2 - \eta_{\text{det}})/\eta_{\text{det}} + 2v_{\text{el}}/\eta_{\text{det}}$ .

For the mutual information's between Bob and Eve the conditional variances are much more complex to obtain, and we will not derive the result here, we will present only the final result.

### 1. Homodyne detection

$$V_{B|E}^{\pm} = \eta_{\text{det}} \left[ \frac{1}{T_{\text{ch}}((V_A + 1)^{-1} + \chi_{\text{ch}})} + \chi_{\text{Hom}} \right] N_0. \quad (6.347)$$

This leads to mutual information between Bob and Eve equal to

$$I_{BE}^{\text{Hom}} = \frac{1}{2} \log_2 \left( \frac{V_B^{\pm}}{V_{B|E}^{\pm}} \right) \quad (6.348)$$

$$= \frac{1}{2} \log_2 \left( \frac{T_{\text{ch}}^2(V_A + 1 + \chi_{\text{ch}} + \chi_{\text{Hom}}/T_{\text{ch}})((V_A + 1)^{-1} + \chi_{\text{ch}})}{1 + T_{\text{ch}}\chi_{\text{Hom}}((V_A + 1)^{-1} + \chi_{\text{ch}})} \right) \quad (6.349)$$

### 2. Intradyne detection

$$V_{B|E}^{\pm} = \frac{\eta_{\text{det}}}{2} \left[ \frac{(V_A + 1)\chi_E + 1}{V_A + 1 + \chi_E} + \chi_{\text{Het}} \right] N_0, \quad (6.350)$$

where

$$\chi_E = \frac{T_{\text{ch}}(2 - \xi_A)^2}{(\sqrt{2 - 2T_{\text{ch}} + T_{\text{ch}}\xi_A} + \sqrt{\xi_A})^2} + 1. \quad (6.351)$$

This leads to an mutual information between Bob and Eve equal to

$$I_{BE}^{\text{Het}} = \log_2 \left( \frac{V_B^{\pm}}{V_{B|E}^{\pm}} \right) \quad (6.352)$$

$$= \log_2 \left( \frac{T_{\text{ch}}(V_A + 1 + \chi_{\text{ch}} + \chi_{\text{Het}}/T_{\text{ch}})(V_A + 1 + \chi_E)}{(V_A + 1)\chi_E + 1 + \chi_{\text{Het}}(V_A + 1 + \chi_E)} \right). \quad (6.353)$$

### Discrete-Modulated Coherent States

#### 6.13.2 Simulation Analysis

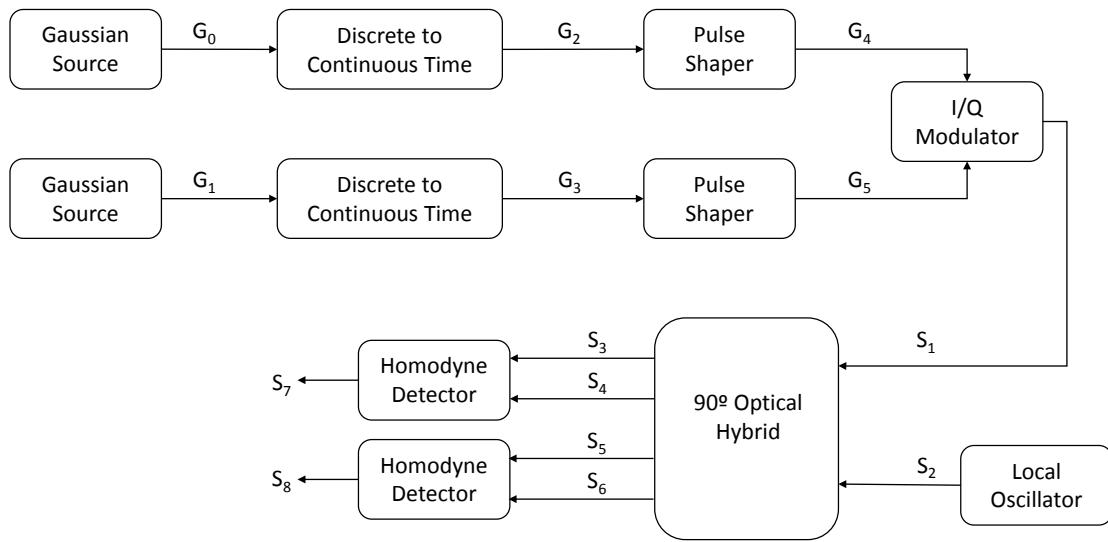


Figure 6.248: Overview of the simulated continuous variable intradyne QKD system.

#### 6.13.3 Experimental Analysis

#### 6.13.4 Comparative Analysis

## 6.14 Classical Multi-Party Computation

### 6.14.1 Introduction

Multi-Party Computation (MPC), also known as Secure Function Evaluation, allows two or more parties to correctly compute a function of their private inputs without exposure, i.e., without the input of one party being revealed to the other parties. In other terms, a generic function  $f$  receives as input a set  $\{a_1, a_2, \dots, a_n\}$  of arguments, where  $a_i$  is the input of the  $i$ -th party, and  $1 \leq i \leq n$ , and outputs a value  $c$ , which represents the result of the joint computation of  $f$ , as shown in Figure 6.249. The output of  $f$  is given by the following expression.

$$c = f(a_1, a_2, \dots, a_n) \quad (6.354)$$

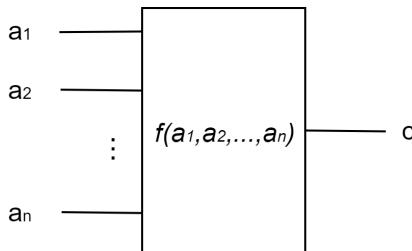


Figure 6.249: Multi-Party Computation Diagram

There are two main models in the literature for the analysis of the Multi-Party Computation problems. The ideal model, which consists in using a Trusted Third Party (TTP). The parties provide their input to the TTP, who will perform the computation of the function. The result is then sent to all the parties. This paradigm relies on the trustworthiness of the TTP because if it turns corrupt, it can supply the private input of one party to the others. This model is extensively used due to its easy implementation and protocols available which prevent the TTP from acting maliciously. The real model of MPC does not use a TTP. In this model, the parties agree on some protocol which will allow them to jointly compute the function. Different protocols are needed for different MPC models and different party behavior models. In 6.14.4 we will be analyzing different solutions to known MPC problems.

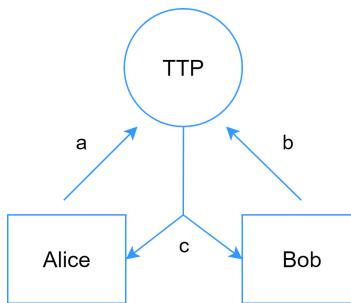


Figure 6.250: MPC using a Third Trusted Party

### 6.14.2 Two-Party Computation

Two-Party Computation (2PC) is a specific case of MPC, where a generic function  $f$  receives as input a set  $\{a, b\}$  of arguments, where  $a$  is the input from the first party and  $b$  is the input from the second, and outputs a value  $c$ , as shown in Figure 6.251. The output of  $f$  is given by the following expression.

$$c = f(a, b) \quad (6.355)$$

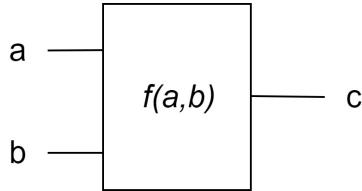


Figure 6.251: Two-Party Computation Diagram

### 6.14.3 Party Behavior Models

There are three main models to describe the behavior of a party. The honest model, where the party follows the protocol and respects the privacy of other parties. A semi honest model, where the party follows the protocol but also tries to gain additional information other than the result. The corrupt model, where the party neither follows the protocol nor respects the privacy of other parties.

#### 6.14.4 MPC Problems and Solutions

In this subsection, we will be analyzing different solutions to known MPC problems without the presence of a TTP. We will present a problem and then analyze various solution.

##### The Millionaires' Problem

Consider 2 parties, Alice and Bob, with inputs  $a$  and  $b$  respectively. The output of this function is 1 when  $a < b$  and 0 when  $a \geq b$ . In other terms,

$$f(a, b) = \begin{cases} 0, & \text{if } a \geq b \\ 1, & \text{if } a < b \end{cases}$$

**Yao's Solution** This method was proposed by Andrew C. Yao in 1982 [1]. Let  $E_a$  be Alice's public key,  $E_a(x)$  the process of encrypting input  $x$  by performing a bitwise XOR between  $x$  and Alice's public key and  $D_a(x)$  the process of decrypting input  $x$ . For this example, we will assume the following values:  $a = 7$ ,  $b = 3$ ,  $N = 16$ ,  $x = 39226$ ,  $p = 211$  and  $E_a = 24698$ .

1. Bob picks a random N-bit integer,  $x$ , and computes privately the value of  $E_a(x)$ ; calls the result  $k$ .

$$k = E_a(x) = x \oplus E_a = 63808 \quad (6.356)$$

2. Bob sends Alice the number  $k - b + 1$

$$k - b + 1 = 63806 \quad (6.357)$$

3. Alice computes privately the values of  $Y_u = D_a(k - b + u)$  for  $u = 1, 2, \dots, 10$

$$Y_u = [39236, 39237, 39226, 39227, 39224, 39225, 39230, 39231, 39228, 39229]$$

4. Alice generates a random prime  $p$  of  $N/2$  bits, and computes  $Z_u = Y_u \bmod p$

$$Z_u = [201, 202, 191, 192, 189, 190, 195, 196, 193, 194]$$

5. Alice sends the prime  $p$  and the following 10 numbers to Bob:  $Z_1, Z_2, \dots, Z_a$  followed by  $Z_{a+1} + 1, \dots, Z_{10} + 1$

$$M = [201, 202, 191, 192, 189, 190, 195, 197, 194, 195]$$

Note that only the elements of  $Z_u$  and  $M$  with indexes 7, 8, 9 and 10 are different, since  $a = 7$

6. Bob looks at the  $b$ -th number sent by Alice, and decides that  $a \geq b$  if it is equal to  $x \bmod p$ , and  $a < b$  otherwise. In other terms,

$$f(a, b) = \begin{cases} 0, & \text{if } M_b = x \bmod p \\ 1, & \text{if } M_b \neq x \bmod p \end{cases}$$

Since  $M_b = x \bmod p = 191$ , we have  $f(a, b) = 0$ , which means that  $a \geq b$ .

### 1-2 Oblivious Transfer

Consider 2 parties, Alice and Bob. Alice has a set of messages  $A = \{m_0, m_1, m_2, \dots, m_{n-1}\}$  and Bob has an index  $b$ . The output of this function should be  $m_b$ , i.e., the message from Alice's set of index  $b$ . Bob should not gain any more information other than  $m_b$  and Alice should not know the value of  $b$ .

$$f(A, b) = A_b \quad (6.358)$$

«««< HEAD

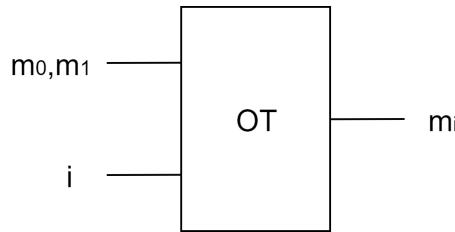


Figure 6.252: Oblivious Transfer Diagram

### Average Value

Consider 2 parties, Alice and Bob, with inputs  $a$  and  $b$  respectively. The output of this function should be the average value of  $a$  and  $b$ . In other terms,

$$f(a, b) = \frac{a + b}{2} \quad (6.359)$$

### 6.14.5 Garbled Circuit Protocol

Introduced in 1986 by Andrew Yao, the Garbled Circuit protocol (GC) addresses the case of Two-Party Computation (2PC), without the presence of a trusted third party. GC allows a secure evaluation of a function given as a Boolean circuit that is represented as a series of logic gates. The circuit is known to both parties.

### 6.14.6 Hardware Description Languages

Contrary to Programming Languages such as C or C++, which are used to specify a set of instructions to a computer, Hardware Description Languages (HDL) are computer languages used to describe the structure and behavior of digital logic circuits. They allow for the synthesis of HDL description code into a netlist (specification of physical electronic components, such as AND gates or NOT gates, and how they are connected together).

### 6.14.7 TinyGarble

TinyGarble is a GC framework that takes advantage of powerful logic synthesis techniques, provided by both HDL synthesis tools and TinyGarble's custom libraries, in order to improve

the overall efficiency of the GC protocol. It is possible to describe the circuit using High-Level Programming Languages (HLPL) such as C, although High-Level Synthesis (HLS) is required. HLS is performed by High-Level Synthesis tools, such as SPARK for the C language.

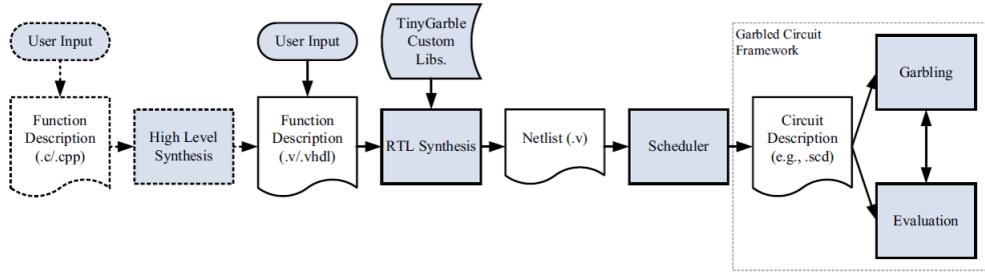


Figure 6.253: TinyGarble Flow Diagram

#### 6.14.8 ARM2GC

Although circuit description in HLPL is possible, it is not very efficient when compared to HDL circuit description. ARM2GC addresses this problem, significantly improving the performance of garbled circuits described in HLPL.

In the case of 2PC, ARM2GC's approach to GC is based on the ARM processor architecture and consists in providing a public parameter to function  $f$ , so that its output would be given by the following expression.

$$z = f(a, b, p) \quad (6.360)$$

, where  $p$  represents a public parameter, known to both parties.

In ARM2GC the Boolean circuit required to perform GC is that of a processor to which the compiled binary of the function is given as a public input ( $p = \text{compiled binary of the function}$ ). This optimization is performed by the SkipGate algorithm.

## References

- [1] Andrew C. Yao. "Protocols for Secure Computations". In: - (1982).

## 6.15 Quantum Multi-Party Computation

### 6.15.1 Our Approach

#### 1 out of 2 Oblivious Transfer

This section will provide the description of a 1-2 Oblivious Transfer (OT) based on the appliance of a Quantum Oblivious Key Distribution Protocol (QOKD). The 1-2 OT consists in a two party, Alice and Bob, communication protocol. Supposing that Alice has two messages  $\{m_1, m_0\}$  length  $s$ , Bob wants to know one of those in such a way that:

- Alice doesn't know Bob's choice, i.e. the protocol is oblivious;
- Bob doesn't get any information on the message he didn't choose, i.e. the protocol is concealing.

Considering the notation of a canonical quantum oblivious transfer protocol, let  $U = \{+, \times\}^n \times \{0, 1\}^n$ , where  $+, \times$  stand for the rectilinear and diagonal bases, with a correspondence previously agreed by both Bob and Alice. Physically this corresponds to the general algorithm of the protocol can be described as:

- Step 1:  
Alice picks a random uniformly chosen  $(a, g) \in U$ , and sends Bob photons  $i, 1 \leq i \leq n$  with polarizations given by the bases  $a[i]$  and states  $g[i]$ .
- Step 2:  
Bob picks a random uniformly chosen  $b \in \{+, \times\}^n$ , measures photons  $i$  in basis  $b[i]$  and records the results, if a photon is detected, as  $h[i] \in \{0, 1\}$ . Bob then makes a bit commitment of all  $n$  pairs  $(b[i], h[i])$  to Alice.
- Step 3:  
Alice picks a random uniformly chosen subset  $R \subset \{1, 2, \dots, n\}$  and tests the commitment made by Bob at positions in  $R$ . If more  $\delta n$  (acceptance threshold) positions  $i \in R$  reveal  $a[i] = b[i]$  and  $g[i] \neq h[i]$  then Alice stops protocol; otherwise, the test result is accepted.
- Step 4:  
Alice announces the base  $a$ . Let  $T_0$  be the set of  $1 \leq i \leq n$  such that  $a[i] = b[i]$  and let  $T_1$  be the set of all  $1 \leq i \leq n$  such that  $a[i] \neq b[i]$ . Bob chooses  $I_0, I_1 \subset T_0 - R, T_1 - R$  and sends  $S_i = \{I_{1-i}, I_i\}$ , wishing to know  $m_i, i \in \{0, 1\}$ .
- Step 5:  
Alice defines two encryption keys  $K_0, K_1$  in such a way that  $K_i = g[I_i]$  for  $i = 0 \vee i = 1$ . Alice then cyphers both messages:  $m_{\text{coded}} = \{m_0 \oplus K_0, m_1 \oplus K_1\}$  and sends the result  $m$  to Bob.
- Step 6:  
Bob will then decode  $m$  using the values of his initially chosen basis:  $b[S_i]$  with  $i \in \{0, 1\}$ .

$\{0, 1\}$ , according to his preference.  $m_{\text{decoded}} = m_{\text{coded}} \oplus b [S_i]$ . The output of this process will be  $m_{\text{decoded}}$  that will have the correct message in the first or last  $s^{\text{th}}$  positions if he chose  $m_0$  or  $m_1$ , respectively.

It is intuitively clear that the above protocol performs correctly if both parties are honest [1]. The security of protocol depends, though on the honesty of both parties (and a potential eavesdropper) involved. The security of the protocol can be evaluated in terms of the amount of information received by any given participant. In order to formalize and proof the security of such a system for any case though one has to think of the proceedings of a hypothetically dishonest Bob and an eventual eavesdropper Eve.

- Step 1:  
Dishonest Bob has no advantage in being dishonest at this point.
- Step 2:  
Dishonest Eve transfers some information from this pulse into her quantum system and she uses that information to modify the residual state of the pulse which is sent to Bob.  
Dishonest Bob executes a coherent measurement on the pulse received in order to determine: whether or not he declares this pulse as detected and the bit that he commits to Alice.
- Step 4:  
Having learnt Alice's string of basis  $a$  dishonest Bob executes a first post-measurement of his choice and uses the outcome to compute the ordered pair  $S$ .
- Step 5:  
Using the information obtained in the previous step dishonest Bob makes a second post-test measurement and obtains the outcome  $\mathcal{J}_{Bob}$ . Eve measures her system and obtains the outcome  $\mathcal{J}_{Eve}$  [2].

From [2] one can concluded that a dishonest Bob following these proceedings learns nothing about  $m$ , the set of both original messages concatenated, in its full extended, either he passes or fails Alice's original verification. This protocol also compensates the errors in the quantum channel. It is also stated that security against Bob and tolerance against errors implies the security of the protocol against Eve [2].

### Comparison Protocol

The millionaire problem was originally a two-party secure computation problem, in which two millionaires, Alice and Bob, want to know which of them is richer without revealing their actual wealth. It is analogous to a more general problem whose goal is to compare two numbers  $a$  and  $b$ , without revealing any extra information on their values other than what can be inferred from the comparison result. Boudot proposed a protocol to solve said. However, Lo pointed out that the equality function cannot be securely evaluated with a two-party scenario. Therefore, some additional assumptions should be considered to reach the goal of private comparison, a quantum comparison protocol (QPC) always needs:

- An at least semi-honest Third Party (TP) is required to help the two parties (Alice and Bob) accomplish the comparison. A semi-honest TP is a party who always follows the procedure of the protocol recording all of intermediate computations and despite not being corrupted by an outside eavesdropper, TP might try to steal the information from the record. TP will know the positions of different bit value in the compared information, but it will not be able to know the actual bit value of the information.
- All outsiders and the two players should only know the result of the comparison, but not the different positions of the information.
- To guarantee the security of private information, it is better to compare several bits instead of one bit at a time.

Since Yao's initial proposal, several QKD protocols using Einsteinâ€"Podolskyâ€"Rosen (EPR) pairs have been proposed in previous work to achieve secret communication for two communicants. These QKD protocols attempt to use the correlation of EPR pairs to distribute a common shared key for two mutually trusted users. However, EPR pairs in the proposed QPC protocol are used to create two individual keys for each of two mutually suspicious users and at the same time allow a semi-honest third party to perform the comparison without knowing the secret content of their information. Therefore, a QKD protocol may not be able to directly solve the QPC problem.

## References

- [1] Andrew Chi-Chih Yao. "Security of quantum protocols against coherent measurements". In: *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing - STOC '95*. ACM Press, 1995. DOI: [10.1145/225058.225085](https://doi.org/10.1145/225058.225085). URL: <https://doi.org/10.1145/225058.225085>.
- [2] Dominic Mayers. "On the Security of the Quantum Oblivious Transfer and Key Distribution Protocols". In: *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '95. London, UK, UK: Springer-Verlag, 1995, pp. 124–135. ISBN: 3-540-60221-6. URL: <http://dl.acm.org/citation.cfm?id=646760.705993>.

## 6.16 Secure Multi-Party Computation

<b>Students Name</b>	:	Mariana Ramos (02/05/2018 - 21/05/2018)
<b>Goal</b>	:	Description of the problem of Secure Multi-Party Computation.
<b>Directory</b>	:	

### 6.16.1 Problem definition

In *Secure Multi-Party Computation* (SMC) the function to be computed is defined as  $(y_1, y_2, \dots, y_N) = f(x_1, x_2, \dots, x_N)$  and each party  $i$  only knows its  $x_i$  and  $y_i$  being unknown any information about the input or output of the other parties. [1].

Thus, there are two ways of solving the problem of SMC:

1. The result can be computed in private using a trusted party involved who has the trust from all the parties. This is a simple solution for the current problem, however, it requires a trusted party.
2. The result can be computed without a trusted party involved using a SMC protocol. We are going to consider the *Garbled Circuit Protocol* (GCP), which will have our best attention in this chapter.

Here, we will only focus on a two-party computation ( $N = 2$ ).

### 6.16.2 Secure Two-Party Computation

The *Garbled Circuit Protocol* (GCP) consists in a transformation of a function into a boolean circuit. The function  $f_c$  is the output of a logical boolean circuit which has a function  $f$  as input.  $f_c$  is the logical function to be computed and it inputs a garbled circuit. From the garbled circuit Alice will generate a garbled version of the input function  $f_c$ , so-called  $F$ , as well as an encryption key  $e$  and a decryption key  $d$ . After that, Alice's input parameter ( $x_1$ ) and Bob's input parameter ( $x_2$ ) are both encrypted using the key  $e$  by the encoder. Next, the garbled version of the input function to be computed ( $F$ ) and the garbled version of Alice and Bob's input parameters ( $X_1$  and  $X_2$  respectively) enter in the following block, Evaluator, where Bob is responsible for evaluate the circuit. Finally, the decoder receive the result from evaluation  $Y$  and the decryption key  $d$ . This way, Bob is able to decode the information he wants, obtaining the value  $y = f(x_1, x_2)$  without know anything about Alice's input parameter  $x_1$ .

Note that Alice should also not know anything about Bob's input parameter, which leads to the need of use *Oblivious Transfer* in order to Alice send to Bob the garbled version of his input parameter.

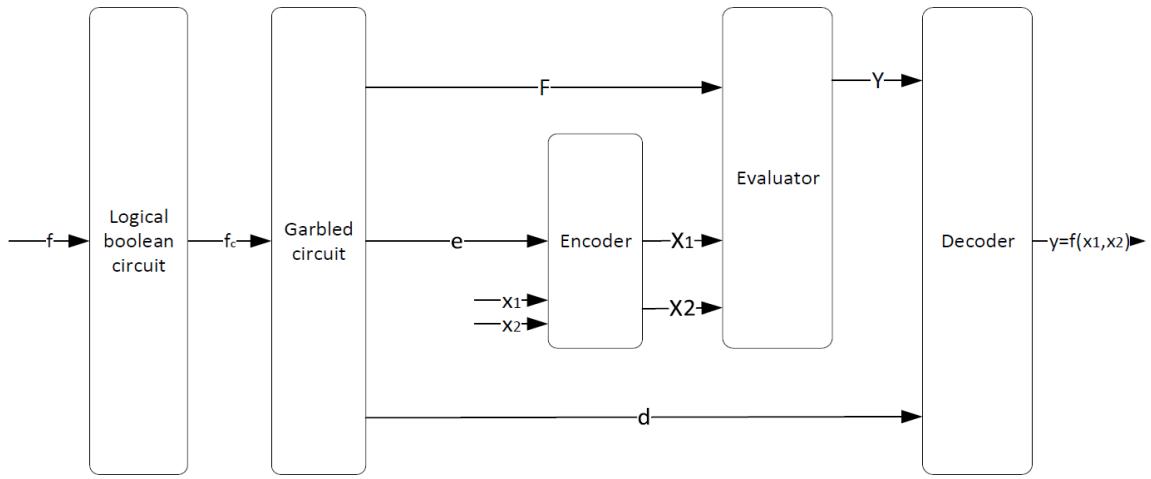


Figure 6.254: Garbled Circuit. Function to be computed  $y = f(x_1, x_2)$ .

Figure 6.254 shows the diagram of *garbled circuit protocol*. Legend of the figure is presented in the table below.

$x_1$	Alice input parameter
$x_2$	Bob input parameter
$f$	Function to compute
$F$	Garbled circuit of $f$
$X_1$	Alice garbled input
$X_2$	Bob garbled input
$e$	Encoding key
$d$	Decoding key
$Y$	Garbled output

Table 6.36: Legend of figure 6.254.

### Example of a two-party computation

Lets assume:

$$y = f(x_1, x_2) = x_1 \wedge x_2, \quad (6.361)$$

where  $x_1$  is Alice's input bit and  $x_2$  is Bob's input bit.

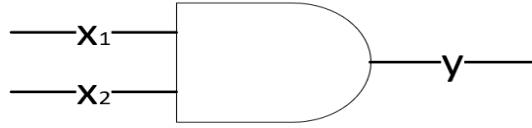


Figure 6.255: AND Gate which mimics the function to be computed.

Figure 6.255 shows the logical boolean circuit representation of the function to be computed. In this particular case there are two parties, each with an input parameter  $x_i$ , where  $i$  can be 1 (Alice) or 2 (Bob), and  $x_i$  can take values 0 or 1. This way, Alice will play the role of *garbled circuit generator* so she will generate a garbled AND gate and send it to Bob. On the other hand, Bob will play the role of *evaluator* and he will be able to evaluate the garbled circuit using Alice and its own garbled inputs.

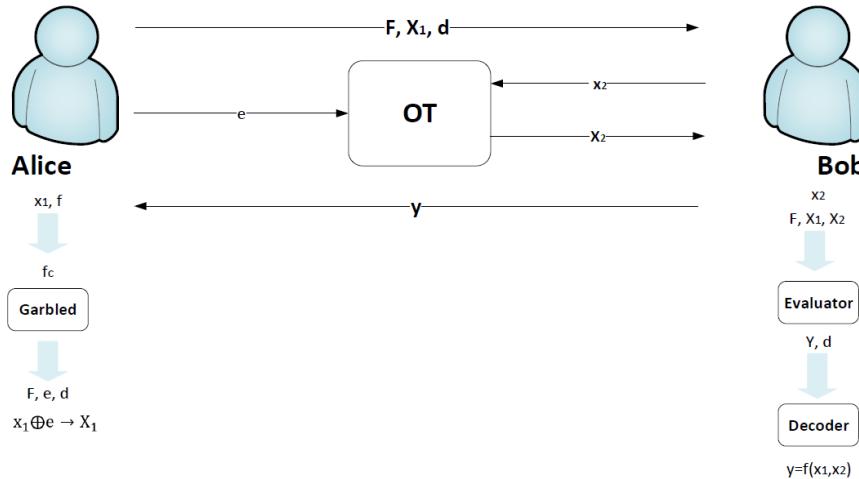


Figure 6.256: Garbled Circuit Protocol with OT.

In order to garble the Bob's input  $x_2$ , Alice and Bob use the OT protocol [1]. At the end Bob has a full garbled version of his input,  $X_2$ , without Bob learning the encoding key,  $e$ , and without Alice learning anything about Bob's input parameter,  $x_2$ .

$x_1$	$x_2$	Output $x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Table 6.37: Initial truth table.

Table above shows the initial boolean gate truth table. First, Alice encrypt its input data using the encoding key  $e$ ,  $X_1 = x_1 \text{ XOR } e$ . Therefore, she matches for each  $e(x_i) = X_i$ , if  $x_i = 0$   $X_i = W_i^0$ , or if  $x_i = 1$   $X_i = W_i^1$ , being  $i \in \{1, 2\}$ .

$W_1^0$ , corresponds to the event that  $x_1 = 0$ .

$W_1^1$ , corresponds to the event that  $x_1 = 1$ .

$W_2^0$ , corresponds to the event that  $x_2 = 0$ .

$W_2^1$ , corresponds to the event that  $x_2 = 1$ .

The table above presents a summary of the four possible labels. In order to encrypt the output column for each possible scenario, Alice computes an Hash Function depending on her gabled label  $W_1^i$ , Bob's gabled label  $W_2^i$  and the correspondent output  $f$ . Lets assume she computed the Hash Functions and the results are the following:

Hash Function	Hash Function Output
$H(W_1^0, W_2^0, 0)$	0 0 0
$H(W_1^0, W_2^1, 0)$	0 1 0
$H(W_1^1, W_2^0, 0)$	1 0 0
$H(W_1^1, W_2^1, 1)$	1 1 1

Table 6.38: Hash function computed and its result.

Before forwarding these values to Bob, Alice also performs an encryption over these results using a key,  $d = 111$ , which she'll also send to Bob. Using this encryption key the values of  $\text{Enc}(H(W_1^i, W_2^i, f)) = H(W_1^i, W_2^i, f) \oplus d$  are the following:

$\text{Enc}(H(W_A^0, W_B^0, 0))$	1 1 1
$\text{Enc}(H(W_A^0, W_B^1, 0))$	1 0 1
$\text{Enc}(H(W_A^1, W_B^0, 0))$	0 1 1
$\text{Enc}(H(W_A^1, W_B^1, 1))$	0 0 0

Table 6.39: Garbled Output.

The table above shows the encryption function applied in each possible output of the initial boolean gate truth table. As one can see in the table, the two relevant labels are put in a derivation function  $H$  in order to derive an encryption key which is used to encrypt the output of the boolean circuit. Furthermore, since the *garbled gate* present the ciphered outputs in a random order, a permutation of the results should be done, being the permuted garbled outputs represented in the table below [2].

Enc( $H(W_A^0, W_B^1, 0)$ )	1 0 1
Enc( $H(W_A^1, W_B^1, 1)$ )	0 0 0
Enc( $H(W_A^0, W_B^0, 0)$ )	1 1 1
Enc( $H(W_A^1, W_B^0, 0)$ )	0 1 1

Table 6.40: Permutated Garbled Output

$W_1^i$	$W_2^i$	$x_1$	$x_2$	Enc( $H(W_1^i, W_2^i, f)$ )	f
0	1	0	0	1 1 1	0
0	0	0	1	1 0 1	0
1	1	1	0	0 1 1	0
1	0	1	1	0 0 0	1

Table 6.41: Summary of the labels, parties inputs and Garbled Circuit.

In table 6.41 is presented a summary of the labels, the parties inputs and the garbled circuit that was built.

Being  $X_1$  the garbled version of the Alice's input  $x_1$  and  $F$  the garbled circuit of the function  $f$  to be computed, Alice now sends to Bob these two as well as the decryption key  $d$ . Having taken into account these parameters, Bob also needs the garbled version of his own input parameter  $x_2$  to be able to evaluate the garbled circuit. At the time Alice build the garbled values for Bob, she does not know anything about his actual input parameter but she built the garbled circuit taking into account all possible values. On the other hand, Bob knows his own input parameter but he is not able to determine the keys which were generated for his input value.

With the information sent by Alice, Bob also needs the garbled version of his input. In this case, lets assume that Bob's input parameter is  $x_2 = 1$ . They must follow the *oblivious transfer protocol* in order to Bob gets his garbled version of his input  $X_2$ . This way, Alice sends to Bob using OT  $W_2^1 = 0$ . With both inputs garbled versions, Bob can perform a XOR operation between the equivalent result of  $\text{Enc}(H(W_1^i, W_2^i, f))$  and the decryption key  $d$ :

$$\text{Enc}(H(W_1^i, W_2^i, f)) \oplus d = 010. \quad (6.362)$$

Since Bob knows  $X_1 = W_1^i$ ,  $X_2 = W_2^i$  and the result of  $\text{Enc}(H(W_1^i, W_2^i, f))$ , he must introduce this information in the Hash Function in order to get  $f$ . This function only outputs the correct result of  $f$  for the garbled values that Bob knows. Thus, he will discover the output value  $f = 0$  without learning anything about the input parameter introduced by Alice. Nevertheless, since an oblivious transfer protocol was used to send to Bob his garbled version of his input parameter, Alice also knows anything about the input parameter introduced by Bob.

## References

- [1] Frederic Naumann. "Garbled Circuits". In: *Network* 71 (2016).
- [2] Sophia Yakoubov. "A Gentle Introduction to Yaos Garbled Circuits". In: () .



## 7.1 ADC

<b>Header File</b>	:	adc_*.h
<b>Source File</b>	:	adc_*.cpp
<b>Version</b>	:	20180423 (Celestino Martins)

This super block block simulates an analog-to-digital converter (ADC), including signal resample and quantization. It receives two real input signal and outputs two real signal with the sampling rate defined by ADC sampling rate, which is externally configured using the resample function, and quantized signal into a given discrete values.

### Input Parameters

Parameter	Unity	Type	Values	Default
samplingPeriod	–	double	any	—
rFactor	–	double	any	1
resolution	bits	double	any	<i>inf</i>
maxValue	volts	double	any	1.5
minValue	volts	double	any	–1.5

Table 7.1: ADC input parameters

### Methods

```

ADC(vector<Signal *> &InputSig, vector<Signal *> &OutputSig);

//void setResampleSamplingPeriod(double sPeriod) B1.setSamplingPeriod(sPeriod);
B2.setSamplingPeriod(sPeriod); ;

void setResampleOutRateFactor(double OUTsRate) B01.setOutRateFactor(OUTsRate);
B02.setOutRateFactor(OUTsRate);

void setQuantizerSamplingPeriod(double sPeriod) B03.setSamplingPeriod(sPeriod);
B04.setSamplingPeriod(sPeriod);

void setSamplingPeriod(double sPeriod) B03.setSamplingPeriod(sPeriod); ;

void setQuantizerResolution(double nbits) B03.setResolution(nbits);
B04.setResolution(nbits);

void setQuantizer.MaxValue(double maxvalue) B03.setMaxValue(maxvalue);
B04.setMaxValue(maxvalue);

void setQuantizer.MinValue(double minvalue) B03.setMinValue(minvalue);
B04.setMinValue(minvalue);

```

### Functional description

This super block is composed of two blocks, resample and quantizer. It can perform the signal resample according to the defined input parameter *rFactor* and signal quantization according to the defined input parameter *nBits*.

**Input Signals**

**Number:** 2

**Output Signals**

**Number:** 2

**Type:** Electrical complex signal

**Examples**

**Sugestions for future improvement**

## 7.2 Add

<b>Header File</b>	:	add.h
<b>Source File</b>	:	add.cpp
<b>Version</b>	:	20180118

### Input Parameters

This block takes no parameters.

### Functional Description

This block accepts two signals and outputs one signal built from a sum of the two inputs. The input and output signals must be of the same type, if this is not the case the block returns an error.

### Input Signals

**Number:** 2

**Type:** Real, Complex or Complex\_XY signal (ContinuousTimeContinuousAmplitude)

### Output Signals

**Number:** 1

**Type:** Real, Complex or Complex\_XY signal (ContinuousTimeContinuousAmplitude)

### 7.3 Balanced Beam Splitter

<b>Header File</b>	:	balanced_beam_splitter.h
<b>Source File</b>	:	balanced_beam_splitter.cpp
<b>Version</b>	:	20180124

#### Input Parameters

Name	Type	Default Value
Matrix	array <t_complex, 4>	$\left\{ \left\{ \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}} \right\} \right\}$
Mode	double	0

#### Functional Description

The structure of the beam splitter can be controlled with the parameter mode.

When **Mode = 0** the beam splitter will have one input port and two output ports - **1x2 Beam Splitter**. If Mode has a value different than 0, the splitter will have two input ports and two output ports - **2x2 Beam Splitter**.

Considering the first case, the matrix representing a 2x2 Beam Splitter can be summarized in the following way,

$$M_{BS} = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (7.1)$$

The relation between the values of the input ports and the values of the output ports can be established in the following way

$$\begin{bmatrix} A' \\ B' \end{bmatrix} = M_{BS} \begin{bmatrix} A \\ B \end{bmatrix} \quad (7.2)$$

Where, A and B represent the inputs and A' and B' represent the outputs of the Beam Splitter.

#### Input Signals

**Number:** 1 or 2

**Type:** Complex

#### Output Signals

**Number:** 2

**Type:** Complex

## 7.4 Bit Error Rate

<b>Header File</b>	:	bit_error_rate_*.h
<b>Source File</b>	:	bit_error_rate_*.cpp
<b>Version</b>	:	20171810 (Daniel Pereira)
	:	20181424 (Mariana Ramos)

### Input Parameters

Name	Type	Default Value
alpha	double	0.05
m	integer	0
lMinorant	double	$1 \times 10^{-10}$

### Methods

- BitErrorRate(vector<Signal \* > &InputSig, vector<Signal \* > &OutputSig) :Block(InputSig,OutputSig){};
- void initialize(void);
- bool runBlock(void);
- void setConfidence(double P) { alpha = 1-P; }
- void setMidReportSize(int M) { m = M; }
- void setLowestMinorant(double lMinorant) { lowestMinorant=lMinorant; }

### Input Signals

**Number:** 2

**Type:** Binary (DiscreteTimeDiscreteAmplitude)

### Output Signals

**Number:** 1

**Type:** Binary (DiscreteTimeDiscreteAmplitude)

### Functional Description

This block accepts two binary strings and outputs a binary string, outputting a 1 if the two input samples are equal to each other and 0 if not. This block also outputs .txt files with a

report of the estimated Bit Error Rate (BER),  $\widehat{\text{BER}}$  as well as the estimated confidence bounds for a given probability  $\alpha$ .

The block allows for mid-reports to be generated, the number of bits between reports is customizable, if it is set to 0 then the block will only output the final report. In version 20180424 this block can operate mid-reports using a CUMULATIVE mode, in which the BER is calculated in a cumulative way taking into account all received bits, coincidences and errors, or in a RESET mode, in which at each  $\mathbf{m}$  bits the number of received bits and coincidence bits is reset for the BER calculation.

### Theoretical Description

The  $\widehat{\text{BER}}$  is obtained by counting both the total number received bits,  $N_T$ , and the number of coincidences,  $K$ , and calculating their relative ratio:

$$\widehat{\text{BER}} = 1 - \frac{K}{N_T}. \quad (7.3)$$

The upper and lower bounds,  $\text{BER}_{\text{UB}}$  and  $\text{BER}_{\text{LB}}$  respectively, are calculated using the Clopper-Pearson confidence interval, which returns the following simplified expression for  $N_T > 40$  [1]:

$$\text{BER}_{\text{UB}} = \widehat{\text{BER}} + \frac{1}{\sqrt{N_T}} z_{\alpha/2} \sqrt{\widehat{\text{BER}}(1 - \widehat{\text{BER}})} + \frac{1}{3N_T} \left[ 2 \left( \frac{1}{2} - \widehat{\text{BER}} \right) z_{\alpha/2}^2 + (2 - \widehat{\text{BER}}) \right] \quad (7.4)$$

$$\text{BER}_{\text{LB}} = \widehat{\text{BER}} - \frac{1}{\sqrt{N_T}} z_{\alpha/2} \sqrt{\widehat{\text{BER}}(1 - \widehat{\text{BER}})} + \frac{1}{3N_T} \left[ 2 \left( \frac{1}{2} - \widehat{\text{BER}} \right) z_{\alpha/2}^2 - (1 + \widehat{\text{BER}}) \right], \quad (7.5)$$

where  $z_{\alpha/2}$  is the  $100(1 - \frac{\alpha}{2})$ th percentile of a standard normal distribution.

### Version 20181424

Version 20181424 allows the user to choose the type of middle reports he wants. So, the input parameter `mideRepType` can have the value *Cumulative*, where the BER estimation is done by taking into account all samples acquired in a cumulative way, or *Reset*, where the BER estimation is done by taking into account only the number of samples set as  $m$  in each middle report.

- **Input Parameters**

Name	Type	Default Value
<code>midRepType</code>	MidReportType	Cumulative

- **Methods**

- `void setMidReportType(MidReportType mrt) { midRepType = mrt; };`

## References

- [1] Álvaro J Almeida et al. “Continuous Control of Random Polarization Rotations for Quantum Communications”. In: *Journal of Lightwave Technology* 34.16 (2016), pp. 3914–3922.

## 7.5 Binary Source

<b>Header File</b>	:	binary_source.h
<b>Source File</b>	:	binary_source.cpp

This block generates a sequence of binary values (1 or 0) and it can work in four different modes:

- |                 |                             |
|-----------------|-----------------------------|
| 1. Random       | 3. DeterministicCyclic      |
| 2. PseudoRandom | 4. DeterministicAppendZeros |

This blocks doesn't accept any input signal. It produces any number of output signals.

### Input Parameters

Parameter	Type	Values	Default
mode	string	Random, PseudoRandom, DeterministicCyclic, DeterministicAppendZeros	PseudoRandom
probabilityOfZero	real	$\in [0,1]$	0.5
patternLength	int	Any natural number	7
bitStream	string	sequence of 0's and 1's	0100011101010101
numberOfBits	long int	any	-1
bitPeriod	double	any	1.0/100e9

Table 7.2: Binary source input parameters

### Methods

```
BinarySource(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig){};
```

```
void initialize(void);
bool runBlock(void);
void setMode(BinarySourceMode m) BinarySourceMode const getMode(void)
void setProbabilityOfZero(double pZero)
double const getProbabilityOfZero(void)
void setBitStream(string bStream)
```

```

string const getBitStream(void)

void setNumberOfBits(long int nOfBits)

long int const getNumberOfBits(void)

void setPatternLength(int pLength)

int const getPatternLength(void)

void setBitPeriod(double bPeriod)

double const getBitPeriod(void)

```

### Functional description

The *mode* parameter allows the user to select between one of the four operation modes of the binary source.

**Random Mode** Generates a 0 with probability *probabilityOfZero* and a 1 with probability  $1 - \text{probabilityOfZero}$ .

**Pseudorandom Mode** Generates a pseudorandom sequence with period  $2^{\text{patternLength}} - 1$ .

**DeterministicCyclic Mode** Generates the sequence of 0's and 1's specified by *bitStream* and then repeats it.

**DeterministicAppendZeros Mode** Generates the sequence of 0's and 1's specified by *bitStream* and then it fills the rest of the buffer space with zeros.

### Input Signals

**Number:** 0

**Type:** Binary (DiscreteTimeDiscreteAmplitude)

### Output Signals

**Number:** 1 or more

**Type:** Binary (DiscreteTimeDiscreteAmplitude)

## Examples

### Random Mode

**PseudoRandom Mode** As an example consider a pseudorandom sequence with *patternLength*=3 which contains a total of 7 ( $2^3 - 1$ ) bits. In this sequence it is possible to find every combination of 0's and 1's that compose a 3 bit long subsequence with the exception of 000. For this example the possible subsequences are 010, 110, 101, 100, 111, 001 and 100 (they appear in figure 7.1 numbered in this order). Some of these require wrap.

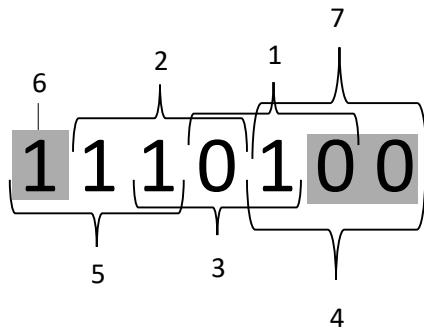


Figure 7.1: Example of a pseudorandom sequence with a pattern length equal to 3.

**DeterministicCyclic Mode** As an example take the *bit stream* '0100011101010101'. The generated binary signal is displayed in.

**DeterministicAppendZeros Mode** Take as an example the *bit stream* '0100011101010101'. The generated binary signal is displayed in 7.2.

### Sugestions for future improvement

Implement an input signal that can work as trigger.

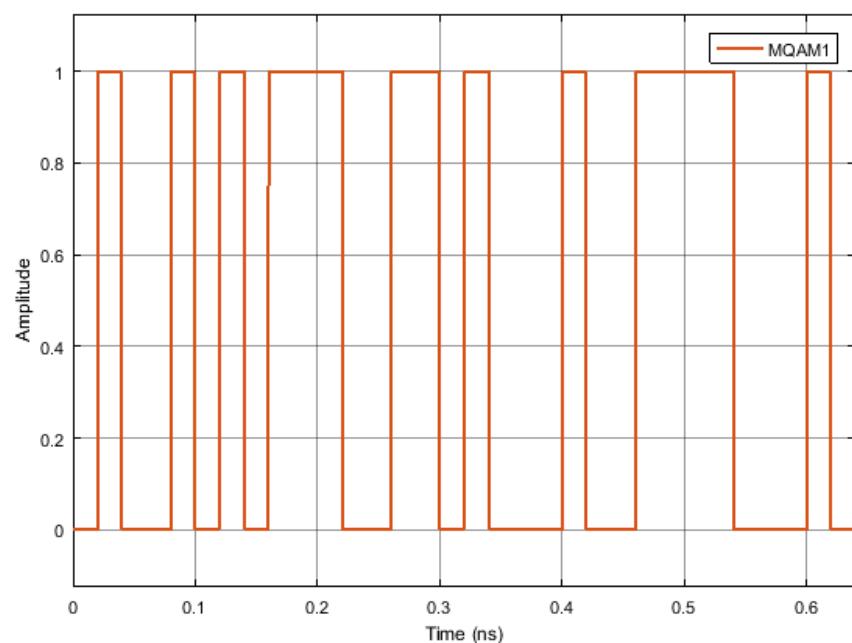


Figure 7.2: Binary signal generated by the block operating in the *Deterministic Append Zeros* mode with a binary sequence 01000...

## 7.6 Bit Decider

<b>Header File</b>	:	bit_decider.h
<b>Source File</b>	:	bit_decider.cpp
<b>Version</b>	:	20170818

### Input Parameters

Name	Type	Default Value
decisionLevel	double	0.0

### Functional Description

This block accepts one real discrete signal and outputs a binary string, outputting a 1 if the input sample is greater than the decision level and 0 if it is less or equal to the decision level.

### Input Signals

**Number:** 1

**Type:** Real signal (DiscreteTimeContinuousAmplitude)

### Output Signals

**Number:** 1

**Type:** Binary (DiscreteTimeDiscreteAmplitude)

## 7.7 Clock

<b>Header File</b>	:	clock.h
<b>Source File</b>	:	clock.cpp

This block doesn't accept any input signal. It outputs one signal that corresponds to a sequence of Dirac's delta functions with a user defined *period*.

### Input Parameters

Parameter	Type	Values	Default
period	double	any	0.0
samplingPeriod	double	any	0.0

Table 7.3: Binary source input parameters

### Methods

Clock()

```
Clock(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig)
```

```
void initialize(void)
```

```
bool runBlock(void)
```

```
void setClockPeriod(double per)
```

```
void setSamplingPeriod(double sPeriod)
```

### Functional description

**Input Signals****Number:** 0**Output Signals****Number:** 1

**Type:** Sequence of Dirac's delta functions.  
(TimeContinuousAmplitudeContinuousReal)

**Examples****Sugestions for future improvement**

## 7.8 Clock\_20171219

This block doesn't accept any input signal. It outputs one signal that corresponds to a sequence of Dirac's delta functions with a user defined *period*, *phase* and *sampling period*.

### Input Parameters

- period{ 0.0 };
- samplingPeriod{ 0.0 };
- phase {0.0};

### Methods

Clock()

Clock(vector<Signal \*> &InputSig, vector<Signal \*> &OutputSig) :Block(InputSig, OutputSig)

void initialize(void)

bool runBlock(void)

void setClockPeriod(double per) double getClockPeriod()

void setClockPhase(double pha) double getClockPhase()

void setSamplingPeriod(double sPeriod) double getSamplingPeriod()

### Functional description

#### Input Signals

**Number:** 0

#### Output Signals

**Number:** 1

**Type:** Sequence of Dirac's delta functions.  
(TimeContinuousAmplitudeContinuousReal)

#### Examples

#### Sugestions for future improvement

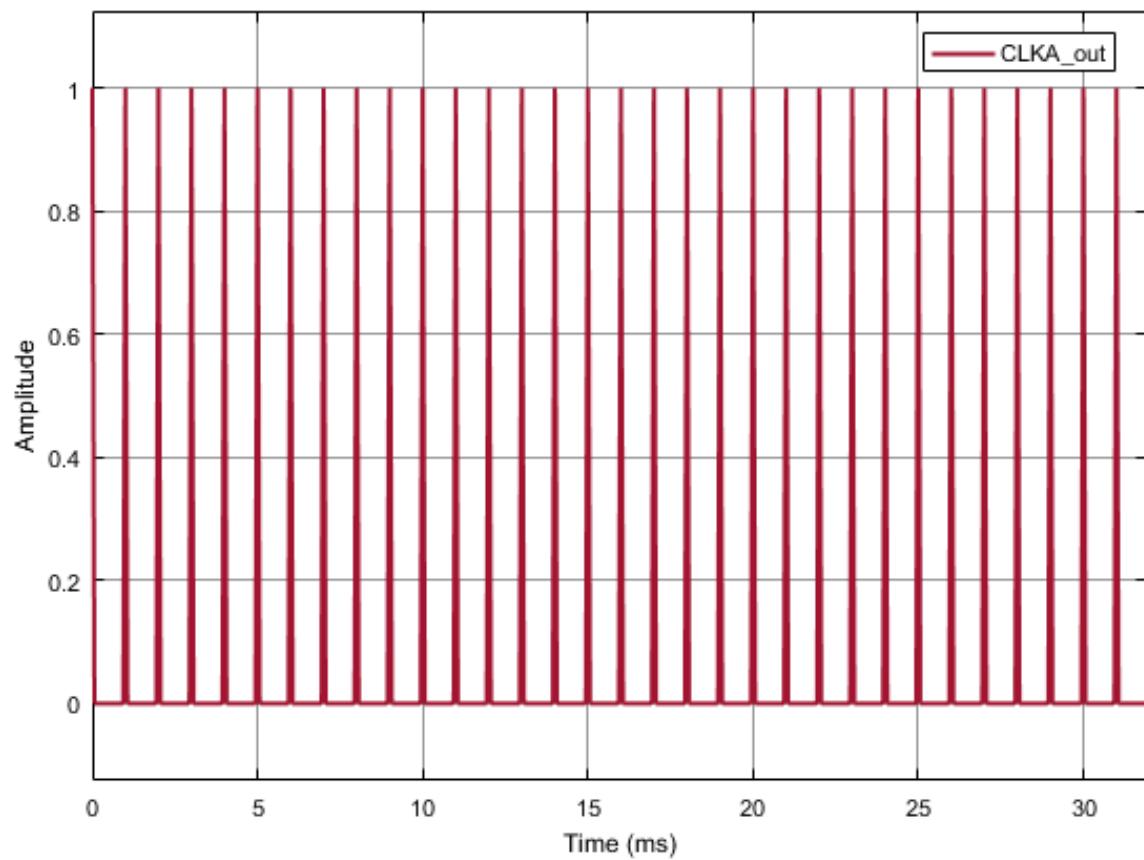


Figure 7.3: Example of the output signal of the clock without phase shift.

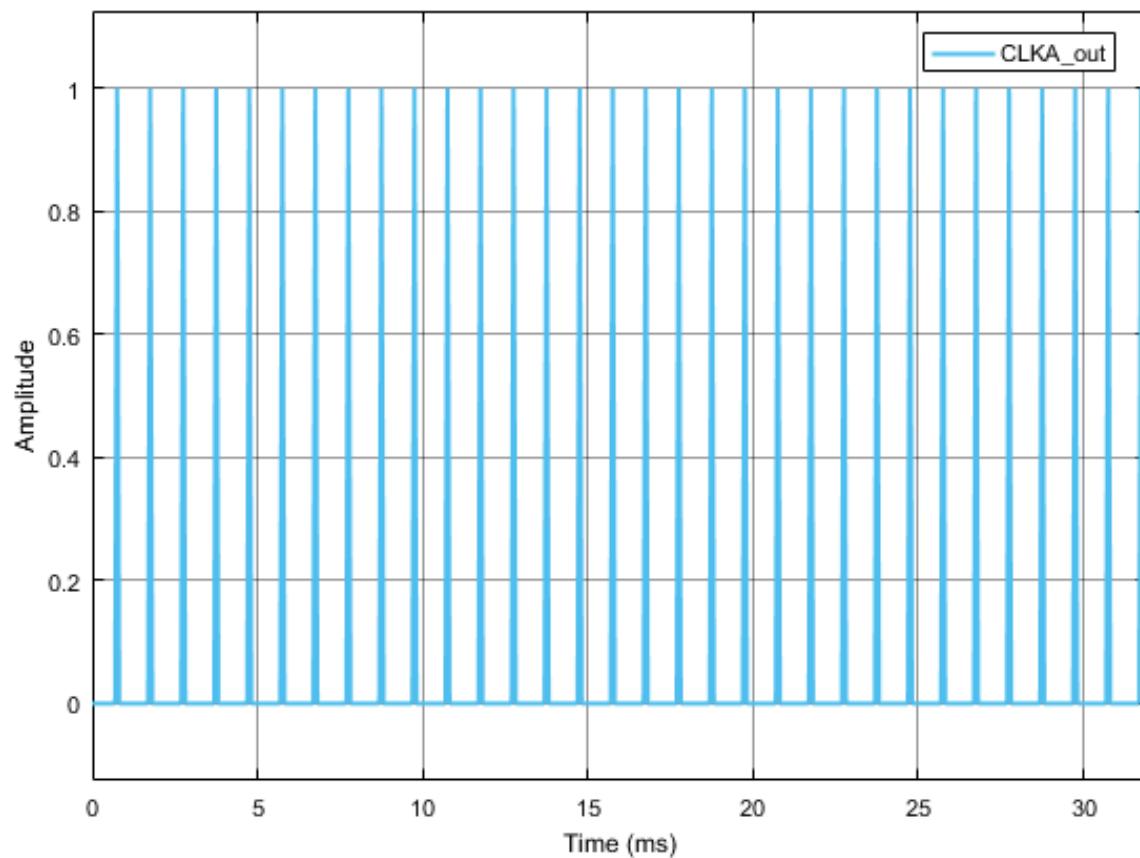


Figure 7.4: Example of the output signal of the clock with phase shift.

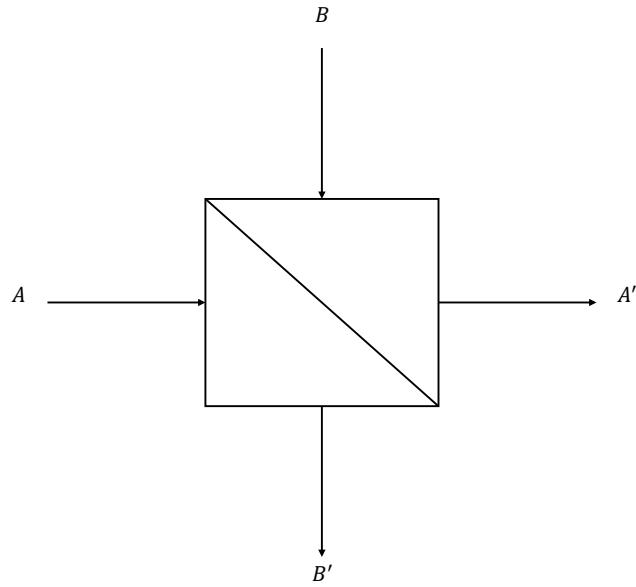


Figure 7.5: 2x2 coupler

## 7.9 Coupler 2 by 2

In general, the matrix representing 2x2 coupler can be summarized in the following way,

$$\begin{bmatrix} A' \\ B' \end{bmatrix} = \begin{bmatrix} T & iR \\ iR & T \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix} \quad (7.6)$$

Where, A and B represent inputs to the 2x2 coupler and A' and B' represent output of the 2x2 coupler. Parameters T and R represent transmitted and reflected part respectively which can be quantified in the following form,

$$T = \sqrt{1 - \eta_R} \quad (7.7)$$

$$R = \sqrt{\eta_R} \quad (7.8)$$

Where, value of the  $\sqrt{\eta_R}$  lies in the range of  $0 \leq \sqrt{\eta_R} \leq 1$ .

It is worth to mention that if we put  $\eta_R = 1/2$  then it leads to a special case of "Balanced Beam splitter" which equally distribute the input power into both output ports.

## 7.10 Carrier Phase Compensation

<b>Header File</b>	:	cpe_*.h
<b>Source File</b>	:	cpe_*.cpp
<b>Version</b>	:	20180423 (Celestino Martins)

This block performs the laser phase noise compensation using either Viterbi-Viterbi (VV) algorithm or blind phase search algorithm (BPS). For both cases, it receives one input complex signal and outputs one complex signal.

### Input Parameters For VV Algorithms

Parameter	Type	Values	Default
nTaps	int	any	25
methodType	string	VV	VV
samplingPeriod	double	any	0.0
symbolPeriod	double	any	0.0

Table 7.4: CPE input parameters

### Input Parameters For BPS Algorithms

Parameter	Type	Values	Default
nTaps	int	any	25
NtestPhase	int	any	32
methodType	string	BPS	VV
samplingPeriod	double	any	0.0
symbolPeriod	double	any	0.0

Table 7.5: CPE input parameters

### Methods

CPE();

```
CPE(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig){};
```

```
void initialize(void);
```

```
bool runBlock(void);
```

```
void setSamplingPeriod(double sPeriod) samplingPeriod = sPeriod;  
void setSymbolPeriod(double sPeriod) symbolPeriod = sPeriod;  
void setnTaps(double ntaps) nTaps = ntaps;  
double getnTaps() return nTaps;  
void setTestPhase(double nTphase) nTestPhase = nTphase;  
double getTestPhase() return nTestPhase;  
void setmethodType(string mType) methodType = mType;  
double getmethodType() return methodType;
```

### Functional description

This block can perform the carrier phase noise compensation originated by the laser source and local oscillator in coherent optical communication systems. For the sake of simplicity, in this simulation we have restricted all the phase noise at the transmitter side, in this case generated by the laser source, which is then compensated at the receiver side using DSP algorithms. In this simulation, the carrier phase noise compensation can be performed by applying either the well known Viterbi-Viterbi (VV) algorithm or blind phase search algorithm (BPS), by configuring the parameter *methodType*. The parameter *methodType* is defined as a string type and it can be configured as: i) When the parameter *methodType* is *VV* it is applied the VV algorithm; When the parameter *methodType* is *BPS* it is applied the BPS algorithm.

**Input Signals**

**Number:** 1

**Output Signals**

**Number:** 1

**Type:** Electrical complex signal

**Examples**

**Sugestions for future improvement**

## 7.11 Decoder

<b>Header File</b>	:	decoder.h
<b>Source File</b>	:	decoder.cpp

This block accepts a complex electrical signal and outputs a sequence of binary values (0's and 1's). Each point of the input signal corresponds to a pair of bits.

### Input Parameters

Parameter	Type	Values	Default
m	int	$\geq 4$	4
iqAmplitudes	vector<t_complex>	—	{ { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }

Table 7.6: Binary source input parameters

### Methods

Decoder()

```
Decoder(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig,
OutputSig)
```

```
void initialize(void)
```

```
bool runBlock(void)
```

```
void setM(int mValue)
```

```
void getM()
```

```
void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues)
```

```
vector<t_iqValues> getIqAmplitudes()
```

### Functional description

This block makes the correspondence between a complex electrical signal and pair of binary values using a predetermined constellation.

To do so it computes the distance in the complex plane between each value of the input signal and each value of the *iqAmplitudes* vector selecting only the shortest one. It then converts the point in the IQ plane to a pair of bits making the correspondence between the input signal and a pair of bits.

## Input Signals

**Number:** 1

**Type:** Electrical complex (TimeContinuousAmplitudeContinuousReal)

## Output Signals

**Number:** 1

**Type:** Binary

## Examples

As an example take an input signal with positive real and imaginary parts. It would correspond to the first point of the *iqAmplitudes* vector and therefore it would be associated to the pair of bits 00.

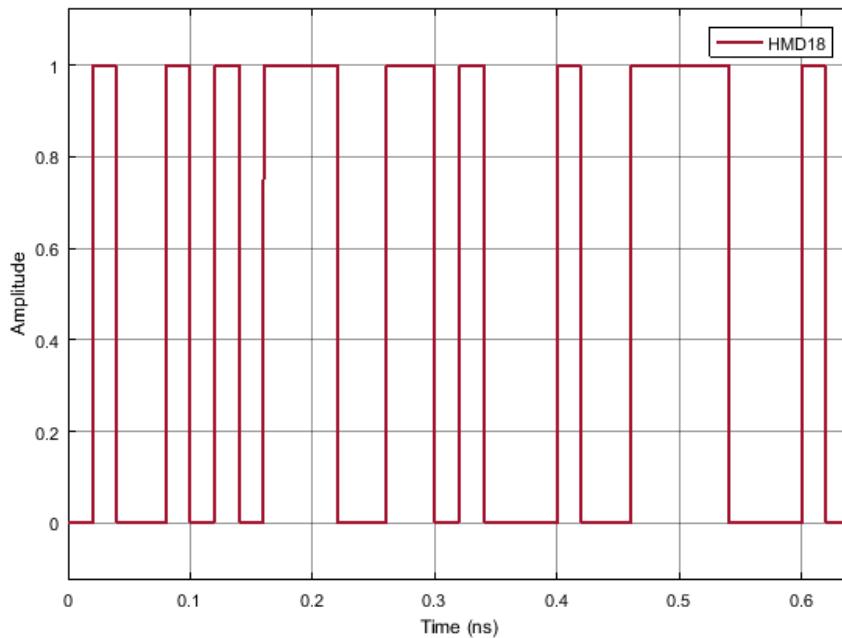


Figure 7.6: Example of the output signal of the decoder for a binary sequence 01. As expected it reproduces the initial bit stream

## Sugestions for future improvement

## 7.12 Discrete To Continuous Time

<b>Header File</b>	:	discrete_to_continuous_time.h
<b>Source File</b>	:	discrete_to_continuous_time.cpp

This block converts a signal discrete in time to a signal continuous in time. It accepts one input signal that is a sequence of 1's and -1's and it produces one output signal that is a sequence of Dirac delta functions.

### Input Parameters

Parameter	Type	Values	Default
numberOfSamplesPerSymbol	int	any	8

Table 7.7: Binary source input parameters

### Methods

```
DiscreteToContinuousTime(vector<Signal * > &inputSignals, vector<Signal * > &outputSignals) :Block(inputSignals, outputSignals){};

void initialize(void);

bool runBlock(void);

void setNumberOfSamplesPerSymbol(int nSamplesPerSymbol)

int const getNumberOfSamplesPerSymbol(void)
```

### Functional Description

This block reads the input signal buffer value, puts it in the output signal buffer and it fills the rest of the space available for that symbol with zeros. The space available in the buffer for each symbol is given by the parameter *numberOfSamplesPerSymbol*.

### Input Signals

**Number** : 1

**Type** : Sequence of 1's and -1's. (DiscreteTimeDiscreteAmplitude)

### Output Signals

**Number** : 1

**Type** : Sequence of Dirac delta functions (ContinuousTimeDiscreteAmplitude)

**Example**

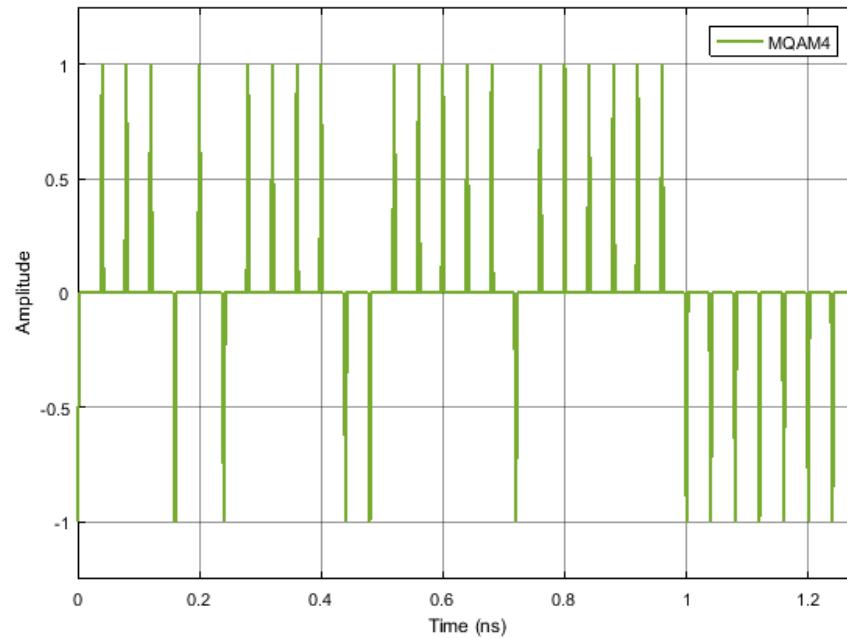


Figure 7.7: Example of the type of signal generated by this block for a binary sequence 0100...

## 7.13 DSP

<b>Header File</b>	:	dsp_*.h
<b>Source File</b>	:	dsp_*.cpp
<b>Version</b>	:	20180423 (Celestino Martins)

This super block simulates the digital signal processing (DSP) algorithms for system impairments compensation in digital domain. It includes the real to complex block and carrier phase recovery block (CPE). It receives two real input signal and outputs a complex signal.

### Input Parameters

Parameter	Type	Values	Default
nTaps	int	any	25
NtestPhase	int	any	32
methodType	int	any	[0, 1]
samplingPeriod	double	any	0.0

Table 7.8: DSP input parameters

### Methods

```
DSP(vector<Signal *> &InputSig, vector<Signal *> &OutputSig);

void setCPEnTaps(double nTaps) B02.setnTaps(nTaps);

void setCPETestPhase(double TestPhase) B02.setTestPhase(TestPhase);

void setCPESamplingPeriod(double sPeriod) B02.setSamplingPeriod(sPeriod);

void setCPEmethodType(string mType) B02.setmethodType(mType);

void setSamplingPeriod(double sPeriod) B02.setSamplingPeriod(sPeriod); ;
```

### Functional description

This super block is composed of two blocks, real to complex block and carrier phase recovery block. The two real input signals are combined into a complex signal using real to complex block. The obtained complex signal is then fed to the CPE block, where the laser phase noise compensation is performed.

**Input Signals**

**Number:** 2

**Output Signals**

**Number:** 1

**Type:** Electrical complex signal

**Examples**

**Sugestions for future improvement**

## 7.14 Electrical Signal Generator

This block generates time continuous amplitude continuous signal, having only one output and no input signal.

### 7.14.1 ContinuousWave

Continuous Wave the function of the desired signal. This must be introduce by using the function `setFunction(ContinuousWave)`. This function generates a continuous signal with value 1. However, this value can be multiplied by a specific gain, which can be set by using the function `setGain()`. This way, this block outputs a continuous signal with value  $1 \times \text{gain}$ .

#### Input Parameters

- `ElectricalSignalFunction` `signalFunction`  
(`ContinuousWave`)
- `samplingPeriod{}` `(double)`
- `symbolPeriod{}` `(double)`

#### Methods

```
ElectricalSignalGenerator() {};
void initialize(void);
bool runBlock(void);
void setFunction(ElectricalSignalFunction fun) ElectricalSignalFunction getFunction()
void setSamplingPeriod(double speriod) double getSamplingPeriod()
void setSymbolPeriod(double speriod) double getSymbolPeriod()
void setGain(double gvalue) double getGain()
```

#### Functional description

The `signalFunction` parameter allows the user to select the signal function that the user wants to output.

**Continuous Wave** Outputs a time continuous amplitude continuous signal with amplitude 1 multiplied by the gain inserted.

**Input Signals**

**Number:** 0

**Type:** No type

**Output Signals**

**Number:** 1

**Type:** TimeContinuousAmplitudeContinuous

**Examples****Sugestions for future improvement**

Implement other functions according to the needs.

## 7.15 Fork

<b>Header File</b>	:	fork_20171119.h
<b>Source File</b>	:	fork_20171119.cpp
<b>Version</b>	:	20171119 (Student Name: Romil Patel)

### Input Parameters

— NA —

### Input Signals

**Number:** 1

**Type:** Any type (BinaryValue, IntegerValue, RealValue, ComplexValue, ComplexValueXY, PhotonValue, PhotonValueMP, Message)

### Output Signals

**Number:** 2

**Type:** Same as applied to the input.

**Number:** 3

**Type:** Same as applied to the input.

### Functional Description

This block accepts any type signal and outputs two replicas of the input signal.

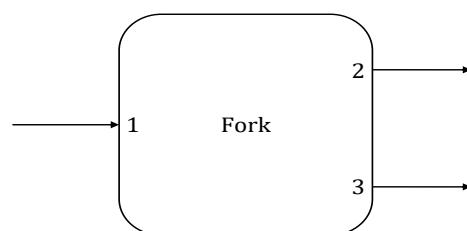


Figure 7.8: Fork

## 7.16 Gaussian Source

<b>Header File</b>	:	gaussian_source.h
<b>Source File</b>	:	gaussian_source.cpp

This block simulates a random number generator that follows a Gaussian statistics. It produces one output real signal and it doesn't accept input signals.

### Input Parameters

Parameter	Type	Values	Default
mean	double	any	0
Variance	double	any	1

Table 7.9: Gaussian source input parameters

### Methods

GaussianSource()

```
GaussianSource(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig){};

void initialize(void);

bool runBlock(void);

void setAverage(double Average);
```

### Functional description

This block generates a complex signal with a specified phase given by the input parameter *phase*.

**Input Signals**

**Number:** 0

**Output Signals**

**Number:** 1

**Type:** Continuous signal (TimeDiscreteAmplitudeContinuousReal)

**Examples**

**Sugestions for future improvement**

## 7.17 MQAM Receiver

<b>Header File</b>	:	m_qam_receiver.h
<b>Source File</b>	:	m_qam_receiver.cpp

**Warning:** *homodyne\_receiver* is not recommended. Use *m\_qam\_homodyne\_receiver* instead.

This block of code simulates the reception and demodulation of an optical signal (which is the input signal of the system) outputing a binary signal. A simplified schematic representation of this block is shown in figure 7.9.



Figure 7.9: Basic configuration of the MQAM receiver

### Functional description

This block accepts one optical input signal and outputs one binary signal that corresponds to the M-QAM demodulation of the input signal. It is a complex block (as it can be seen from figure 7.10) of code made up of several simpler blocks whose description can be found in the *lib* repository.

It can also be seen from figure 7.10 that there's an extra internal (generated inside the homodyne receiver block) input signal generated by the *Clock*. This block is used to provide the sampling frequency to the *Sampler*.

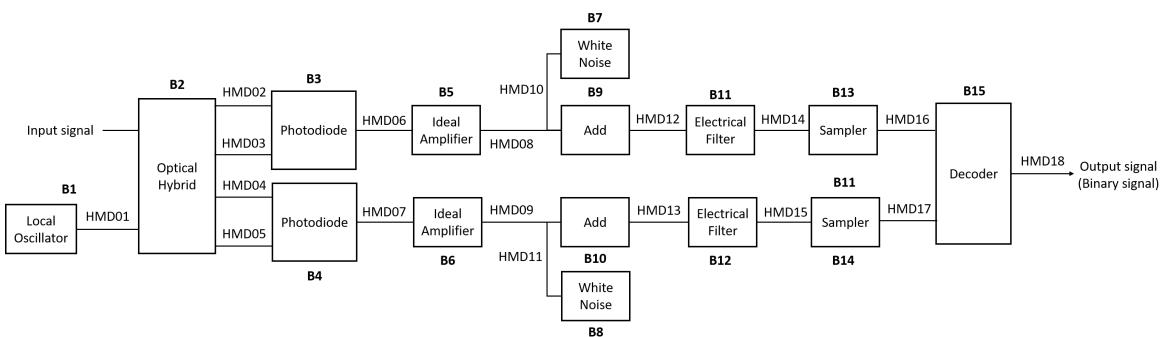


Figure 7.10: Schematic representation of the block homodyne receiver.

## Input parameters

This block has some input parameters that can be manipulated by the user in order to change the basic configuration of the receiver. Each parameter has associated a function that allows for its change. In the following table (table 7.15) the input parameters and corresponding functions are summarized.

Input parameters	Function	Type	Accepted values
IQ amplitudes	setIqAmplitudes	Vector of coordinate points in the I-Q plane	Example for a 4-QAM mapping: { { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }
Local oscillator power (in dBm)	setLocalOscillatorOpticalPower_dBm	double(t_real)	Any double greater than zero
Local oscillator phase	setLocalOscillatorPhase	double(t_real)	Any double greater than zero
Responsivity of the photodiodes	setResponsivity	double(t_real)	$\in [0,1]$
Amplification (of the TI amplifier)	setAmplification	double(t_real)	Positive real number
Noise amplitude (introduced by the TI amplifier)	setNoiseAmplitude	double(t_real)	Real number greater than zero
Samples to skip	setSamplesToSkip	int(t_integer)	
Save internal signals	setSaveInternalSignals	bool	True or False
Sampling period	setSamplingPeriod	double	Given by $symbolPeriod / samplesPerSymbol$

Table 7.10: List of input parameters of the block MQAM receiver

## Methods

HomodyneReceiver(vector<Signal \*> &inputSignal, vector<Signal \*> &outputSignal)  
**(constructor)**

```
void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues)
vector<t_iqValues> const getIqAmplitudes(void)
void setLocalOscillatorSamplingPeriod(double sPeriod)
void setLocalOscillatorOpticalPower(double opticalPower)
void setLocalOscillatorOpticalPower_dBm(double opticalPower_dBm)
void setLocalOscillatorPhase(double lOscillatorPhase)
void setLocalOscillatorOpticalWavelength(double lOscillatorWavelength)
void setSamplingPeriod(double sPeriod)
void setResponsivity(t_real Responsivity)
void setAmplification(t_real Amplification)
void setNoiseAmplitude(t_real NoiseAmplitude)
void setImpulseResponseTimeLength(int impResponseTimeLength)
void setFilterType(PulseShaperFilter fType)
void setRollOffFactor(double rOffFactor)
void setClockPeriod(double per)
void setSamplesToSkip(int sToSkip)
```

**Input Signals**

**Number:** 1

**Type:** Optical signal

**Output Signals**

**Number:** 1

**Type:** Binary signal

**Example**

**Sugestions for future improvement**

## 7.18 IQ Modulator

<b>Header File</b>	:	iq_modulator.h
<b>Source File</b>	:	iq_modulator.cpp

This blocks accepts one input signal continuous in both time and amplitude and it can produce either one or two output signals. It generates an optical signal and it can also generate a binary signal.

### Input Parameters

Parameter	Type	Values	Default
outputOpticalPower	double	any	$1e - 3$
outputOpticalWavelength	double	any	$1550e - 9$
outputOpticalFrequency	double	any	speed_of_light/outputOpticalWavelength

Table 7.11: Binary source input parameters

### Methods

```
IqModulator(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig){};
```

```
void initialize(void);
bool runBlock(void);
void setOutputOpticalPower(double outOpticalPower)
void setOutputOpticalPower_dBm(double outOpticalPower_dBm)
void setOutputOpticalWavelength(double outOpticalWavelength)
void setOutputOpticalFrequency(double outOpticalFrequency)
```

### Functional Description

This block takes the two parts of the signal: in phase and in amplitude and it combines them to produce a complex signal that contains information about the amplitude and the phase.

This complex signal is multiplied by  $\frac{1}{2}\sqrt{outputOpticalPower}$  in order to reintroduce the information about the energy (or power) of the signal. This signal corresponds to an optical signal and it can be a scalar or have two polarizations along perpendicular axis. It is the signal that is transmitted to the receptor.

The binary signal is sent to the Bit Error Rate (BER) measurement block.

## Input Signals

**Number** : 2

**Type** : Sequence of impulses modulated by the filter  
(ContinuousTimeContiousAmplitude)

## Output Signals

**Number** : 1 or 2

**Type** : Complex signal (optical) (ContinuousTimeContinuousAmplitude) and binary signal (DiscreteTimeDiscreteAmplitude)

## Example

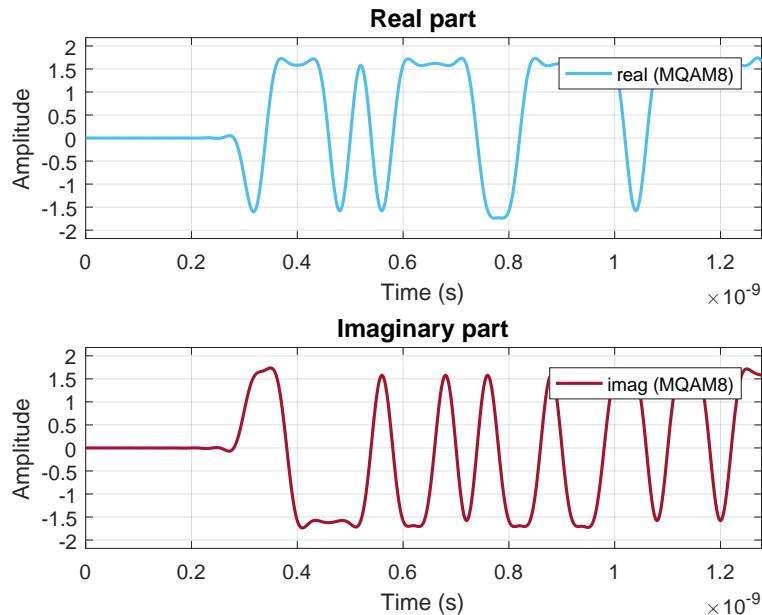


Figure 7.11: Example of a signal generated by this block for the initial binary signal 0100...

## 7.19 Local Oscillator

<b>Header File</b>	:	local_oscillator.h
<b>Source File</b>	:	local_oscillator.cpp
<b>Version</b>	:	20180130

This block simulates a local oscillator with constant power and initial phase. It produces one output complex signal and it doesn't accept input signals.

### Input Parameters

Parameter	Type	Values	Default
opticalPower	double	any	1e - 3
outputOpticalWavelength	double	any	1550e - 9
outputOpticalFrequency	double	any	SPEED_OF_LIGHT / outputOpticalWavelength
phase	double	$\in [0, \frac{\pi}{2}]$	0
samplingPeriod	double	any	0.0
symbolPeriod	double	any	0.0
signaltoNoiseRatio	double	any	0.0
laserLineWidth	double	any	0.0
laserRIN	double	any	0.0

Table 7.12: Binary source input parameters

### Methods

LocalOscillator()

```

LocalOscillator(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig){};

void initialize(void);

bool runBlock(void);

void setSamplingPeriod(double sPeriod);

void setSymbolPeriod(double sPeriod);

void setOpticalPower(double oPower);

void setOpticalPower_dBm(double oPower_dBm);

void setWavelength(double wlength);

```

```
void setFrequency(double freq);  
void setPhase(double lOscillatorPhase);  
void setSignaltoNoiseRatio(double sNoiseRatio);  
void setLaserLinewidth(double laserLinewidth);  
void setLaserRIN(double laserRIN);
```

### Functional description

This block generates a complex signal with a specified phase given by the input parameter *phase*.

### Input Signals

**Number:** 0

### Output Signals

**Number:** 1

**Type:** Optical signal

## 7.20 Local Oscillator

<b>Header File</b>	:	local_oscillator.h
<b>Source File</b>	:	local_oscillator.cpp

This block simulates a local oscillator with constant power and initial phase. It produces one output complex signal and it doesn't accept input signals.

### Input Parameters

Parameter	Type	Values	Default
opticalPower	double	any	1e - 3
outputOpticalWavelength	double	any	1550e - 9
outputOpticalFrequency	double	any	SPEED_OF_LIGHT / outputOpticalWavelength
phase0	double	$\in [0, \frac{\pi}{2}]$	0
samplingPeriod	double	any	0.0
laserLW	double	any	0.0
laserRIN	double	any	0.0

Table 7.13: Local oscillator input parameters

### Methods

LocalOscillator()

```

LocalOscillator(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig) {};

void initialize(void);

bool runBlock(void);

void setSamplingPeriod(double sPeriod);

void setOpticalPower(double oPower);

void setOpticalPower_dBm(double oPower_dBm);

void setWavelength(double wlengh);

void setPhase(double lOscillatorPhase);

void setLaserLinewidth(double laserLinewidth);

```

```
double getLaserLinewidth();  
  
void setLaserRIN(double LOlaserRIN);  
  
double getLaserRIN();
```

### Functional description

This block generates a complex signal with a specified initial phase given by the input parameter *phase0*. The phase noise can be simulated by adjusting the laser linewidth in parameter *laserLW*. The relative intensity noise (RIN) can be also adjusting according to the parameter *laserRIN*.

**Input Signals**

**Number:** 0

**Output Signals**

**Number:** 1

**Type:** Optical signal

**Examples**

**Sugestions for future improvement**

## 7.21 MQAM Mapper

<b>Header File</b>	:	m_qam_mapper.h
<b>Source File</b>	:	m_qam_mapper.cpp

This block does the mapping of the binary signal using a  $m$ -QAM modulation. It accepts one input signal of the binary type and it produces two output signals which are a sequence of 1's and -1's.

### Input Parameters

Parameter	Type	Values	Default
m	int	$2^n$ with $n$ integer	4
iqAmplitudes	vector<t_complex>	—	{ { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }

Table 7.14: Binary source input parameters

### Methods

```
MQamMapper(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig) {}

void initialize(void);

bool runBlock(void);

void setM(int mValue);

void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues);
```

### Functional Description

In the case of  $m=4$  this block attributes to each pair of bits a point in the I-Q space. The constellation used is defined by the *iqAmplitudes* vector. The constellation used in this case is illustrated in figure 7.12.

### Input Signals

**Number** : 1

**Type** : Binary (DiscreteTimeDiscreteAmplitude)

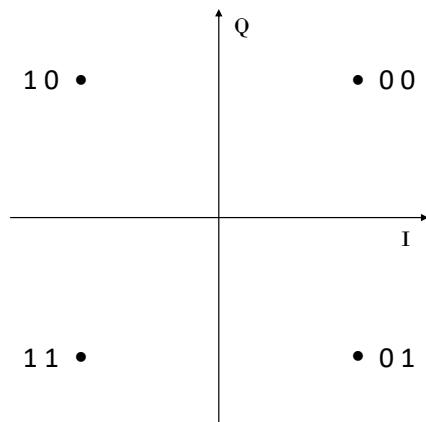


Figure 7.12: Constellation used to map the signal for  $m=4$

### Output Signals

**Number** : 2

**Type** : Sequence of 1's and -1's (DiscreteTimeDiscreteAmplitude)

### Example

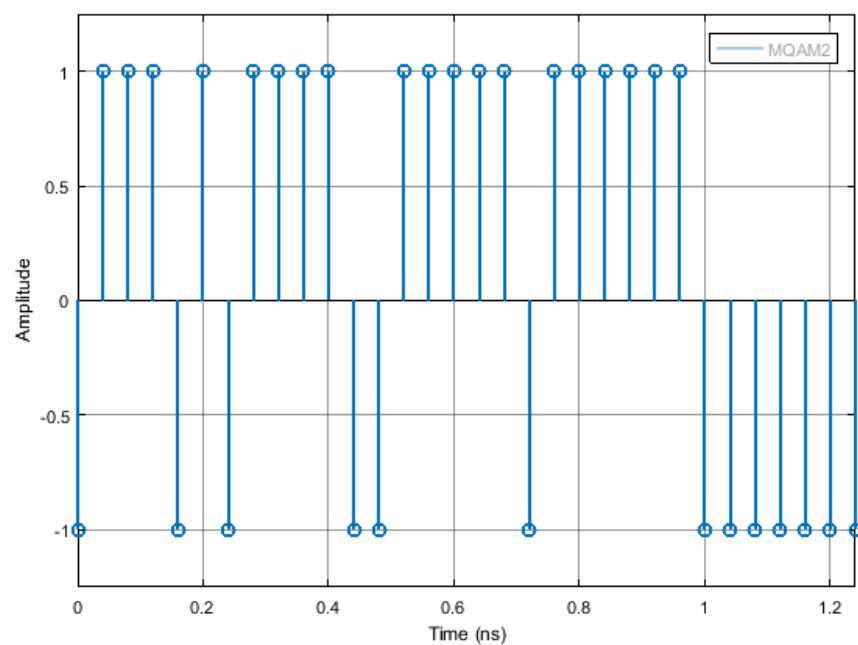


Figure 7.13: Example of the type of signal generated by this block for the initial binary signal 0100...

## 7.22 MQAM Transmitter

<b>Header File</b>	:	m_qam_transmitter.h
<b>Source File</b>	:	m_qam_transmitter.cpp

This block generates a MQAM optical signal. It can also output the binary sequence. A schematic representation of this block is shown in figure 7.14.

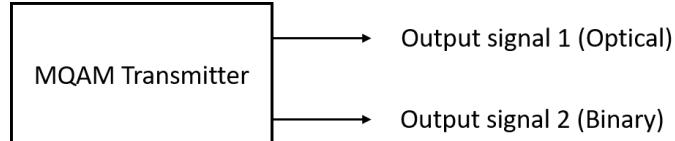


Figure 7.14: Basic configuration of the MQAM transmitter

### Functional description

This block generates an optical signal (output signal 1 in figure 7.15). The binary signal generated in the internal block Binary Source (block B1 in figure 7.15) can be used to perform a Bit Error Rate (BER) measurement and in that sense it works as an extra output signal (output signal 2 in figure 7.15).

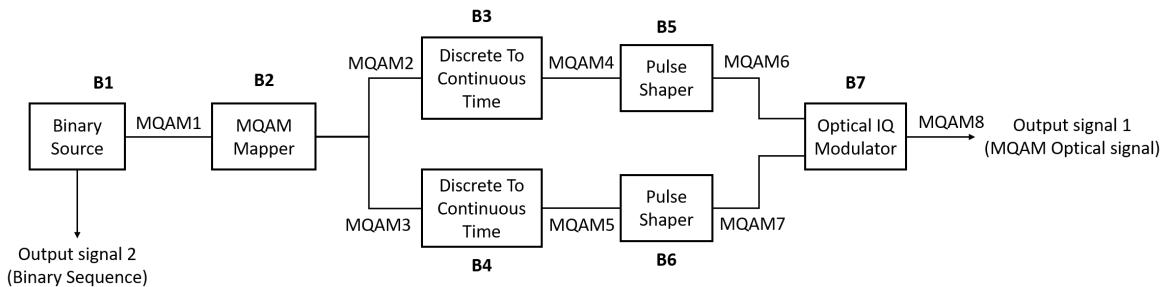


Figure 7.15: Schematic representation of the block MQAM transmitter.

### Input parameters

This block has a special set of functions that allow the user to change the basic configuration of the transmitter. The list of input parameters, functions used to change them and the values that each one can take are summarized in table 7.15.

Input parameters	Function	Type	Accepted values
Mode	setMode()	string	PseudoRandom Random DeterministicAppendZeros DeterministicCyclic
Number of bits generated	setNumberOfBits()	int	Any integer
Pattern length	setPatternLength()	int	Real number greater than zero
Number of bits	setNumberOfBits()	long	Integer number greater than zero
Number of samples per symbol	setNumberOfSamplesPerSymbol()	int	Integer number of the type $2^n$ with n also integer
Roll off factor	setRollOffFactor()	double	$\in [0,1]$
IQ amplitudes	setIqAmplitudes()	Vector of coordinate points in the I-Q plane	Example for a 4-qam mapping: { { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }
Output optical power	setOutputOpticalPower()	int	Real number greater than zero
Save internal signals	setSaveInternalSignals()	bool	True or False

Table 7.15: List of input parameters of the block MQAM transmitter

## Methods

MQamTransmitter(vector<Signal \*> &inputSignal, vector<Signal \*> &outputSignal);  
**(constructor)**

void set(int opt);

void setMode(BinarySourceMode m)

BinarySourceMode const getMode(void)

void setProbabilityOfZero(double pZero)

double const getProbabilityOfZero(void)

void setBitStream(string bStream)

```
string const getBitStream(void)

void setNumberOfBits(long int nOfBits)

long int const getNumberOfBits(void)

void setPatternLength(int pLength)

int const getPatternLength(void)

void setBitPeriod(double bPeriod)

double const getBitPeriod(void)

void setM(int mValue) int const getM(void)

void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues)

vector<t_iqValues> const getIqAmplitudes(void)

void setNumberOfSamplesPerSymbol(int n)

int const getNumberOfSamplesPerSymbol(void)

void setRollOffFactor(double rOffFactor)

double const getRollOffFactor(void)

void setSeeBeginningOfImpulseResponse(bool sBeginningOfImpulseResponse)

double const getSeeBeginningOfImpulseResponse(void)

void setOutputOpticalPower(t_real outOpticalPower)

t_real const getOutputOpticalPower(void)

void setOutputOpticalPower_dBm(t_real outOpticalPower_dBm)

t_real const getOutputOpticalPower_dBm(void)
```

## Output Signals

**Number:** 1 optical and 1 binary (optional)

**Type:** Optical signal

## Example

### Sugestions for future improvement

Add to the system another block similar to this one in order to generate two optical signals with perpendicular polarizations. This would allow to combine the two optical signals and generate an optical signal with any type of polarization.

## 7.23 Netxpto

<b>Header File</b>	:	netxpto.h
	:	netxpto_20180118.h
<b>Source File</b>	:	netxpto.cpp
	:	netxpto_20180118.cpp

This block can work in two configurations: with an external clock or without it. In the latter it accepts two input signals one being the clock and the other the signal to be demodulated. In the other configuration there's only one input signal which is the signal.

The output signal is obtained by sampling the input signal with a predetermined sampling rate provided either internally or by the clock.

### Input Parameters

Parameter	Type	Values	Default
samplesToSkip	int	any (smaller than the number of samples generated)	0

Table 7.16: Sampler input parameters

### Methods

Sampler()

    Sampler(vector<Signal \*> &InputSig, vector<Signal \*> &OutputSig) :Block(InputSig, OutputSig)

    void initialize(void)

    bool runBlock(void)

    void setSamplesToSkip(t\_integer sToSkip)

### Functional description

This block can work with an external clock or without it.

In the case of having an external clock it accepts two input signals. The signal to be demodulated which is complex and a clock signal that is a sequence of Dirac delta functions with a predetermined period that corresponds to the sampling period. The signal and the clock signal are scanned and when the clock has the value of 1.0 the correspondent complex value of the signal is placed in the buffer corresponding to the output signal.

There's a detail worth noting. The electrical filter has an impulse response time length of 16 (in units of symbol period). This means that when modulating a bit the spike in the signal corresponding to that bit will appear 8 units of symbol period later. For this reason there's the need to skip the earlier samples of the signal when demodulating it. That's the purpose of the *samplesToSkip* parameter.

Between the binary source and the current block the signal is filtered twice which means that this effect has to be taken into account twice. Therefore the parameter *samplesToSkip* is given by  $2 * 8 * \text{samplesPerSymbol}$ .

## Input Signals

**Number:** 1

**Type:** Electrical real (TimeContinuousAmplitudeContinuousReal)

## Output Signals

**Number:** 1

**Type:** Electrical real (TimeDiscreteAmplitudeContinuousReal)

## Examples

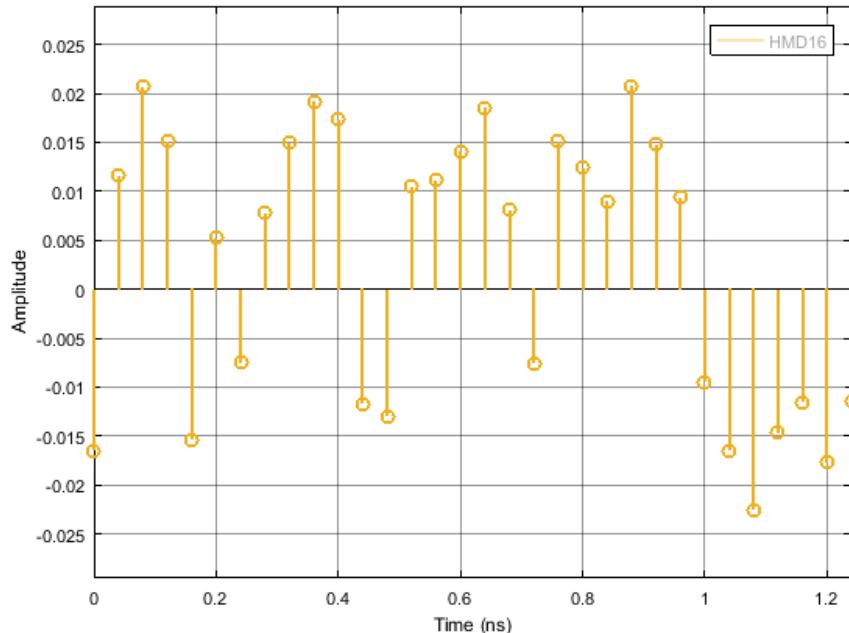


Figure 7.16: Example of the output signal of the sampler

### 7.23.1 Version 20180118

Adds the type `t_photon_mp_xy`, to support multi-path photon signals with polarization information.

Changes the signal data type to make private its data structure, only allowing its access through appropriate methods.

### Sugestions for future improvement

## 7.24 Alice QKD

This block is the processor for Alice does all tasks that she needs. This block accepts binary, messages, and real continuous time signals. It produces messages, binary and real discrete time signals.

### Input Parameters

- double RateOfPhotons{1e3}
- int StringPhotonsLength{ 12 }

### Methods

```
AliceQKD (vector <Signal*> &inputSignals, vector <Signal*> &outputSignals) :  
Block(inputSignals, outputSignals) {};  
void initialize(void);  
bool runBlock(void);  
void setRateOfPhotons(double RPhotons) { RateOfPhotons = RPhotons; }; double const  
getRateOfPhotons(void) { return RateOfPhotons; };  
void setStringPhotonsLength(int pLength) { StringPhotonsLength = pLength; }; int const  
getStringPhotonsLength(void) { return StringPhotonsLength; };
```

### Functional description

This block receives a sequence of binary numbers (1's or 0's) and a clock signal which will set the rate of the signals produced to generate single polarized photons. The real discrete time signal **SA\_1** is generated based on the clock signal and the real discrete time signal **SA\_2** is generated based on the random sequence of bits received through the signal **NUM\_A**. This last sequence is analysed by the polarizer in pairs of bits in which each pair has a bit for basis choice and other for direction choice.

This block also produces classical messages signals to send to Bob as well as binary messages to the mutual information block with information about the photons it sent.

### Input Signals

**Number** : 3

**Type** : Binary, Real Continuous Time and Messages signals.

### Output Signals

**Number** : 3

**Type** : Binary, Real Discrete Time and Messages signals.

**Examples**

**Sugestions for future improvement**

## 7.25 Polarizer

This block is responsible of changing the polarization of the input photon stream signal by using the information from the other real time discrete input signal. This way, this block accepts two input signals: one photon stream and other real discrete time signal. The real discrete time input signal must be a signal discrete in time in which the amplitude can be 0 or 1. The block will analyse the pairs of values by interpreting them as basis and polarization direction.

### Input Parameters

- $m\{4\}$
- Amplitudes { {1,1}, {-1,1}, {-1,-1}, { 1,-1} }

### Methods

```
Polarizer (vector <Signal*> &inputSignals, vector <Signal*>&outputSignals) :  
Block(inputSignals, outputSignals) {};  
void initialize(void);  
bool runBlock(void);  
void setM(int mValue);  
void setAmplitudes(vector <t_iqValues> AmplitudeValues);
```

### Functional description

Considering  $m=4$ , this block attributes for each pair of bits a point in space. In this case, it is be considered four possible polarization states:  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$  and  $135^\circ$ .

### Input Signals

**Number** : 2

**Type** : Photon Stream and a Sequence of 0's and '1s (DiscreteTimeDiscreteAmplitude).

### Output Signals

**Number** : 1

**Type** : Photon Stream

### Examples

### Sugestions for future improvement

## 7.26 Probability Estimator

This blocks accepts an input binary signal and it calculates the probability of having a value "1" or "0" according to the number of samples acquired and according to the z-score value set depending on the confidence interval. It produces an output binary signal equals to the input. Nevertheless, this block has an additional output which is a txt file with information related with probability values, number of samples acquired and margin error values for each probability value.

In statistics theory, considering the results space  $\Omega$  associated with a random experience and  $A$  an event such that  $P(A) = p \in ]0, 1[$ . Lets  $X : \Omega \rightarrow \mathbb{R}$  such that

$$\begin{aligned} X(\omega) &= 1 & \text{,if } \omega \in A \\ X(\omega) &= 0 & \text{,if } \omega \in \bar{A} \end{aligned} \tag{7.9}$$

This way, there only are two possible results: success when the outcome is 1 or failure when the outcome is 0. The probability of success is  $P(X = 1)$  and the probability of failure is  $P(X = 0)$ ,

$$\begin{aligned} P(X = 1) &= P(A) = p \\ P(X = 0) &= P(\bar{A}) = 1 - p \end{aligned} \tag{7.10}$$

$X$  follows the Bernoulli law with parameter  $p$ ,  $X \sim \mathbf{B}(p)$ , being the expected value of the Bernoulli random value  $E(X) = p$  and the variance  $\text{VAR}(X) = p(1 - p)$  [1].

Assuming that  $N$  independent trials are performed, in which a success occurs with probability  $p$  and a failure occurs with probability  $1-p$ . If  $X$  is the number of successes that occur in the  $N$  trials,  $X$  is a binomial random variable with parameters  $(n, p)$ . Since  $N$  is large enough,  $X$  can be approximately normally distributed with mean  $np$  and variance  $np(1 - p)$ .

$$\frac{X - np}{\sqrt{np(1 - p)}} \sim N(0, 1). \tag{7.11}$$

In order to obtain a confidence interval for  $p$ , lets assume the estimator  $\hat{p} = \frac{X}{N}$  the fraction of samples equals to 1 with regard to the total number of samples acquired. Since  $\hat{p}$  is the estimator of  $p$ , it should be approximately equal to  $p$ . As a result, for any  $\alpha \in 0, 1$  we have that:

$$\frac{X - np}{\sqrt{np(1 - p)}} \sim N(0, 1) \tag{7.12}$$

$$\begin{aligned} P\{-z_{\alpha/2} < \frac{X - np}{\sqrt{np(1 - p)}} < z_{\alpha/2}\} &\approx 1 - \alpha \\ P\{-z_{\alpha/2}\sqrt{np(1 - p)} < np - X < z_{\alpha/2}\sqrt{np(1 - p)}\} &\approx 1 - \alpha \\ P\{\hat{p} - z_{\alpha/2}\sqrt{\hat{p}(1 - \hat{p})/n} < p < \hat{p} + z_{\alpha/2}\sqrt{\hat{p}(1 - \hat{p})/n}\} &\approx 1 - \alpha \end{aligned} \tag{7.13}$$

This way, a confidence interval for  $p$  is approximately  $100(1 - \alpha)$  percent.

## Input Parameters

- zscore  
(double)
- fileName  
(string)

## Methods

```
ProbabilityEstimator(vector<Signal *> &InputSig, vector<Signal *> &OutputSig)
:Block(InputSig, OutputSig){};

void initialize(void);

bool runBlock(void);

void setProbabilityExpectedX(double probx) double getProbabilityExpectedX()

void setProbabilityExpectedY(double proby) double getProbabilityExpectedY()

void setZScore(double z) double getZScore()
```

## Functional description

This block receives an input binary signal with values "0" or "1" and it calculates the probability of having each number according with the number of samples acquired. This probability is calculated using the following formulas:

$$\text{Probability}_1 = \frac{\text{Number of 1's}}{\text{Number of Received Bits}} \quad (7.14)$$

$$\text{Probability}_0 = \frac{\text{Number of 0's}}{\text{Number of Received Bits}}. \quad (7.15)$$

The error margin is calculated based on the z-score set which specifies the confidence interval using the following formula:

$$ME = z_{\text{score}} \times \sqrt{\frac{\hat{p}(1 - \hat{p})}{N}} \quad (7.16)$$

being  $\hat{p}$  the expected probability calculated using the formulas above and  $N$  the total number of samples.

This block outputs a txt file with information regarding with the total number of received bits, the probability of 1, the probability of 0 and the respective errors.

## Input Signals

**Number:** 1

**Type:** Binary

## Output Signals

**Number:** 2

**Type:** Binary

**Type:** txt file

## Examples

Lets calculate the margin error for N of samples in order to obtain  $X$  inside a specific confidence interval, which in this case we assume a confidence interval of 99%.

We will use *z-score* from a table about standard normal distribution, which in this case is 2.576, since a confidence interval of 99% was chosen, to calculate the expected error margin,

$$\begin{aligned} ME &= \pm z_{\alpha/2} \frac{\sigma}{\sqrt{N}} \\ ME &= \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{N}}, \end{aligned} \quad (7.17)$$

where, ME is the error margin,  $z_{\alpha/2}$  is the *z-score* for a specific confidence interval,  $\sigma = \sqrt{\text{VAR}(X)} = \sqrt{\hat{p}(1 - \hat{p})}$  is the standard deviation and  $N$  the number of samples.

This way, with a 99% confidence interval, between  $(\hat{p} - ME) \times 100$  and  $(\hat{p} + ME) \times 100$  percent of the samples meet the standards.

## Sugestions for future improvement

## 7.27 Bob QKD

This block is the processor for Bob does all tasks that she needs. This block accepts and produces:

1.

2.

### **Input Parameters**

- 
- 

### **Methods**

### **Functional description**

### **Input Signals**

### **Examples**

### **Sugestions for future improvement**

## 7.28 Eve QKD

This block is the processor for Eve does all tasks that she needs. This block accepts and produces:

1.

2.

### **Input Parameters**

- 
- 

### **Methods**

### **Functional description**

### **Input Signals**

### **Examples**

### **Sugestions for future improvement**

## 7.29 Rotator Linear Polarizer

This block accepts a Photon Stream signal and a Real discrete time signal. It produces a photon stream by rotating the polarization axis of the linearly polarized input photon stream by an angle of choice.

### Input Parameters

- m[2]
- axis { {1,0}, { $\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}$  } }

### Methods

```
RotatorLinearPolarizer(vector <Signal*> &inputSignals, vector <Signal*> &outputSignals) :
Block(inputSignals, outputSignals) {};
void initialize(void);
bool runBlock(void);
void setM(int mValue);
void setAxis(vector <t_iqValues> AxisValues);
```

### Functional description

This block accepts the input parameter m, which defines the number of possible rotations. In this case m=2, the block accepts the rectilinear basis, defined by the first position of the second input parameter axis, and the diagonal basis, defined by the second position of the second input parameter axis. This block rotates the polarization axis of the linearly polarized input photon stream to the basis defined by the other input signal. If the discrete value of this signal is 0, the rotator is set to rotate the input photon stream by 0°, otherwise, if the value is 1, the rotator is set to rotate the input photon stream by an angle of 45°.

### Input Signals

**Number** : 2

**Type** : Photon Stream and a Sequence of 0's and '1s (DiscreteTimeDiscreteAmplitude)

### Output Signals

**Number** : 1

**Type** : Photon Stream

**Examples**

**Sugestions for future improvement**

## 7.30 Optical Switch

This block has one input signal and two input signals. Furthermore, it accepts an additional input binary input signal which is used to decide which of the two outputs is activated.

### Input Parameters

No input parameters.

### Methods

```
OpticalSwitch(vector <Signal*> &inputSignals, vector <Signal*> &outputSignals) :  
Block(inputSignals, outputSignals) {};  
void initialize(void);  
bool runBlock(void);
```

### Functional description

This block receives an input photon stream signal and it decides which path the signal must follow. In order to make this decision it receives a binary signal (0's and 1's) and it switch the output path according with this signal.

### Input Signals

**Number** : 1

**Type** : Photon Stream

### Output Signals

**Number** : 2

**Type** : Photon Stream

### Examples

### Sugestions for future improvement

### 7.31 Optical Hybrid

<b>Header File</b>	:	optical_hybrid.h
<b>Source File</b>	:	optical_hybrid.cpp

This block simulates an optical hybrid. It accepts two input signals corresponding to the signal and to the local oscillator. It generates four output complex signals separated by  $90^\circ$  in the complex plane. Figure 7.17 shows a schematic representation of this block.

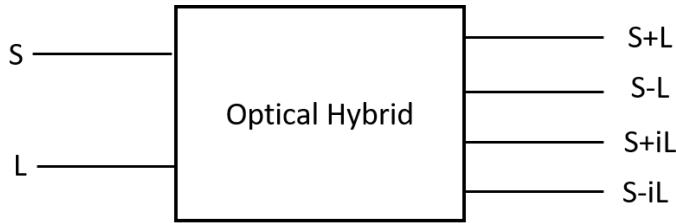


Figure 7.17: Schematic representation of an optical hybrid.

#### Input Parameters

Parameter	Type	Values	Default
outputOpticalPower	double	any	$1e - 3$
outputOpticalWavelength	double	any	$1550e - 9$
outputOpticalFrequency	double	any	$SPEED\_OF\_LIGHT / outputOpticalWavelength$
powerFactor	double	$\leq 1$	0.5

Table 7.17: Optical hybrid input parameters

#### Methods

OpticalHybrid()

```
OpticalHybrid(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig)
```

```
void initialize(void)
```

```
bool runBlock(void)
```

```
void setOutputOpticalPower(double outOpticalPower)
```

```
void setOutputOpticalPower_dBm(double outOpticalPower_dBm)
```

```
void setOutputOpticalWavelength(double outOpticalWavelength)  
void setOutputOpticalFrequency(double outOpticalFrequency)  
void setPowerFactor(double pFactor)
```

### Functional description

This block accepts two input signals corresponding to the signal to be demodulated ( $S$ ) and to the local oscillator ( $L$ ). It generates four output optical signals given by  $powerFactor \times (S + L)$ ,  $powerFactor \times (S - L)$ ,  $powerFactor \times (S + iL)$ ,  $powerFactor \times (S - iL)$ . The input parameter  $powerFactor$  assures the conservation of optical power.

### Input Signals

**Number:** 2

**Type:** Optical (OpticalSignal)

### Output Signals

**Number:** 4

**Type:** Optical (OpticalSignal)

### Examples

### Sugestions for future improvement

## 7.32 Photodiode pair

<b>Header File</b>	:	photodiode_old.h
<b>Source File</b>	:	photodiode_old.cpp

This block simulates a block of two photodiodes assembled like in figure 7.18. It accepts two optical input signals and outputs one electrical signal. Each photodiode converts an optical signal to an electrical signal. The two electrical signals are then subtracted and the resulting signals corresponds to the output signal of the block.

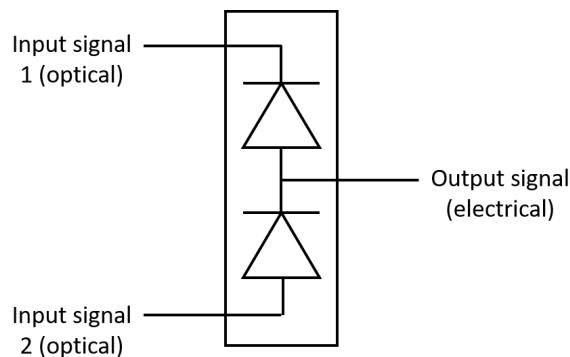


Figure 7.18: Schematic representation of the physical equivalent of the photodiode code block.

### Input Parameters

- responsivity{1}
- outputOpticalWavelength{ 1550e-9 }
- outputOpticalFrequency{ SPEED\_OF\_LIGHT / wavelength }

### Methods

Photodiode()

```
Photodiode(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig)
```

```
void initialize(void)
```

```
bool runBlock(void)
```

```
void setResponsivity(t_real Responsivity)
```

### Functional description

This block accepts two input optical signals. It computes the optical power of the signal (given by the absolute value squared of the input signal) and multiplies it by the *responsivity* of the photodiode. This product corresponds to the current generated by the photodiode. This is done for each of the input signals. The two currents are then subtracted producing a single output current, that corresponds to the output electrical signal of the block.

### Input Signals

**Number:** 2

**Type:** Optical (OpticalSignal)

### Output Signals

**Number:** 1

**Type:** Electrical (TimeContinuousAmplitudeContinuousReal)

### Examples

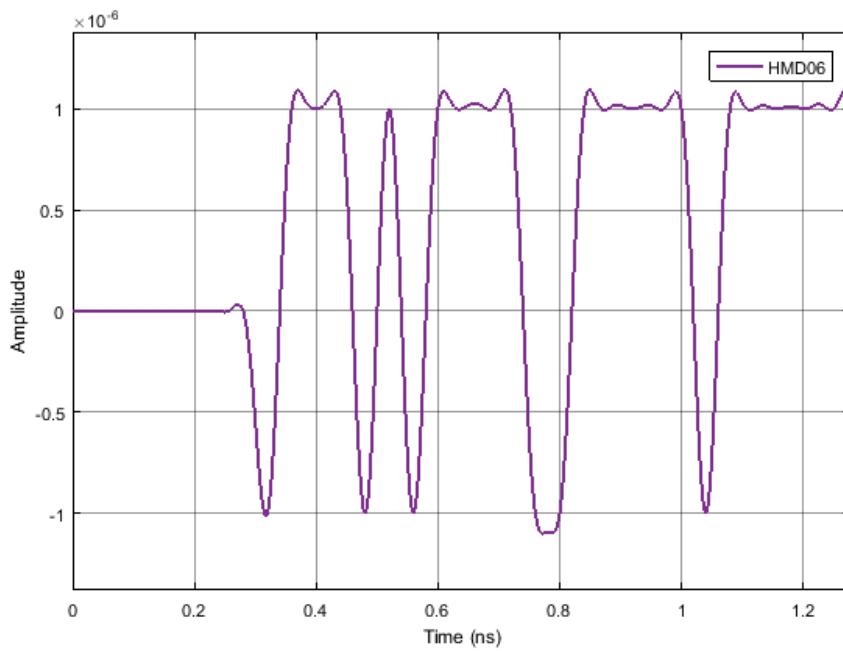


Figure 7.19: Example of the output singal of the photodiode block for a bimary sequence 01

**Sugestions for future improvement**

### 7.33 Photoelectron Generator

<b>Header File</b>	:	photoelectron_generator_*.h
<b>Source File</b>	:	photoelectron_generator_*.cpp
<b>Version</b>	:	20180302 (Diamantino Silva)

This block simulates the generation of photoelectrons by a photodiode, performing the conversion of an incident electric field into an output current proportional to the field's instantaneous power. It is also capable of simulating shot noise.

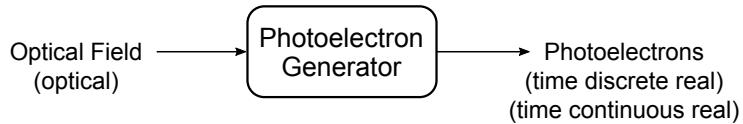


Figure 7.20: Schematic representation of the photoelectron generator code block

#### Theoretical description

The operation of a real photodiode is based on the photoelectric effect, which consists on the removal of one electron from the target material by a single photon, with a probability  $\eta$ . Given an input beam with an optical power  $P(t)$  in which the photons are around the wavelength  $\lambda$ , the flux of photons  $\phi(t)$  is calculated as [1]

$$\phi(t) = \frac{P(t)\lambda}{hc} \quad (7.18)$$

therefore, the mean number of photons in a given interval  $[t, t + T]$  is

$$\bar{n}(t) = \int_t^{t+T} \phi(\tau) d\tau \quad (7.19)$$

But the actual number of photons in a given time interval,  $n(t)$ , is random. If we assume that the electric field is generated by an ideal laser with constant power, then  $n(t)$  will follow a Poisson distribution

$$p(n) = \frac{\bar{n}^n \exp(-\bar{n})}{n!} \quad (7.20)$$

where  $n = n(t)$  and  $\bar{n} = \bar{n}(t)$ .

For each incident photon, there is a probability  $\eta$  of generating a phototelectron. Therefore, we can model the generation of photoelectrons during this time interval, as a binomial process where the number of events is equal to the number of incident photons,  $n(t)$ , and the rate of success is  $\eta$ . If we combine the two random processes, binomial photoelectron generation after poissonian photon flux, the number of output photoelectrons in this time interval,  $m(t)$ , will follow [1]

$$m \sim \text{Poisson}(\eta\bar{n}) \quad (7.21)$$

with  $\bar{m} = \eta\bar{n}$  where  $m = m(t)$ .

## Functional description

The input of this block is the electric field amplitude,  $A(t)$ , with sampling period  $T$ . The first step consists on the calculation the instantaneous power. Given that the input amplitude is a baseband representation of the original signal, then  $P(t) = 4|A(t)|^2$ . From this result, the average number of photons  $\bar{n}(t) = TP(t)\lambda/hc$ .

If the shot-noise is negleted, then the output number of photoelectrons,  $n_e(t)$  in the interval, will be equal to

$$m(t) = \eta \bar{n}(t) \quad (7.22)$$

If the shot-noise is considered, then the output fluctuations will be simulated by generating a value from a Poissonian random number generator with mean  $\eta \bar{n}(t)$

$$m(t) \sim \text{Poisson} \left( \eta \bar{n}(t) \right) \quad (7.23)$$

## Input Parameters

Parameter	Default Value	Description
efficiency	1.0	Photodiode's quantum efficiency.
shotNoise	false	Shot-noise off/on.

## Methods

PhotoelectronGenerator()

PhotoelectronGenerator(vector<Signal \*> &InputSig, vector<Signal \*> &OutputSig)  
:Block(InputSig, OutputSig)

void initialize(void)

bool runBlock(void)

void setEfficiency(t\_real efficiency)

void getEfficiency()

void setShotNoise(bool shotNoise)

void getShotNoise()

## Input Signals

**Number:** 1

**Type:** Optical (OpticalSignal)

## Output Signals

**Number:** 1

**Type:** Electrical (TimeDiscreteAmplitudeContinuousReal  
TimeContinuousAmplitudeContinuousReal) or

## Examples

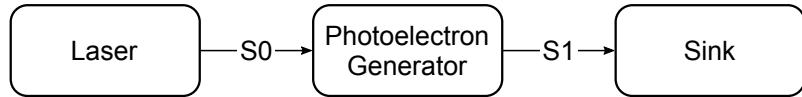


Figure 7.21: Constant power simulation setup

To test the output of this block, we recreated the results of figure 11.2 – 3 in [1]. We started by simulating the constant optical power case, in which the local oscillator power was fixed to a constant value. Two power levels were tested,  $P = 1\mu W$  and  $P = 1nW$ , using a sample period of 20 picoseconds and photoelectron generator efficiency of 1.0. The simulation code is in folder `lib \photoelectron_generator \photoelectron_generator_test_constant`. The following plots show the number of output electrons per sample when the shot noise is ignored or considered

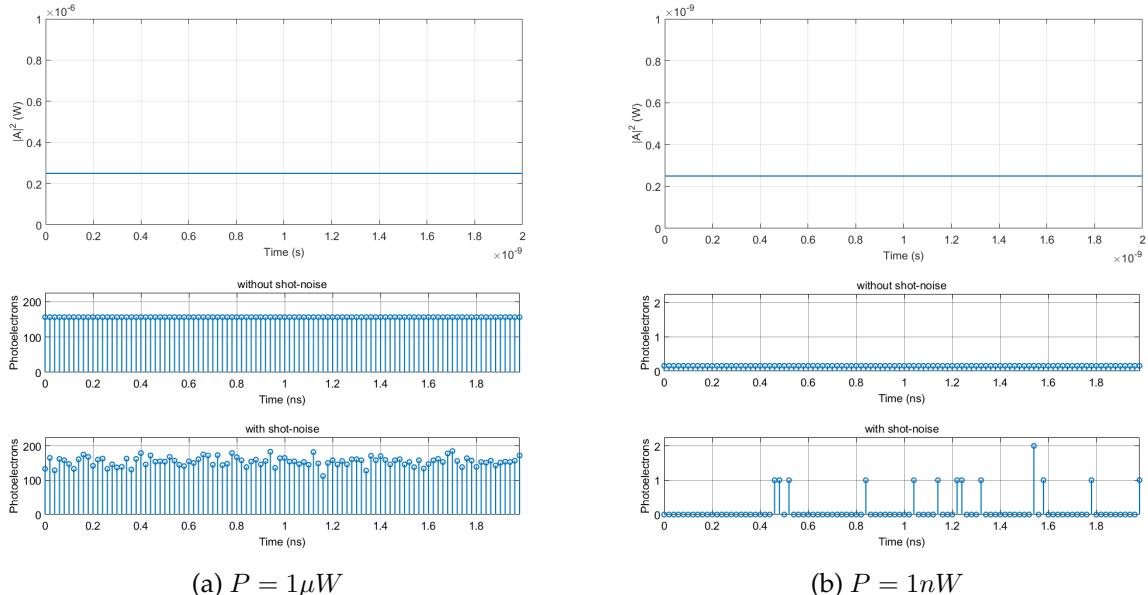


Figure 7.22: Upper plots: input optical squared amplitude on the photoelectron generator; Middle plots: number of output photoelectrons per sample neglecting quantum noise; Bottom plots: number of output photoelectrons per sample considering quantum noise.

Figure 7.22 shows clearly that turning the quantum noise on or off will produce a signal with or without variance, as predicted. If we compare this result with plot (a) in [1], in particular  $P = 1nW$ , we see that they are in conformance, with a slight difference, where a sample has more than one photoelectron. In contrast with the reference result, where only single events are represented, the  $P = 1\mu W$  case shows that all samples account many photoelectrons. Given it's input power, multiple photoelectron generation events will occur during the sample time window. Therefore, to recreate the reference result, we just need to reduce the sample period until the probability of generating more than 1 photoelectron per sample goes to 0.

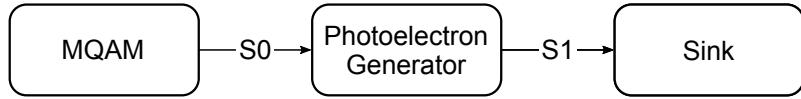


Figure 7.23: Variable power simulation setup

To recreate plot (b) in [1], a more complex setup was used, where a series of states are generated and shaped by a MQAM, creating a input electric field with time-varying power. The simulation code is in folder lib \photoelectron\_generator \photoelectron\_generator\_variable.

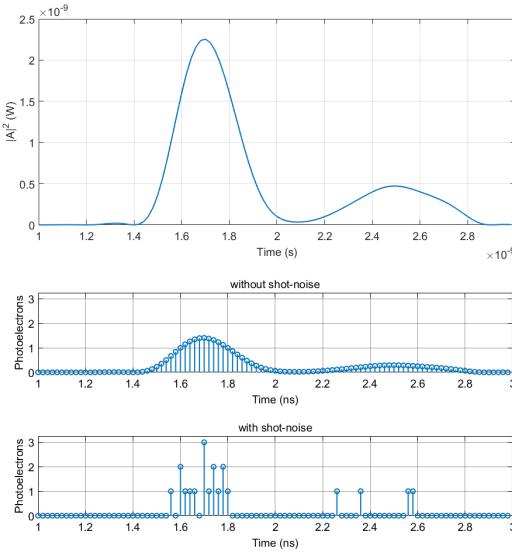


Figure 7.24: Upper plots: input optical squared amplitude on the photoelectron generator; Middle plots: number of output photoelectrons per sample neglecting quantum noise; Bottom plots: number of output photoelectrons per sample considering quantum noise.

Figure 7.24 shows that the output without shot-noise is following the input power perfectly, apart from a constant factor. In the case with shot-noise, we see that there are only output samples in high power input samples. These results are not following strictly plot (b) in [1], because has already discussed previously, in the high power input samples, we have a great probability of generating many photoelectrons.

### Sugestions for future improvement

## References

- [1] Bahaa E. A. Saleh and Malvin Carl Teich. *Fundamentals of photonics*. Wiley series in pure and applied optics. New York (NY): John Wiley & Sons, 1991.

## 7.34 Pulse Shaper

<b>Header File</b>	:	pulse_shaper.h
<b>Source File</b>	:	pulse_shaper.cpp

This block applies an electrical filter to the signal. It accepts one input signal that is a sequence of Dirac delta functions and it produces one output signal continuous in time and in amplitude.

### Input Parameters

Parameter	Type	Values	Default
filterType	string	RaisedCosine, Gaussian	RaisedCosine
impulseResponseTimeLength	int	any	16
rollOffFactor	real	$\in [0, 1]$	0.9

Table 7.18: Pulse shaper input parameters

### Methods

```

PulseShaper(vector<Signal * > &InputSig, vector<Signal * > OutputSig)
:FIR_Filter(InputSig, OutputSig){};

void initialize(void);

void setImpulseResponseTimeLength(int impResponseTimeLength)

int const getImpulseResponseTimeLength(void)

void setFilterType(PulseShaperFilter fType)

PulseShaperFilter const getFilterType(void)

void setRollOffFactor(double rOffFactor)

double const getRollOffFactor()

```

### Functional Description

The type of filter applied to the signal can be selected through the input parameter *filterType*. Currently the only available filter is a raised cosine.

The filter's transfer function is defined by the vector *impulseResponse*. The parameter *rollOffFactor* is a characteristic of the filter and is used to define its transfer function.

## Input Signals

**Number** : 1

**Type** : Sequence of Dirac Delta functions (ContinuousTimeDiscreteAmplitude)

## Output Signals

**Number** : 1

**Type** : Sequence of impulses modulated by the filter (ContinuousTimeContinuousAmplitude)

## Example

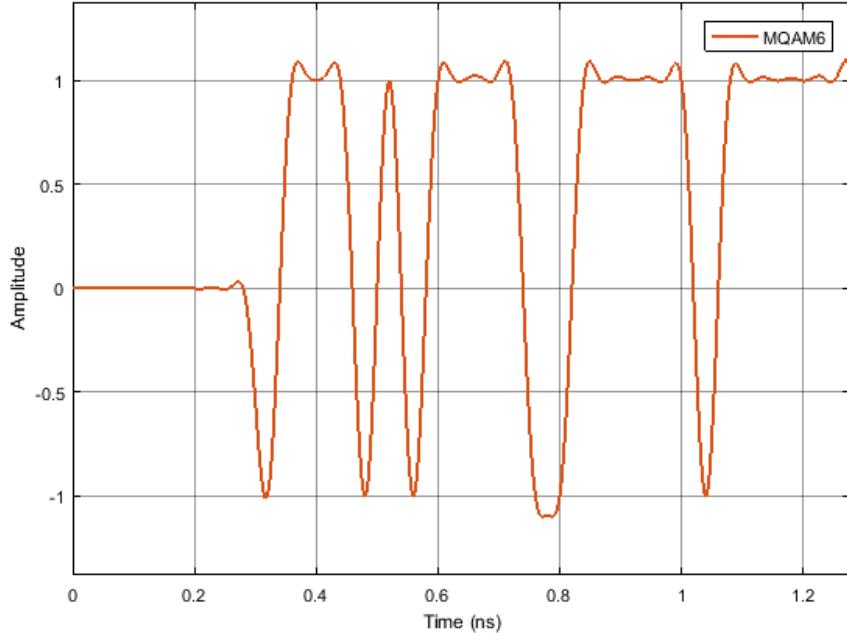


Figure 7.25: Example of a signal generated by this block for the initial binary signal 0100...

## Sugestions for future improvement

Include other types of filters.

## 7.35 Quantizer

<b>Header File</b>	:	quantizer_*.h
<b>Source File</b>	:	quantizer_*.cpp
<b>Version</b>	:	20180423 (Celestino Martins)

This block simulates a quantizer, where the signal is quantized into discrete levels. Given a quantization bit precision, *resolution*, the outputs signal will be comprise  $2^{nBits} - 1$  levels.

### Input Parameters

Parameter	Unity	Type	Values	Default
resolution	bits	double	any	<i>inf</i>
maxValue	volts	double	any	1.5
minValue	volts	double	any	-1.5

Table 7.19: Quantizer input parameters

### Methods

```

Quantizer() ;

Quantizer(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig,
OutputSig){};

void initialize(void);

bool runBlock(void);

void setSamplingPeriod(double sPeriod) samplingPeriod = sPeriod;

void setSymbolPeriod(double sPeriod) symbolPeriod = sPeriod;

void setResolution(double nbBits) resolution = nbBits;

double getResolution() return resolution;

void setMinValue(double maxValue) maxValue = maxValue;

double getMinValue() return maxValue;

void setMaxValue(double maxValue) maxValue = maxValue;

double getMaxValue() return maxValue;

```

### Functional description

This block can performs the signal quantization according to the defined input parameter *resolution*.

Firstly, the parameter *resolution* is checked and if it is equal to the infinity, the output signal correspond to the input signal. Otherwise, the quantization process is applied. The input signal is quantized into  $2^{\text{resolution}-1}$  discrete levels using the standard *round* function.

**Input Signals**

**Number:** 1

**Output Signals**

**Number:** 1

**Type:** Electrical complex signal

**Examples**

**Sugestions for future improvement**

## 7.36 Resample

<b>Header File</b>	:	resample_*.h
<b>Source File</b>	:	resample_*.cpp
<b>Version</b>	:	20180423 (Celestino Martins)

This block simulates the resampling of a signal. It receives one input signal and outputs a signal with the sampling rate defined by sampling rate, which is externally configured.

### Input Parameters

Parameter	Type	Values	Default
rFactor	double	any	<i>inf</i>
samplingPeriod	double	any	0.0
symbolPeriod	double	any	1.5

Table 7.20: Resample input parameters

### Methods

```
Resample() ; Resample(vector<Signal *> &InputSig, vector<Signal *> &OutputSig)
:Block(InputSig, OutputSig){};
void initialize(void); bool runBlock(void);
void setSamplingPeriod(double sPeriod) samplingPeriod = sPeriod; void
setSymbolPeriod(double sPeriod) symbolPeriod = sPeriod;
void setOutRateFactor(double OUTsRate) rFactor = OUTsRate; double
getOutRateFactor() return rFactor;
```

### Functional description

This block can performs the signal resample according to the defined input parameter *rFactor*. It resamples the input signal at *rFactor* times the original sample rate.

Firstly, the parameter *nBits* is checked and if it is greater than 1 it is performed a linear interpolation, increasing the input signal original sample rate to *rFactor* times.

**Input Signals**

**Number:** 1

**Output Signals**

**Number:** 1

**Type:** Electrical complex signal

**Examples**

**Sugestions for future improvement**

## 7.37 Sampler

<b>Header File</b>	:	sampler.h
<b>Source File</b>	:	sampler_20171119.cpp

This block can work in two configurations: with an external clock or without it. In the latter it accepts two input signals one being the clock and the other the signal to be demodulated. In the other configuration there's only one input signal which is the signal.

The output signal is obtained by sampling the input signal with a predetermined sampling rate provided either internally or by the clock.

### Input Parameters

Parameter	Type	Values	Default
samplesToSkip	int	any (smaller than the number of samples generated)	0

Table 7.21: Sampler input parameters

### Methods

Sampler()

Sampler(vector<Signal \*> &InputSig, vector<Signal \*> &OutputSig) :Block(InputSig, OutputSig)

void initialize(void)

bool runBlock(void)

void setSamplesToSkip(t\_integer sToSkip)

### Functional description

This block can work with an external clock or without it.

In the case of having an external clock it accepts two input signals. The signal to be demodulate which is complex and a clock signal that is a sequence of Dirac delta functions with a predetermined period that corresponds to the sampling period. The signal and the clock signal are scanned and when the clock has the value of 1.0 the correspondent complex value of the signal is placed in the buffer corresponding to the output signal.

There's a detail worth noting. The electrical filter has an impulse response time length of 16 (in units of symbol period). This means that when modulating a bit the spike in the signal corresponding to that bit will appear 8 units of symbol period later. For this reason there's

the need to skip the earlier samples of the signal when demodulating it. That's the purpose of the *samplesToSkip* parameter.

Between the binary source and the current block the signal is filtered twice which means that this effect has to be taken into account twice. Therefore the parameter *samplesToSkip* is given by  $2 * 8 * \text{samplesPerSymbol}$ .

## Input Signals

**Number:** 1

**Type:** Electrical real (TimeContinuousAmplitudeContinuousReal)

## Output Signals

**Number:** 1

**Type:** Electrical real (TimeDiscreteAmplitudeContinuousReal)

## Examples

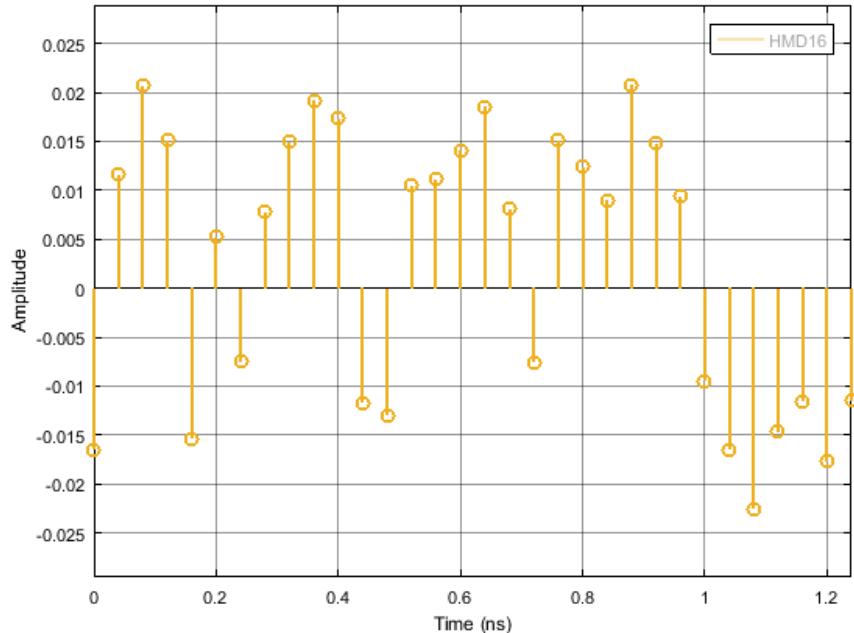


Figure 7.26: Example of the output signal of the sampler

## Sugestions for future improvement

### 7.38 SNR of the Photoelectron Generator

<b>Header File</b>	:	srn_photoelectron_generator_*.h
<b>Source File</b>	:	snr_photoelectron_generator_*.cpp
<b>Version</b>	:	20180309 (Diamantino Silva)

This block estimates the signal to noise ratio (SNR) of a input stream of photoelectrons, for a given time window.

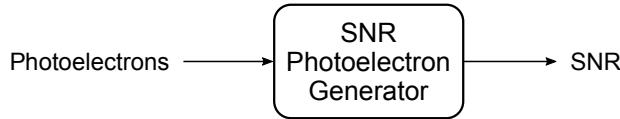


Figure 7.27: Schematic representation of the SNR of the Photoelectron Generator code block

#### Theoretical description

The input of this block is a stream of samples,  $y_j$ , each one of them corresponding to a number of photoelectrons generated in a time interval  $\Delta t$ . These photoelectrons are usually the output of a photodiode (photoelectron generator). To calculate the SNR of this stream, we will use the definition used in [1]

$$\text{SNR} = \frac{\bar{n}^2}{\sigma_n^2} \quad (7.24)$$

in which  $\bar{n}$  is the mean value and  $\sigma_n^2$  is the variance of the photon number in a given time interval  $T$ .

To apply this definition to our input stream, we start by separate it's samples in contiguous time windows with duration  $T$ . Each time window  $k$  is defined as the time interval  $[kT, (k + 1)T[$ . To estimate the SNR for each time window, we will use the following estimators for the mean,  $\mu_k$ , and variance,  $s_k^2$  [2]

$$\mu_k = \langle y \rangle_k \quad s_k^2 = \frac{N}{N-1} \left( \langle y^2 \rangle_k - \langle y \rangle_k^2 \right) \quad (7.25)$$

where  $\langle y^n \rangle_k$  is the  $n$  moment of the  $k$  window given by

$$\langle y^n \rangle_k = \frac{1}{N_k} \sum_{j=j_{\min}(k)}^{j_{\max}(k)} y_j^n \quad (7.26)$$

in which

$$j_{min}(k) = \lceil t_k / \Delta t \rceil \quad (7.27)$$

$$j_{max}(k) = \lceil t_{k+1} / \Delta t \rceil - 1 \quad (7.28)$$

$$N_k = j_{max}(k) - j_{min}(k) + 1 \quad (7.29)$$

$$t_k = kT \quad (7.30)$$

where  $\lceil x \rceil$  is the ceiling function.

In our implementation, we define two variables,  $S_1(k)$  and  $S_2(k)$ , corresponding to the sum of the samples and the sum of the squares of the sample in the time interval  $k$ . These two sums are related to the moments as

$$S_1(k) = N_k \langle y \rangle_k \quad (7.31)$$

$$S_2(k) = N_k \langle y^2 \rangle_k \quad (7.32)$$

Using these two variables, we can rewrite  $\mu_k$  and  $s_k^2$  as

$$\mu_k = \frac{S_1(k)}{N_k} \quad s_k^2 = \frac{1}{N_k - 1} \left( S_2(k) - \frac{1}{N_k} (S_1(k))^2 \right) \quad (7.33)$$

The signal to noise ratio of the time interval  $k$ ,  $\text{SNR}_k$ , can be expressed as

$$\text{SNR}_k = \frac{\mu_k^2}{\sigma_k^2} = \frac{N_k - 1}{N_k} \frac{(S_1(k))^2}{N_k S_2(k) - (S_1(k))^2} \quad (7.34)$$

One particularly important case is the phototransistor stream resulting from the conversion of a laser photon stream by a photodiode (photoelectron generator). The resulting SNR will be [1]

$$\text{SNR} = \eta \bar{n} \quad (7.35)$$

in which  $\eta$  is the photodiode quantum efficiency.

## Functional description

This block is designed to operate in time windows, dividing the input stream in contiguous sets of samples with a duration  $t_{\text{Window}} = T$ . For each time window, the general process consists in accumulating the input sample values and the square of the input sample values, and calculating the SNR of the time window based on these two variables.

To process this accumulation, the block uses two state variables, `aux_sum1` and `aux_sum2`, which hold the accumulation of the sample values and accumulation of the square of sample values, respectively.

The block starts by calculating the number of samples it has to process for the current time window, using equations 7.28, 7.29 and 7.30. If the duration of  $t_{\text{Window}}$  is 0, then we assume that this time window has infinite time (infinite samples). The values of `aux_sum1` and `aux_sum2` are set to 0, and the processing of the samples of current window begins.

After processing all the samples of the time window, we obtain  $S_1(k)$  and  $S_2(k)$  from the

state variables as  $S_1(k) = \text{aux\_sum1}$  and  $S_2(k) = \text{aux\_sum2}$ , and proceed to the calculation of the  $\text{SNR}_k$ , using equation 7.34.

If the simulation ends before reaching the end of the current time window, we calculate the  $\text{SNR}_k$ , using the current values of `aux_sum1`, `aux_sum2` for  $S_1(k)$  and  $S_2(k)$ , and the number of samples already processed, `currentWindowSample`, for  $N_k$ .

## Input Parameters

Parameter	Default Value	Description
windowTime	0	SNR time window.

## Methods

`SnrPhotoelectronGenerator()`

`SnrPhotoelectronGenerator(vector<Signal *> &InputSig, vector<Signal *> &OutputSig)`  
`:Block(InputSig, OutputSig)`

`void initialize(void)`

`bool runBlock(void)`

`void setTimeWindow(t_real timeWindow)`

## Input Signals

**Number:** 1

**Type:** Electrical (TimeDiscreteAmplitudeContinuousReal)

## Output Signals

**Number:** 1

**Type:** Electrical (TimeDiscreteAmplitudeContinuousReal)

## Examples

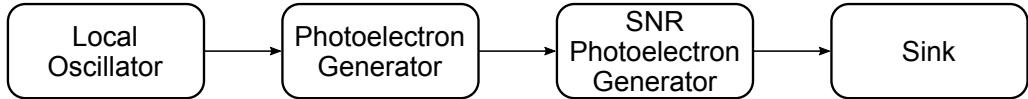


Figure 7.28: Simulation setup

To confirm the block's correct output, we have designed a simulation setup which calculates the SNR of a stream of photoelectrons generated by the detection of a laser photon stream by a photodiode.

The simulation has three main parameters, the power of the local oscillator,  $P_{LO}$ , the duration of the time window,  $T$ , and the photodiode's quantum efficiency,  $\eta$ . For each combination of these three parameters, the simulation generates 1000 SNR samples, during which all parameters stay constant. The final result is the average of these SNR samples. The simulations were performed with a sample time  $\Delta t = 10^{-10}s$ .

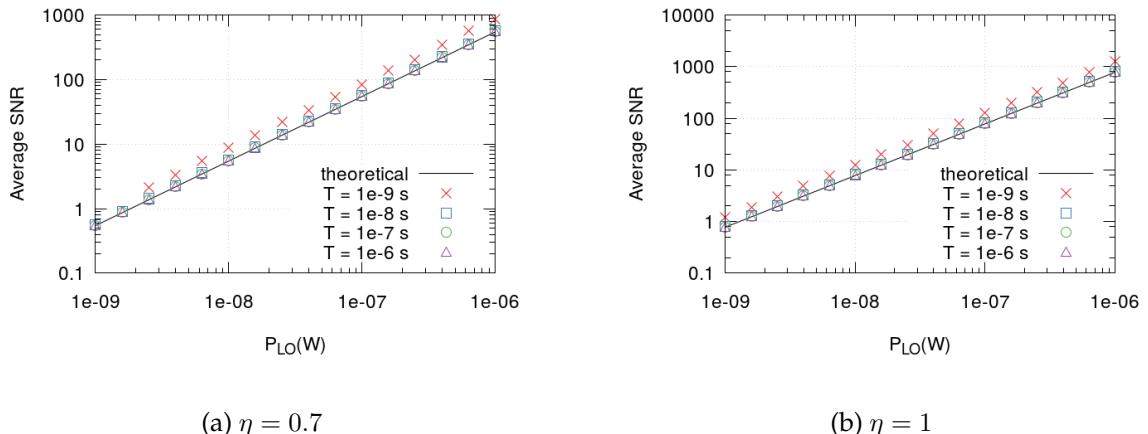


Figure 7.29: Theoretical and simulated results of the average SNR, for two photodiode efficiencies.

The plots in 7.29 show the comparison between the theoretical result 7.35 and the simulation results. We see that for low values of  $T$ , the average SNR shows a systematic deviation from the theoretical result, but for  $T > 10^{-6}s$  (10000 samples per time window), the simulation result shows a very good agreement with the theoretical result.

The simulations also show a lack of average SNR results when low power, low efficiency and small time window are combined (see plot 7.29a). This is because in those conditions, the probability of having a time windows with no photoelectrons, creating an invalid SNR, is very high, which will prevent the calculation of the average SNR.

We can estimate the probability of calculating a valid SNR average by calculating the probability of no time window having 0 photoelectrons,  $p_{ave} = (1 - q)^M$ , in which  $q$  is the probability of a time window having all its samples equal to 0 and  $M$  is the number of time windows. We know that the input stream follows a Poisson distribution with mean  $\bar{m}$ , therefore  $q = (\exp(-\bar{m}))^N$ , in which  $\bar{m} = \eta P \lambda / hc$  and  $N = T / \Delta t$ , is the average number of samples per time window. Using this result, we obtain the probability of calculating a valid SNR average as

$$p_{ave} = (1 - \exp(-N\bar{m}))^M \quad (7.36)$$

## Block problems

### Future work

The block could also output a confidence interval for the calculated SNR. Given that the output of the Photoelectron Generator follows a Poissonian distribution when the shot noise is on, the article "Confidence intervals for signal to noise ratio of a Poisson distribution" by Florence George and B.M. Kibria [3], could be used as a reference to implement such feature.

## References

- [1] Bahaa E. A. Saleh and Malvin Carl Teich. *Fundamentals of photonics*. Wiley series in pure and applied optics. New York (NY): John Wiley & Sons, 1991.
- [2] S. W. Smith et al. *The scientist and engineer's guide to digital signal processing*. California Technical Pub. San Diego, 1997.
- [3] G. Florence and K. B. Golam. "Confidence intervals for signal to noise ratio of a Poisson distribution". In: *American Journal of Biostatistics* 2.2 (2011), p. 44.

## 7.39 Sink

<b>Header File</b>	:	sink.h
<b>Source File</b>	:	sink.cpp

This block accepts one input signal and it does not produce output signals. It takes samples out of the buffer until the buffer is empty. It has the option of displaying the number of samples still available.

### Input Parameters

Parameter	Type	Values	Default
numberOfSamples	long int	any	-1

Table 7.22: Sampler input parameters

### Methods

`Sink(vector<Signal *> &InputSig, vector<Signal *> &OutputSig)`

`bool runBlock(void)`

`void setNumberOfSamples(long int nOfSamples)`

`void setDisplayNumberOfSamples(bool opt)`

### Functional Description

## 7.40 White Noise

<b>Header File</b>	:	white_noise_20180420.h
<b>Source File</b>	:	white_noise_20180420.cpp

This block generates a gaussian pseudo-random noise signal with a given spectral density. It can be initialized with three different seeding methods to allow control over correlation and reproducibility:

1. DefaultDeterministic
2. RandomDevice
3. Selected

This block does not accept any input signal. It produces can produce a real or complex output, depending on the used output signal.

### Input Parameters

Parameter	Type	Values	Default
seedType	enum	DefaultDeterministic, RandomDevice, Selected	RandomDevice
spectralDensity	real	$> 0$	$1.5 \times 10^{-17}$
seed	int	$\in [1, 2^{32} - 1]$	1
samplingPeriod	double	$> 0$	1.0

Table 7.23: White noise input parameters

### Methods

```
WhiteNoise(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig);
```

```
void initialize(void);

bool runBlock(void);

void setNoiseSpectralDensity(double SpectralDensity) spectralDensity = SpectralDensity;

double const getNoiseSpectralDensity(void) return spectralDensity;

void setSeedType(SeedType sType) seedType = sType; ;
```

```

SeedType const getSeedType(void) return seedType; ;

void setSeed(int newSeed) seed = newSeed;

int getSeed(void) return seed;

```

### Functional description

The *seedType* parameter allows the user to select between one of the three seeding methods to initialize the pseudo-random number generators (PRNGs) responsible for generating the noise signal.

**DefaultDeterministic:** Uses default seeds to initialize the PRNGs. These are different for all generators used within the same block, but remain the same for sequential runs or different *white\_noise* blocks. Therefore, if more than one *white\_noise* block is used, another seeding method should be chosen to avoid producing the exact same noise signal in all sources.

**RandomDevice:** Uses randomly chosen seeds using *std::random\_device* to initialize the PRNGs.

**SingleSelected:** Uses one user selected seed to initialize the PRNGs. The selected seed is passed through the variable *seed*. If more than one generator is used, additional seeds are created by choosing the next sequential integers. For instance, if the user selected seed is 10, and all the four PRNGs are used, the used seeds will be [10, 11, 12, 13].

The noise is obtained from a gaussian distribution with zero mean and a given variance. The variance is equal to the noise power, which can be calculated from the spectral density  $n_0$  and the signal's bandwidth  $B$ , where the bandwidth is obtained from the defined sampling time  $T$ .

$$N = n_0 B = n_0 \frac{2}{T} \quad (7.37)$$

If the signal is complex, the noise is calculated independently for the real and imaginary parts, and the spectral density value is divided by two, to account for the two-sided noise spectral density.

### Input Signals

**Number:** 0

### Output Signals

**Number:** 1 or more

**Type:** RealValue, ComplexValue or ComplexValueXY

**Examples**

**Random Mode**

**Suggestions for future improvement**

## 7.41 Ideal Amplifier

This block has one input signal and one output signal both corresponding to electrical signals. The output signal is a perfect amplification of the input signal.

### Input Parameters

Parameter	Type	Values	Default
gain	double	any	$1 \times 10^4$

Table 7.24: Ideal Amplifier input parameters

### Methods

`IdealAmplifier()`

```
IdealAmplifier(vector<Signal * > &InputSig, vector<Signal * > &OutputSig)
:Block(InputSig, OutputSig);

void initialize(void);

bool runBlock(void);

void setGain(double ga) gain = ga;

double getGain() return gain;
```

### Functional description

The output signal is the product of the input signal with the parameter *gain*.

### **Input Signals**

**Number:** 1

**Type:** Electrical (TimeContinuousAmplitudeContinuousReal)

### **Output Signals**

**Number:** 1

**Type:** Electrical (TimeContinuousAmplitudeContinuousReal)

### **Examples**

### **Sugestions for future improvement**

## Chapter 8

---

### Mathlab Tools

## 8.1 Generation of AWG Compatible Signals

<b>Students Name</b>	:	Francisco Marques dos Santos
	:	Romil Patel
<b>Goal</b>	:	Convert simulation signals into waveform files compatible with the laboratory's Arbitrary Waveform Generator
<b>Version</b>	:	sgnToWfm.m ( <b>Student Name</b> : Francisco Marques dos Santos) : sgnToWfm_20171119.m ( <b>Student Name</b> : Romil Patel)

This section shows how to convert a simulation signal into an AWG compatible waveform file through the use of a matlab function called sgnToWfm. This allows the application of simulated signals into real world systems.

### 8.1.1 sgnToWfm.m

#### Structure of a function

```
[data, symbolPeriod, samplingPeriod, type, numberOfSymbols, samplingRate] =  
sgnToWfm(fname_sgn, nReadr, fname_wfm);
```

#### Inputs

**fname\_sgn**: Input filename of the signal (\*.sgn) you want to convert. It must be a real signal (Type: TimeContinuousAmplitudeContinuousReal).

**nReadr**: Number of symbols you want to extract from the signal.

**fname\_wfm**: Name that will be given to the waveform file.

#### Outputs

A waveform file will be created in the Matlab current folder. It will also return six variables in the workspace which are:

**data**: A vector with the signal data.

**symbolPeriod**: Equal to the symbol period of the corresponding signal.

**samplingPeriod**: Sampling period of the signal.

**type**: A string with the name of the signal type.

**numberOfSymbols**: Number of symbols retrieved from the signal.

**samplingRate**: Sampling rate of the signal.

## Functional Description

This matlab function generates a \*.wfm file given an input signal file (\*.sgn). The waveform file is compatible with the laboratory's Arbitrary Waveform Generator (Tektronix AWG70002A). In order to recreate it appropriately, the signal must be real, not exceed  $8 \times 10^9$  samples and have a sampling rate equal or below 16 GS/s.

### This function can be called with one, two or three arguments:

Using one argument:

```
[ data, symbolPeriod, samplingPeriod, type, numberOfSymbols, samplingRate] =  
sgnToWfm('S6.sgn');
```

This creates a waveform file with the same name as the \*.sgn file and uses all of the samples it contains.

Using two arguments:

```
[ data, symbolPeriod, samplingPeriod, type, numberOfSymbols, samplingRate] =  
sgnToWfm('S6.sgn',256);
```

This creates a waveform file with the same name as the signal file name and the number of samples used equals nReadr x samplesPerSymbol. The samplesPerSymbol constant is defined in the \*.sgn file.

Using three arguments:

```
[ data, symbolPeriod, samplingPeriod, type, numberOfSymbols, samplingRate] =  
sgnToWfm('S6.sgn',256,'myWaveform.wfm');
```

This creates a waveform file with the name "myWaveform" and the number of samples used equals nReadr x samplesPerSymbol. The samplesPerSymbol constant is defined in the \*.sgn file.

### 8.1.2 sgnToWfm\_20171121.m

#### Structure of a function

```
[dataDecimate, data, symbolPeriod,  
samplingPeriod, type, numberOfSymbols, samplingRate, samplingRateDecimate] =  
sgnToWfm_20171121(fname_sgn, nReadr, fname_wfm)
```

#### Inputs

Same as discussed above in the file sgnToWfm.m.

## Outputs

The output of the function sgnToWfm\_20171121.m contains eight different parameters. Among those eight different parameters, six output parameters are the same as discussed above in the function sgnToWfm.m and remaining two parameters are the following:

**dataDecimate:** A vector which contains decimated signal data by an appropriate decimation factor to make it compatible with the AWG.

**samplingRateDecimate:** Reduced sampling rate which is compatible with AWG. (i.e. less than 16 GSa/s).

«««< HEAD

## Functional Description

The functional description is same as discussed above in sgnToWfm.m. =====

## Outputs

The output of the function version 20171121 contains eight different parameters. Among those eight parameters, six output parameters are the same as discussed above in the version 20170930 and remaining two parameters are the following:

Name of output signals	Description
<b>dataDecimate</b>	A vector which contains decimated signal data by an appropriate decimation factor to make it compatible with the AWG.
<b>samplingRateDecimate</b>	Reduced sampling rate which is compatible with AWG. (i.e. less than 16 GSa/s).

### Decimation factor calculation

The flowchart for calculating the decimation factor is as follows:

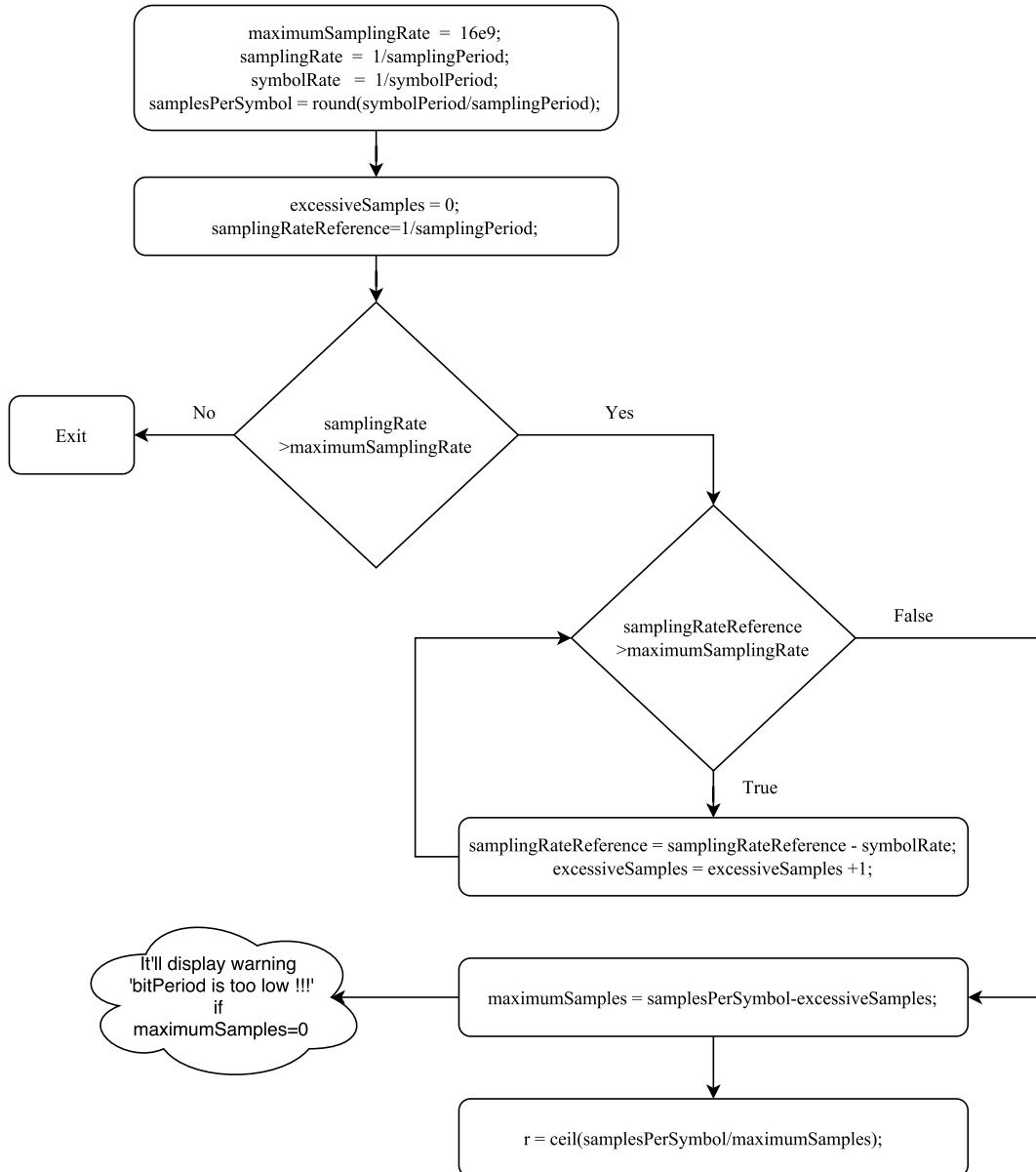


Figure 8.1: Flowchart to calculate decimation factor

»»»» Develop.Romil

#### 8.1.3 Loading a signal to the Tektronix AWG70002A

The AWG we will be using is the Tektronix AWG70002A which has the following key specifications:

**Sampling rate up to 16 GS/s:** This is the most important characteristic because it determines the maximum sampling rate that your signal can have. It must not be over 16 GS/s or else the AWG will not be able to recreate it appropriately.

**8 GSample waveform memory:** This determines how many data points your signal can have.

After making sure this specifications are respected you can create your waveform using the function. When you load your waveform, the AWG will output it and repeat it constantly until you stop playing it.

**1. Using the function `sgnToWfm`:** Start up Matlab and change your current folder to mtools and add the signals folder that you want to convert to the Matlab search path. Use the function accordingly, putting as the input parameter the signal file name you want to convert.

**2. AWG sampling rate:** After calling the function there should be waveform file in the mtools folder, as well as a variable called samplingRate in the Matlab workspace. Make sure this is equal or bellow the maximum sampling frequency of the AWG (16 GS/s), or else the waveform can not be equal to the original signal. If it is higher you have to adjust the parameters in the simulation in order to decrease the sampling frequency of the signal(i.e. decreasing the bit period or reducing the samples per symbol).

**3. Loading the waveform file to the AWG:** Copy the waveform file to your pen drive and connect it to the AWG. With the software of the awg open, go to browse for waveform on the channel you want to use, and select the waveform file you created (Figure 7.1).

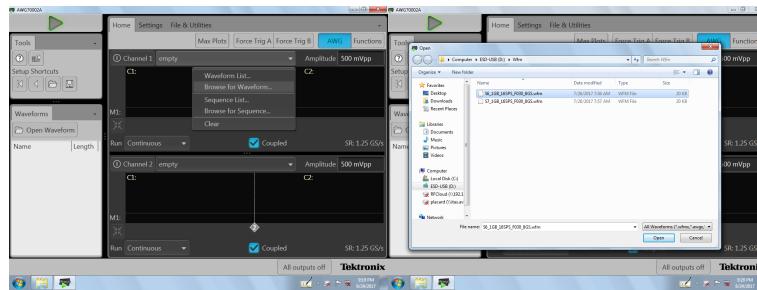


Figure 8.2: Selecting your waveform in the AWG

Now you should have the waveform displayed on the screen. Although it has the same shape, the waveform might not match the signal timing wise due to an incorrect sampling rate configured in the AWG. In this example (Figure 7.2), the original signal has a sample rate of 8 GS/s and the AWG is configured to 1.25 GS/s. Therefore it must be changed to the correct value. To do this go to the settings tab, clock settings, and change the sampling rate to be equal to the one of the original signal, 8 GS/s (Figure 7.3). Compare the waveform in the AWG with the original signal, they should be identical (Figure 7.4).

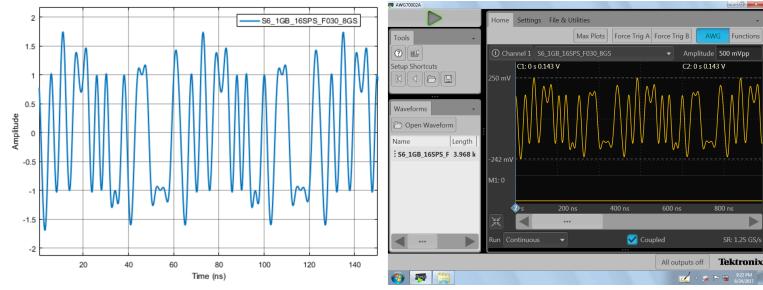


Figure 8.3: Comparison between the waveform in the AWG and the original signal before configuring the sampling rate

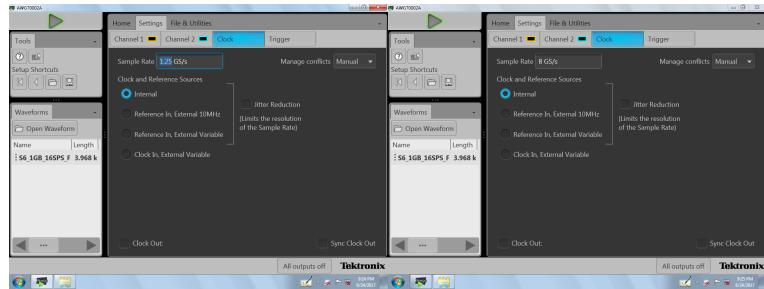


Figure 8.4: Configuring the right sampling rate

**4. Generate the signal:** Output the wave by enabling the channel you want and clicking on the play button.

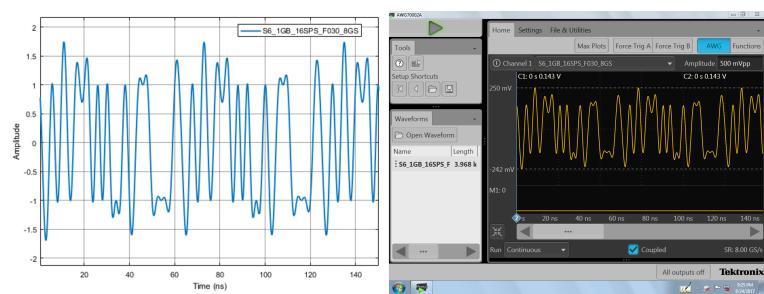


Figure 8.5: Comparison between the waveform in the AWG and the original signal after configuring the sampling rate

## Chapter 9

---

## Algorithms

## 9.1 Fast Fourier Transform

<b>Header File</b>	:	fft_*.h
<b>Source File</b>	:	fft_*.cpp
<b>Version</b>	:	20180201 (Romil Patel)

### Algorithm

The algorithm for the FFT will be implemented according with the following expression,

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{i2\pi kn/N} \quad 0 \leq k \leq N-1 \quad (9.1)$$

Similarly, for IFFT,

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{-i2\pi kn/N} \quad 0 \leq k \leq N-1 \quad (9.2)$$

From equations 9.1 and 9.2, we can write only one script for the implementations of the direct and inverse Discrete Fourier Transfer and manipulate its functionality as a FFT or IFFT by applying an appropriate input arguments. The generalized form for the algorithm can be given as,

$$y = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x e^{m i2\pi kn/N} \quad 0 \leq k \leq N-1 \quad (9.3)$$

where,  $x$  is an input complex signal,  $y$  is the output complex signal and  $m$  equals 1 or -1 for FFT and IFFT, respectively. An optimized fft function is also implemented without the  $1/\sqrt{N}$  factor, see below in the optimized fft section.

### Function description

To perform FFT operation, the fft\_\*.h header file must be included and the input argument to the function can be given as follows,

$$y = fft(x, 1)$$

or

$$y = fft(x)$$

where  $x$  and  $y$  are of the C++ type vector<complex>. In a similar way, IFFT can be manipulated as,

$$x = fft(y, 1)$$

or

$$x = ifft(y)$$

### Flowchart

The figure 9.1 displays top level architecture of the FFT algorithm. If the length of the input signal is  $2^N$ , it'll execute Radix-2 algorithm otherwise it'll execute Bluestein algorithm [1]. The computational complexity of Radix-2 and Bluestein algorithm is  $O(N \log_2 N)$ , however, the computation of Bluestein algorithm involves the circular convolution which increases the number of computations. Therefore, to reduce the computational time it is advisable to work with the vectors of length  $2^N$  [2].

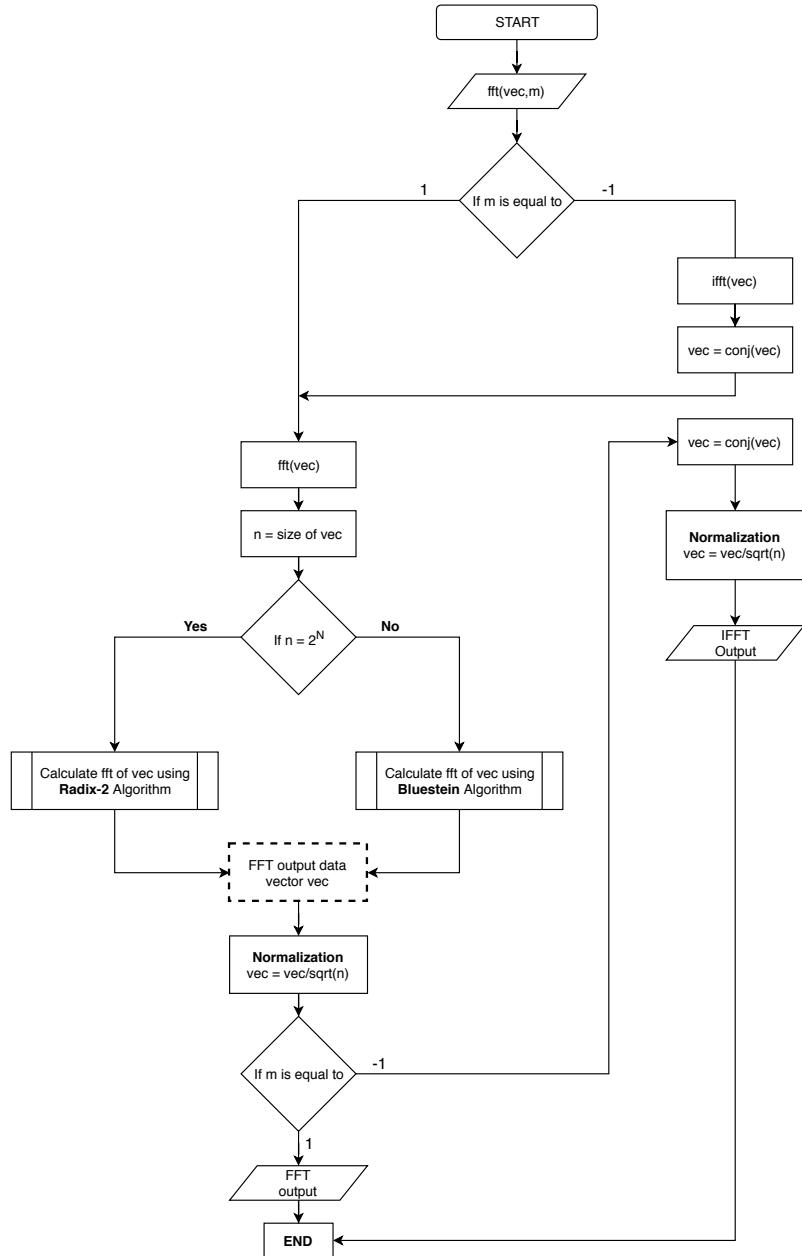


Figure 9.1: Top level architecture of FFT algorithm

### Radix-2 algorithm

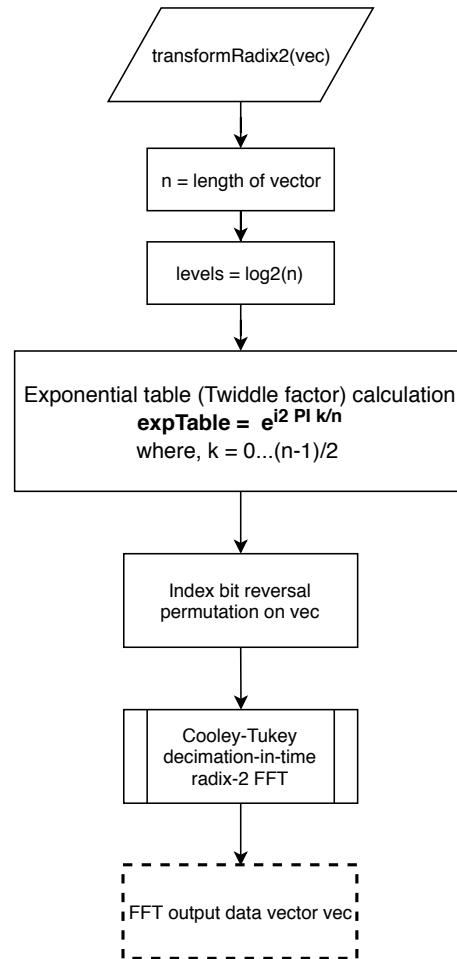


Figure 9.2: Radix-2 algorithm

## Cooley-Tukey algorithm

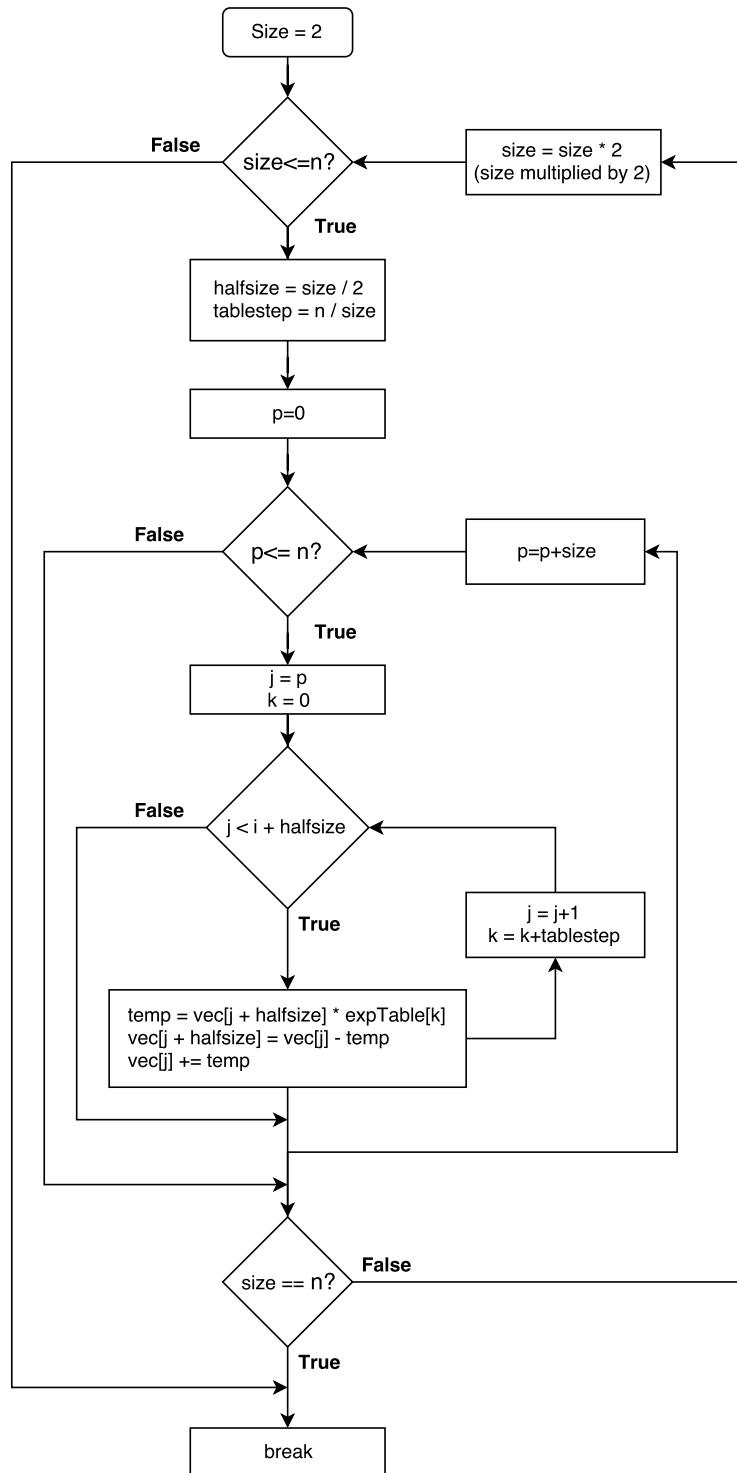


Figure 9.3: Cooley-Tukey algorithm

### Bluestein algorithm

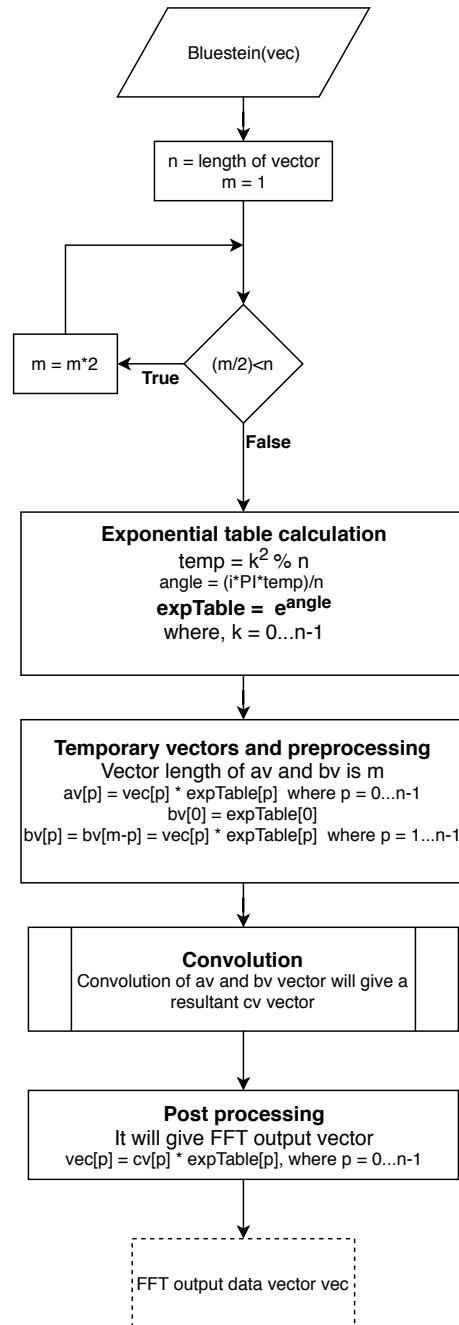


Figure 9.4: Bluestein algorithm

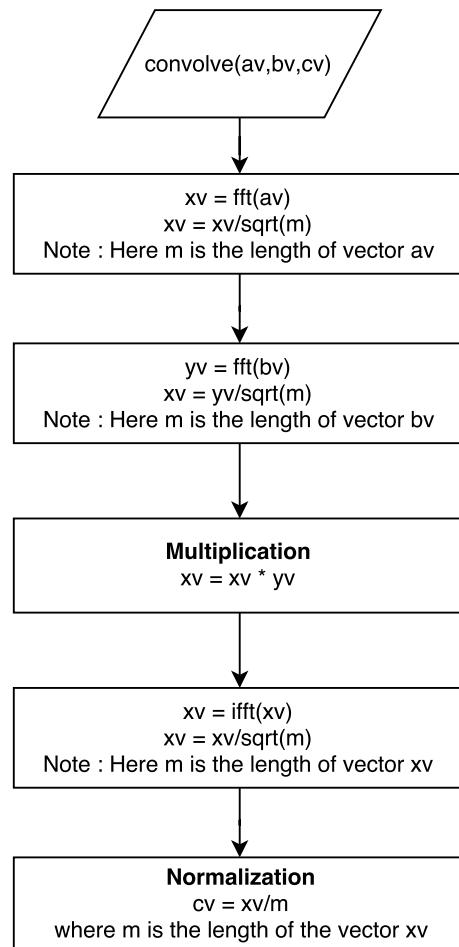
**Convolution algorithm**

Figure 9.5: Circular convolution algorithm

### Test example

This sections explains the steps to compare our C++ FFT program with the MATLAB FFT program.

**Step 1** : Open the **fft\_test** folder by following the path "/algorithms/fft/fft\_test".

**Step 2** : Find the **fft\_test.m** file and open it.

This **fft\_test.m** consists of two sections; section 1 generates the time domain signal and save it in the form of the text file with the name *time\_function.txt* in the same folder. Section 2 reads the fft complex data generated by C++ program.

```

%
%%%%%%%%%%%%%%%
2 %%%%%%%% SECTION 1
%
%
4 clc
5 clear all
6 close all

8 Fs = 1e5;           % Sampling frequency
9 T = 1/Fs;           % Sampling period
10 L = 2^10;           % Length of signal
11 t = (0:L-1)*(5*T); % Time vector
12 f = linspace(-Fs/2,Fs/2,L);

14 %Choose for sig a value between [1, 7]
15 sig = 7;
16 switch sig
17     case 1
18         signal_title = 'Signal with one signusoid and random noise';
19         S = 0.7*sin(2*pi*50*t);
20         X = S + 2*randn(size(t));
21     case 2
22         signal_title = 'Sinusoids with Random Noise';
23         S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
24         X = S + 2*randn(size(t));
25     case 3
26         signal_title = 'Single sinusoids';
27         X = sin(2*pi*t);
28     case 4
29         signal_title = 'Summation of two sinusoids';
30         X = sin(2*pi*t) + cos(2*pi*t);
31     case 5
32         signal_title = 'Single Sinusoids with Exponent';

```

```

34 X = sin(2*pi*200*t).*exp(-abs(70*t));
35 case 6
36     signal_title = 'Mixed signal 1';
37     X = sin(2*pi*10*t).*exp(-t)+sin(2*pi*t)+7*sin(2*pi*+5*t)+7*cos(2*pi*+20*t)+5*sin(2*pi*+50*t);
38 case 7
39     signal_title = 'Mixed signal 2';
40     X = 2*sin(2*pi*100*t).*exp(-t)+2.5*sin(2*pi*+250*t)+sin(2*pi*+50*t).*cos(2*pi*+20*t)+1.5*sin(2*pi*+50*t).*sin(2*pi*+150*t);
41 case 8
42     signal_title = 'Sinusoid tone';
43     X = cos(2*pi*100*t);
44 end
45
46 plot(t(1:end),X(1:end))
47 title ( signal_title )
48 axis([ min(t) max(t) 1.1*min(X) 1.1*max(X)]);
49 xlabel('t (s)')
50 ylabel('X(t)')
51 grid on
52
53 % dlmwrite will generate text file which represents the time domain signal.
54 % dlmwrite('time_function.txt', X, 'delimiter','\t');
55 fid=fopen('time_function.txt','w');
56 b=fprintf(fid ,'%0.15f\n',X); % 15-Digit accuracy
57 fclose(fid);
58
59 tic
60 fy = ifft (X)*sqrt(length(X));% According to the definition of "optical fft"
61 % with the (1/sqrt(N)) concept.
62 toc
63 fy = fftshift (fy);
64 figure(2);
65 subplot(2,1,1)
66 plot(f,abs(fy));
67 axis([-Fs/(2*5) Fs/(2*5) 0 1.1*max(abs(fy))]);
68 xlabel('f');
69 ylabel('|Y(f)|');
70 title ('MATLAB program Calculation : Magnitude');
71 grid on
72 subplot(2,1,2)
73 plot(f,phase(fy));
74 xlim([-Fs/(2*5) Fs/(2*5)]);
75 xlabel('f');
76 ylabel('phase(Y(f))');
77 title ('MATLAB program Calculation : Phase');
78 grid on
79
80 %%
81 %
82 %%%%%% SECTION 2

```

```

82 %
83 %%%%%%
84 %%%%%%
85 %%%%%%
86 %%%%%%
87 %%%%%%
88 %%%%%%
89 %%%%%%
90 %%%%%%
91 %%%%%%
92 %%%%%%
93 %%%%%%
94 %%%%%%
95 %%%%%%
96 %%%%%%
97 %%%%%%
98 %%%%%%
99 %%%%%%
100 %%%%%%
101 %%%%%%
102 %%%%%%
103 %%%%%%
104 %%%%%%
105 %%%%%%
106 %%%%%%
107 %%%%%%
108 %%%%%%
109 %%%%%%
110 %%%%%%
111 %%%%%%
112 %%%%%%
113 %%%%%%
114 %%%%%%
115 %%%%%%
116 %%%%%%
117 %%%%%%
118 %
119 %%%%%%
120 %%%%%%

```

Listing 9.1: fft\_test.m code

**Step 3 :** Choose for sig a value between [1, 7] and run the first section namely **section 1** by pressing "ctrl+Enter".

This will generate a *time\_function.txt* file in the same folder which contains the time domain signal data.

**Step 4 :** Now, find the `fft_test.vcxproj` file in the same folder and open it.

In this project file, find *fft\_test.cpp* and click on it. This file is an example of FFT calculation using C++ program. Basically this *fft\_test.cpp* file consists of four sections:

## Section 1. Read the input text file (import "time\_function.txt" data file)

## Section 2. It calculates FFT.

### Section 3. Save FFT calculated data (export *frequency\_function.txt* data file).

**Section 4.** Displays in the screen the FFT calculated data and length of the data.

```

1 # include "fft_20180208.h"
# include <complex>
3 # include <fstream>
# include <iostream>
5 # include <math.h>
# include <stdio.h>
# include <string>
# include <iostream>
# include <sstream>
# include <algorithm>
# include <vector>
11 #include <iomanip>

13 using namespace std;

15 int main()
{
17 ////////////////////////////////////////////////////////////////// Section 1 //////////////////////////////////////////////////////////////////
18 ////////////////////////////////////////////////////////////////// Read the input text file (import "time_function.txt") //////////////////////////////////////////////////////////////////
19 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
20
21 ifstream inFile;
22 inFile.precision(15);
23 double ch;
24 vector <double> inTimeDomain;
25 inFile.open("time_function.txt");
26 // First data (at 0th position) applied to the ch it is similar to the "cin".
27 inFile >> ch;
28 // It'll count the length of the vector to verify with the MATLAB
29 int count=0;
30 while (!inFile.eof()){
31     // push data one by one into the vector
32     inTimeDomain.push_back(ch);
33     // it'll increase the position of the data vector by 1 and read full vector.
34     inFile >> ch;
35     count++;
36 }
37 inFile.close(); // It is mandatory to close the file at the end.
38
39 ////////////////////////////////////////////////////////////////// Section 2 //////////////////////////////////////////////////////////////////
40 ////////////////////////////////////////////////////////////////// Calculate FFT //////////////////////////////////////////////////////////////////
41
42 vector <complex<double>> inTimeDomainComplex(inTimeDomain.size());

```

```

43 vector <complex<double>> fourierTransformed;
44 vector <double> re(inTimeDomain.size());
45 vector <double> im(inTimeDomain.size());

47 for (unsigned int i = 0; i < inTimeDomain.size(); i++)
48 {
49     re[i] = inTimeDomain[i]; // Real data of the signal
50 }
51
52 // Next, Real and Imaginary vector to complex vector conversion
53 inTimeDomainComplex = reImVect2ComplexVector(re, im);

55 // calculate FFT
56 clock_t begin = clock();
57 fourierTransformed = fft(inTimeDomainComplex);
58 clock_t end = clock();
59 double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;

61 ////////////////////////////////////////////////////////////////// Section 3 //////////////////////////////////////////////////////////////////
62 ////////////////////////////////////////////////////////////////// Save FFT calculated data (export "frequency_function.txt" ) //////////////////////////////////////////////////////////////////
63 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
64 ofstream outFile;
65 complex<double> outFileData;
66 outFile.open("frequency_function.txt");
67 outFile.precision(15);
68 for (unsigned int i = 0; i < fourierTransformed.size(); i++){
69     outFile << fourierTransformed[i].real() << endl;
70     outFile << fourierTransformed[i].imag() << endl;
71 }
72 outFile.close();

73 ////////////////////////////////////////////////////////////////// Section 4 //////////////////////////////////////////////////////////////////
74 ////////////////////////////////////////////////////////////////// Display Section //////////////////////////////////////////////////////////////////
75 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
76 for (unsigned int i = 0; i < fourierTransformed.size(); i++){
77     cout << fourierTransformed[i] << endl; // Display all FFT calculated data
78 }
79 cout << "\n\nTime elapsed to calculate FFT : " << elapsed_secs << " seconds" << endl;
80 cout << "\nTotal length of of data :" << count << endl;
81 getchar();
82 return 0;
83 }

```

Listing 9.2: fft\_test.cpp code

**Step 5 :** Run the *fft\_test.cpp* file.

This will generate a *frequency\_function.txt* file in the same folder which contains the Fourier transformed data.

**Step 6 :** Now, go to the *fft\_test.m* and run section 2 in the code by pressing "ctrl+Enter".

The section 2 reads *frequency\_function.txt* and compares both C++ and MATLAB calculation

of Fourier transformed data.

### Resultant analysis of various test signals

The following section will display the comparative analysis of MATLAB and C++ FFT program to calculate several type of signals.

#### 1. Signal with two sinusoids and random noise

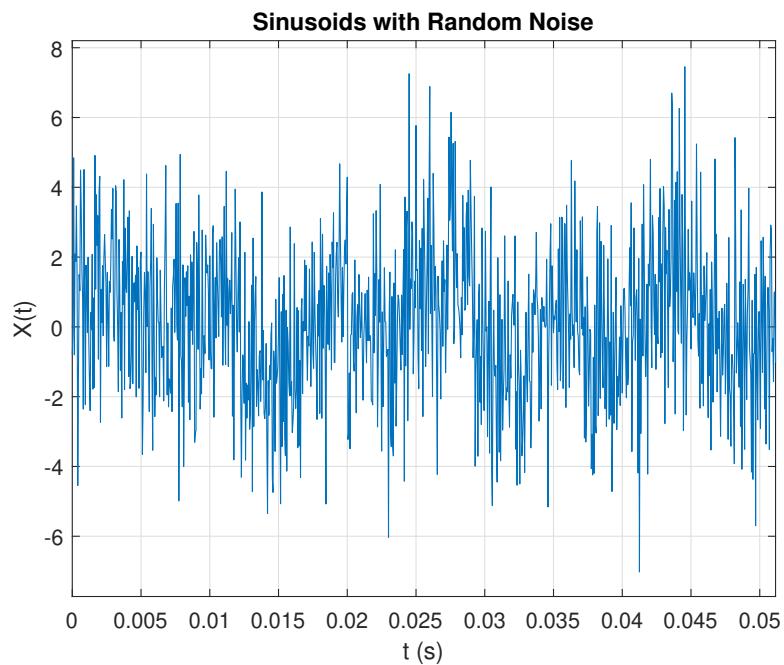


Figure 9.6: Random noise and two sinusoids

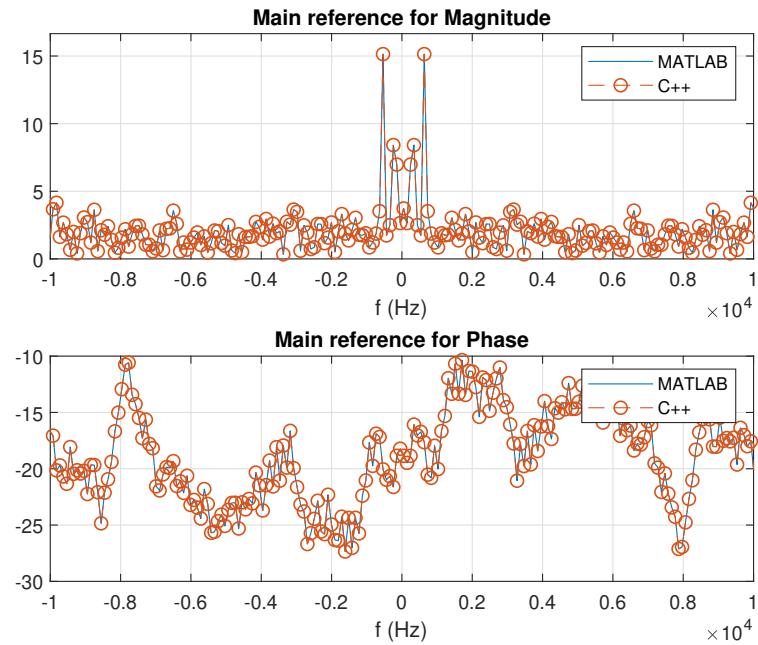


Figure 9.7: MATLAB and C++ comparison

## 2. Sinusoid with an exponent

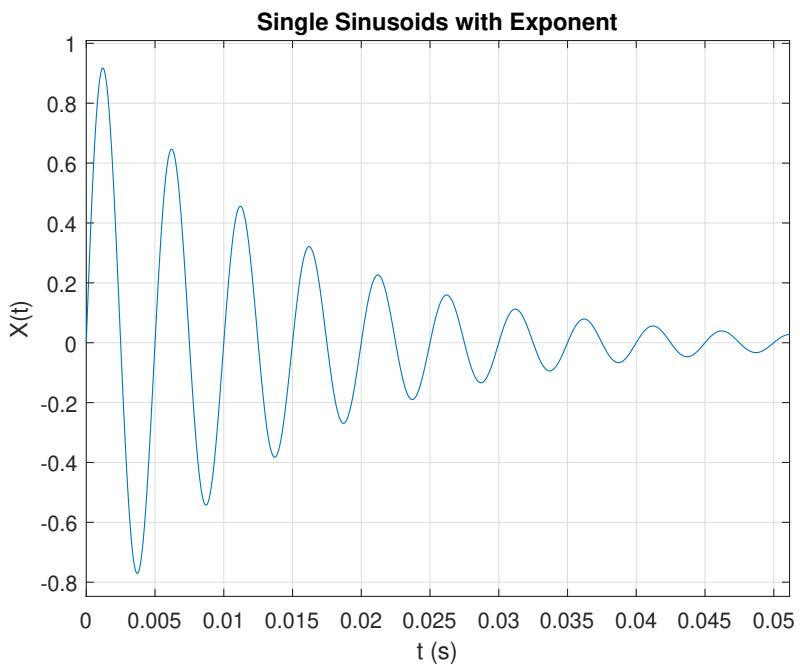


Figure 9.8: Sinusoids with exponent

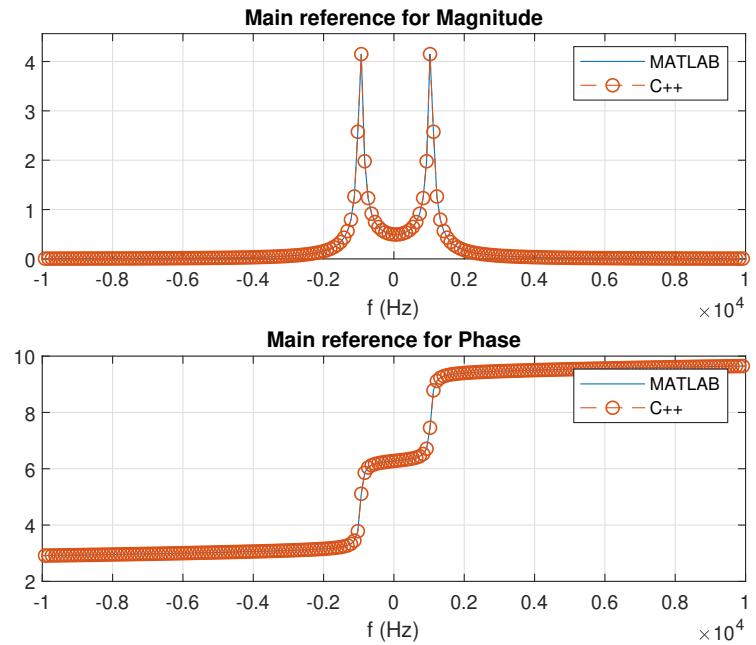


Figure 9.9: MATLAB and C++ comparison

### 3. Mixed signal

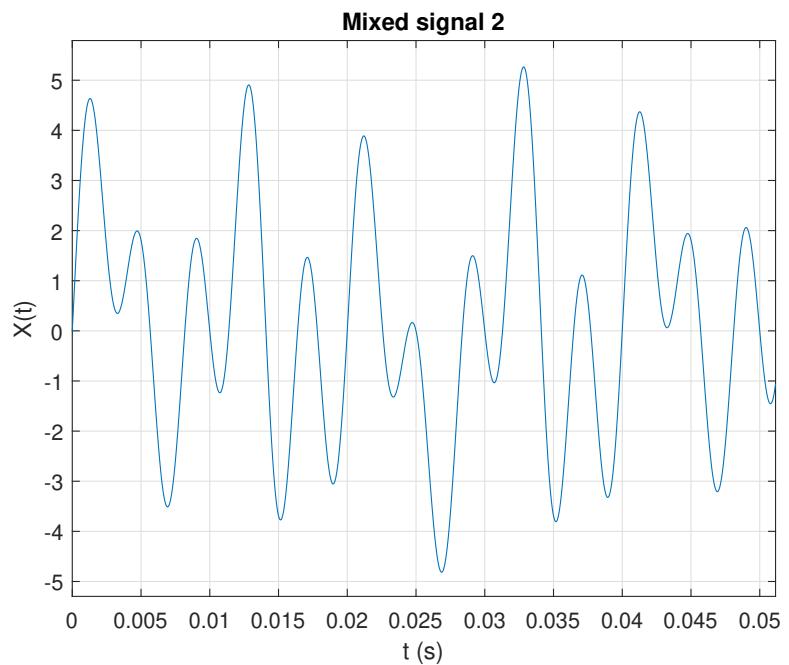


Figure 9.10: mixed signal

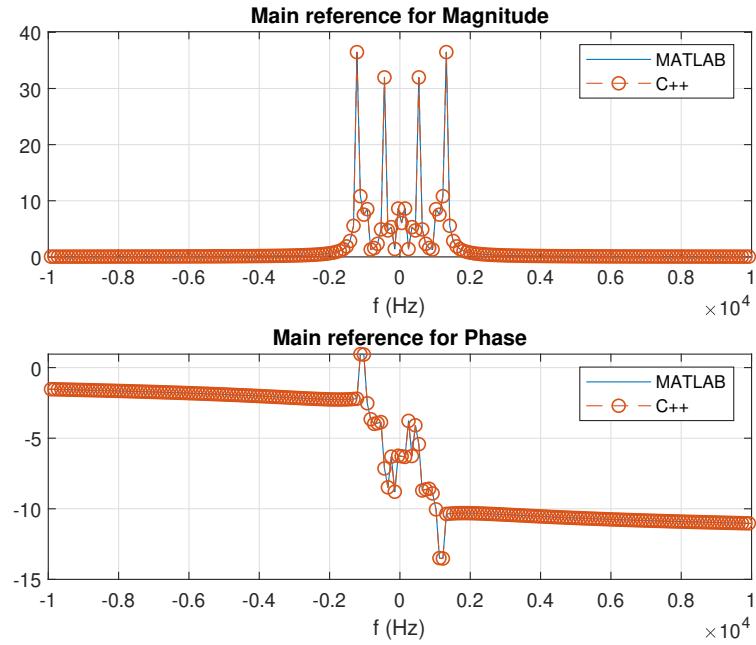


Figure 9.11: MATLAB and C++ comparison

### Remarks

To write the data from the MATLAB in the form of text file, **fprintf** MATLAB function was used with the accuracy of the 15 digits. Similarly; to write the fft calculated data from the C++ in the form of text file, C++ **double** data type with precision of 15 digits applied to the object of **ofstream** class.

## Optimized FFT

### Algorithm

The algorithm for the optimized FFT will be implemented according with the following expression,

$$X_k = \sum_{n=0}^{N-1} x_n e^{m i 2\pi k n / N} \quad 0 \leq k \leq N - 1 \quad (9.4)$$

Similarly, for IFFT,

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{m i 2\pi k n / N} \quad 0 \leq k \leq N - 1 \quad (9.5)$$

where,  $X_k$  is the Fourier transform of  $x_n$ , and  $m$  equals 1 or -1 for FFT and IFFT, respectively.

### Function description

To perform optimized FFT operation, the `fft_*.h` header file must be included and the input argument to the function can be given as follows,

$$y = fft(x, 1, 1)$$

where  $x$  and  $y$  are of the C++ type `vector<complex>`. In a similar way, IFFT can be manipulated as,

$$x = fft(y, -1, 1)$$

If we manipulate the optimized FFT and IFFT functions as  $y = fft(x, 1, 0)$  and  $x = fft(y, -1, 0)$  then it'll calculate the FFT and IFFT as discussed in equation 9.1 and 9.2 respectively.

### Comparative analysis

The following table displays the comparative analysis of time elapsed by FFT and optimized FFT for the various length of the data sequence. This comparison performed on the computer having configuration of 16 GB RAM, i7-3770 CPU @ 3.40GHz with 64-bit Microsoft Windows 10 operating system.

Length of data	Optimized FFT	FFT	MATLAB
$2^{10}$	0.011 s	0.012 s	0.000485 s
$2^{10} + 1$	0.174 s	0.179 s	0.000839 s
$2^{15}$	0.46 s	0.56 s	0.003470 s
$2^{15} + 1$	6.575 s	6.839 s	0.004882 s
$2^{18}$	4.062 s	4.2729 s	0.016629 s
$2^{18} + 1$	60.916 s	63.024 s	0.018992 s
$2^{20}$	18.246 s	19.226 s	0.04217 s
$2^{20} + 1$	267.932 s	275.642 s	0.04217 s

## References

- [1] K. Ramamohan (Kamisetty Ramamohan) Rao, D. N. Kim, and J. J. Hwang. *Fast Fourier transform : algorithms and applications*. Springer, 2010, p. 423. ISBN: 9781402066290.
- [2] Eleanor Chin-hwa Chu and Alan. George. *Inside the FFT black box : serial and parallel fast Fourier transform algorithms*. CRC Press, 2000, p. 312. ISBN: 9781420049961. URL: <https://www.crcpress.com/Inside-the-FFT-Black-Box-Serial-and-Parallel-Fast-Fourier-Transform-Algorithms/Chu-George/p/book/9780849302701>.

## 9.2 Overlap-Save Method

<b>Header File</b>	:	overlap_save_*.h
<b>Source File</b>	:	overlap_save_*.cpp
<b>Version</b>	:	20180201 (Romil Patel)

Overlap-save is an efficient way to evaluate the discrete convolution between a very long signal and a finite impulse response (FIR) filter. The overlap-save procedure cuts the signal into equal length segments with some overlap and then it performs convolution of each segment with the FIR filter. The overlap-save method can be computed in the following steps [1, 2] :

**Step 1 :** Determine the length  $M$  of impulse response,  $h(n)$ .

**Step 2 :** Define the size of FFT and IFFT operation,  $N$ . The value of  $N$  must greater than  $M$  and it should in the form  $N = 2^k$  for the efficient implementation.

**Step 3 :** Determine the length  $L$  to section the input sequence  $x(n)$ , considering that  $N = L + M - 1$ .

**Step 4 :** Pad  $L - 1$  zeros at the end of the impulse response  $h(n)$  to obtain the length  $N$ .

**Step 5 :** Make the segments of the input sequences of length  $L$ ,  $x_i(n)$ , where index  $i$  correspond to the  $i^{th}$  block. Overlap  $M - 1$  samples of the previous block at the beginning of the segmented block to obtain a block of length  $N$ . In the first block, it is added  $M - 1$  null samples.

**Step 6 :** Compute the circular convolution of segmented input sequence  $x_i(n)$  and  $h(n)$  described as,

$$y_i(n) = x_i(n) \circledast h(n). \quad (9.6)$$

This is obtained in the following steps:

1. Compute the FFT of  $x_i$  and  $h$  both with length  $N$ .
2. Compute the multiplication of  $X_i(f)$  and the transfer function  $H(f)$ .
3. Compute the IFFT of the multiplication result to obtain the time-domain block signal,  $y_i$ .

**Step 7 :** Discarded  $M - 1$  initial samples from the  $y_i$ , and save only the error-free  $N - M - 1$  samples in the output record.

In the Figure 9.12 it is illustrated an example of overlap-save method.

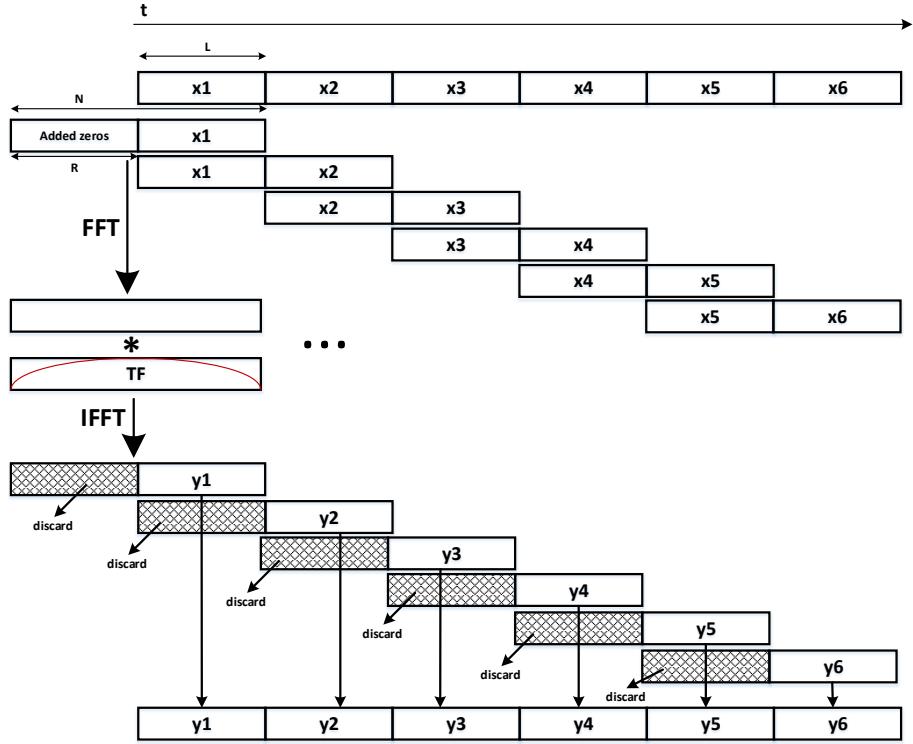


Figure 9.12: Illustration of Overlap-save method.

### Function description

Traditionally, overlap-save method performs the convolution (More precisely, circular convolution) between discrete time-domain signal  $x(n)$  and the filter impulse response  $h(n)$ . Here the length of the signal  $x(n)$  is greater than the length of the filter  $h(n)$ . To perform convolution between the time domain signal  $x(n)$  with the filter  $h(n)$ , include the header file `overlap_save_*.h` and then supply input argument to the function as follows,

$$y(n) = \text{overlapSave}(x(n), h(n))$$

Where,  $x(n)$ ,  $h(n)$  and  $y(n)$  are of the C++ type `vector<complex<double>>` and the length of the signal  $x(n)$  and filter  $h(n)$  could be arbitrary.

The one noticeable thing in the traditional way of implementation of overlap-save is that it cannot work with the real-time system. Therefore, to make it usable in the real-time environment, one more `overlapSave` function with three input parameters was implemented and used along with the traditional overlap-save method. The structure of the new function is as follows,

$$y(n) = \text{overlapSave}(x_m(n), x_{m-1}(n), h(n))$$

Here,  $x_m(n)$ ,  $x_{m-1}(n)$  and  $h(n)$  are of the C++ type `vector<complex<double>>` and the length of each of them are arbitrary. However, the combined length of  $x_{m-1}(n)$  and  $x_m(n)$  must be greater than the length of  $h(n)$ .

### Linear and circular convolution

In the circular convolution, if we determine the length of the signal  $x(n)$  is  $N_1 = 8$  and length of the filter is  $h(n)$  is  $N_2 = 5$ ; then the length of the output signal is determined by  $N = \max(N_1, N_2) = 8$ . Next, the circular convolution can be performed after padding 0 in the filter  $h(n)$  to make it's length equals  $N$ .

In the linear convolution, if we determine the length of the signal  $x(n)$  is  $N_1 = 8$  and length of the filter is  $h(n)$  is  $N_2 = 5$ ; then the length of the output signal is determined by  $N = N_1 + N_2 - 1 = 12$ . Next, the linear convolution using circular convolution can be performed after padding 0 in the signal  $x(n)$  filter  $h(n)$  to make it's length equals  $N$ .

### Flowchart of real-time overlap-save method

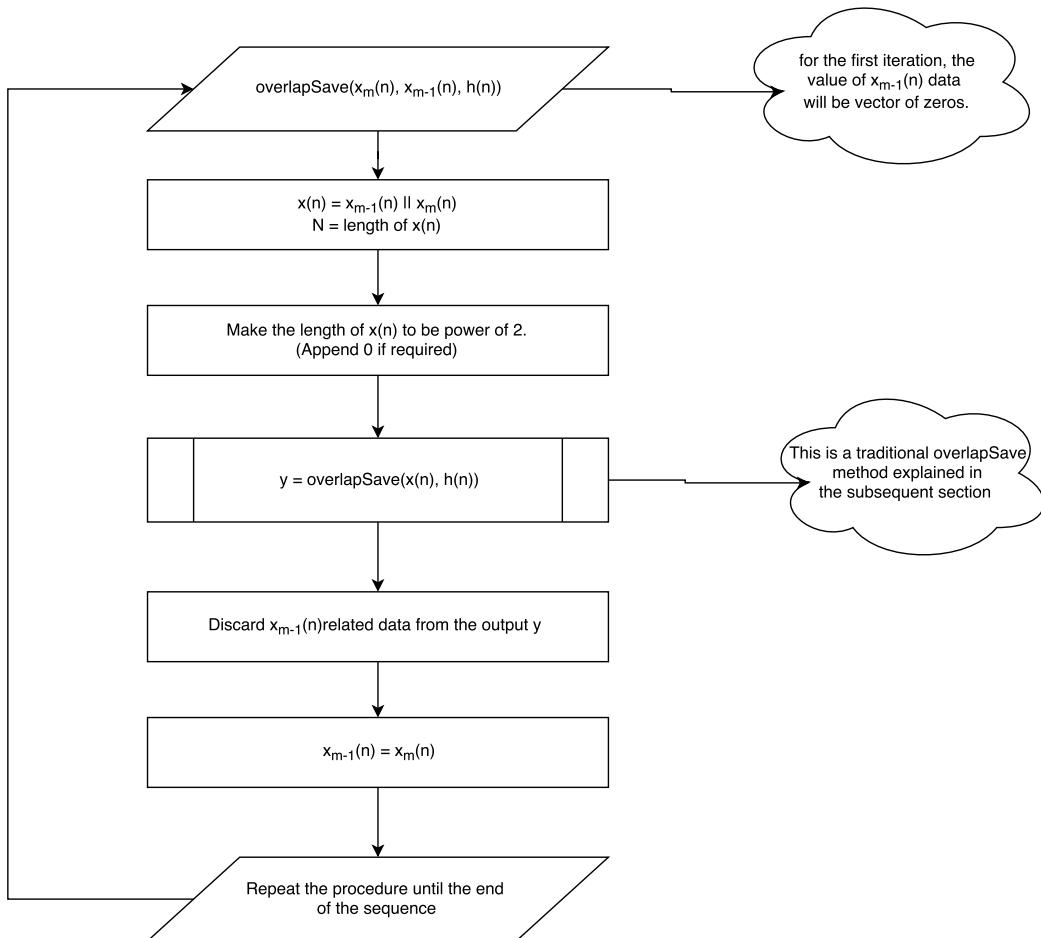


Figure 9.13: Flowchart for real-time overlap-save method

### Flowchart of traditional overlap-save method

The following three flowcharts describe the logical flow of the traditional overlap-save method with two inputs as  $overlapSave(x(n), h(n))$ . In the flowchart,  $x(n)$  and  $h(n)$  are regarded as  $inTimeDomainComplex$  and  $inTimeDomainFilterComplex$  respectively.

#### 1. Decide length of FFT, data block and filter

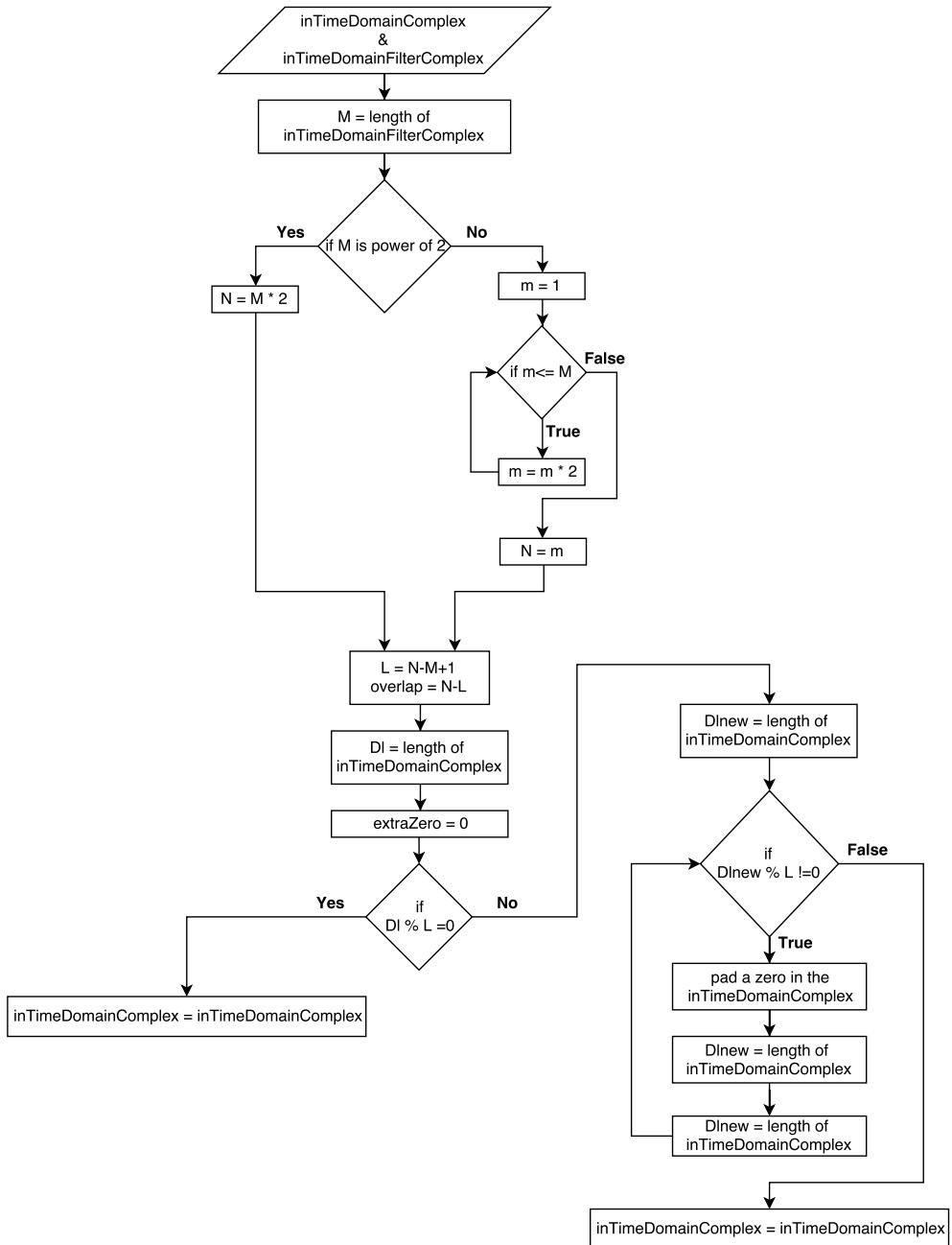


Figure 9.14: Flowchart for calculating length of FFT, data block and filter

## 2. Create matrix with overlap

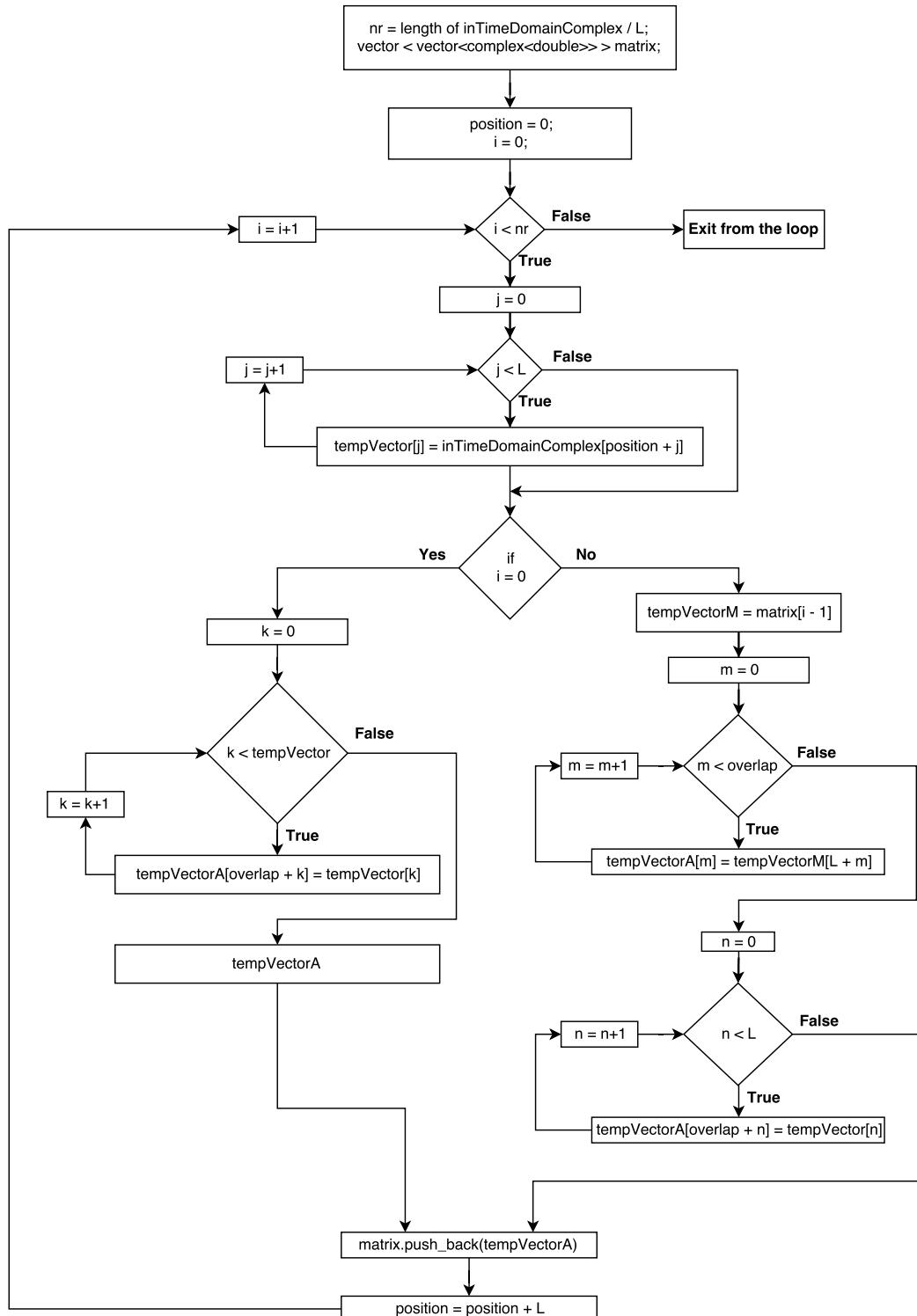


Figure 9.15: Flowchart of creating matrix with overlap

### 3. Convolution between filter and data blocks

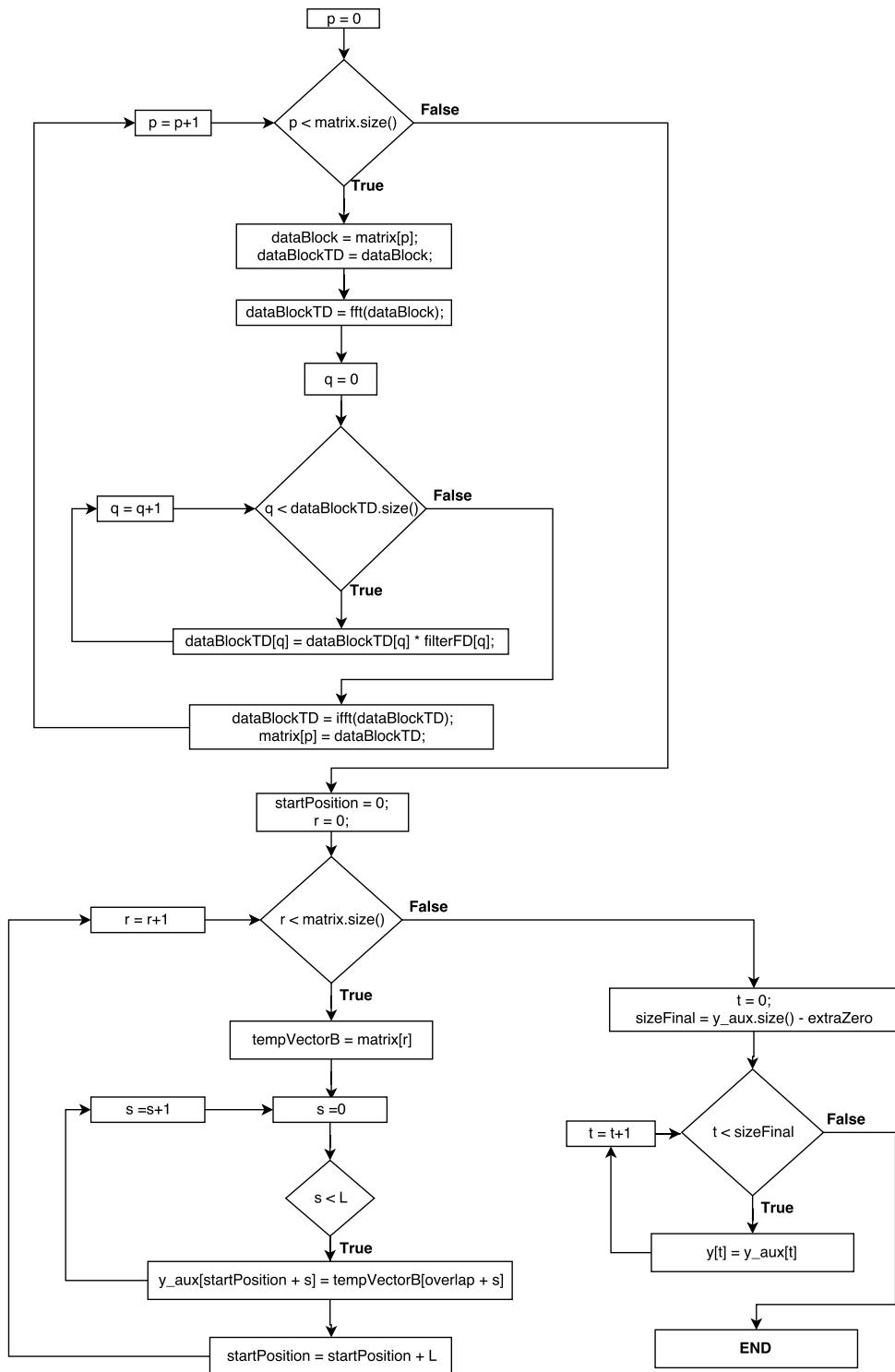


Figure 9.16: Flowchart of the convolution

### Test example of traditional overlap-save function

This sections explains the steps of comparing our C++ based  $overlapSave(x(n), h(n))$  function with the MATLAB overlap-save program and MATLAB's built-in `conv()` function.

**Step 1 :** Open the folder namely `overlapSave_test` by following the path "/algorithms/overlapSave/overlapSave\_test".

**Step 2 :** Find the `overlapSave_test.m` file and open it.

This `overlapSave_test.m` consists of five sections:

**section 1 :** It generates the time domain signal and filter impulse response and save them in the form of the text file with the name of `time_domain_data.txt` and `time_domain_filter.txt` respectively in the same folder.

**Section 2 :** It calculates the length of FFT, data blocks and filter to perform convolution using overlap-save method.

**Section 3 :** It consists of overlap-save code which first converts the data into the form of matrix with 50% overlap and then performs circular convolution with filter.

**Section 4 :** It read `overlap_save_data.txt` data file generated by C++ program and compare with MATLAB implementation.

**Section 5 :** It compares our MATLAB and C++ implementation with the built-in MATLAB function `conv()`.

```

%
%%%%%%%%%%%%%%%
2 %%%%%%%%%%%%%%% SECTION 1
%%%%%%%%%%%%%%%%
%
%
4 % generate signal and filter data and save it as a .txt file .
clc
6 clear all
close all
8
10 Fs = 1e5; % Sampling frequency
12 T = 1/Fs; % Sampling period
14 L = 2^10; % Length of signal
16 t = (0:L-1)*(5*T); % Time vector
18 f = linspace(-Fs/2,Fs/2,L);
20
%Choose for sig a value between [1, 7]
sig = 7;
switch sig
    case 1
        signal_title = 'Signal with one signusoid and random noise';
        S = 0.7*sin(2*pi*50*t);

```

```

22 X = S + 2*randn(size(t));
23 case 2
24     signal_title = 'Sinusoids with Random Noise';
25     S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
26     X = S + 2*randn(size(t));
27 case 3
28     signal_title = 'Single sinusoids';
29     X = sin(2*pi*t);
30 case 4
31     signal_title = 'Summation of two sinusoids';
32     X = sin(2*pi*1205*t) + cos(2*pi*1750*t);
33 case 5
34     signal_title = 'Single Sinusoids with Exponent';
35     X = sin(2*pi*250*t).*exp(-70*abs(t));
36 case 6
37     signal_title = 'Mixed signal 1';
38     X = sin(2*pi*10*t).*exp(-t)+sin(2*pi*t)+7*sin(2*pi*+5*t)+7*cos(2*pi*+20*t)+5*sin(2*pi*+50*t);
39 case 7
40     signal_title = 'Mixed signal 2';
41     X = 2*sin(2*pi*100*t).*exp(-t)+2.5*sin(2*pi*+250*t)+sin(2*pi*+50*t).*cos(2*pi*+20*t)+1.5*sin(2*pi*+50*t).*sin(2*pi*+150*t);
42 end
43
44 Xref = X;
45 % dlmwrite will generate text file which represents the time domain signal.
46 %dlmwrite('time_domain_data.txt', X, 'delimiter', '\t');
47 fid=fopen('time_domain_data.txt','w');
48 fprintf(fid, '%.20f\n', X); % 12-Digit accuracy
49 fclose(fid);
50
51 % Choose for filt a value between [1, 3]
52 filt = 1;
53 switch filt
54     case 1
55         filter_type = 'Impulse response of rcos filter';
56         h = rcosdesign(0.25,11,6);
57     case 2
58         filter_type = 'Impulse response of rrcos filter';
59         h = rcosdesign(0.25,11,6, 'sqrt');
60     case 3
61         filter_type = 'Impulse response of Gaussian filter';
62         h = gaussdesign(0.25,11,6);
63 end
64
65 %dlmwrite('time_domain_filter.txt', h, 'delimiter', '\t');
66 fid=fopen('time_domain_filter.txt','w');
67 fprintf(fid, '%.20f\n', h); % 20-Digit accuracy
68 fclose(fid);
69
70 figure;

```

```

72 subplot(211)
73 plot(t,X)
74 grid on
75 title ( signal_title )
76 axis([ min(t) max(t) 1.1*min(X) 1.1*max(X)]);
77 xlabel('t (s)')
78 ylabel('X(t)')

80 subplot(212)
81 plot(h)
82 grid on
83 title ( filter_type )
84 axis([1 length(h) 1.1*min(h) 1.1*max(h)]);
85 xlabel('Samples')
86 ylabel('h(t)')

88 %%
89 %

%%%%%%%%%%%%%% SECTION 2
90 %

92 % Calculate the length of FFT, data blocks and filter
M = length(h);

94
if (bitand(M,M-1)==0)
    N = 2 * M; % Where N is the size of the FFT
else
    m =1;
    while(m<=M) % Next value of the order of power 2.
        m = m*2;
    end
    N = m;
end

104
L = N -M+1; % Size of data block (50% of overlap)
106 overlap = N - L; % size of overlap
Dl = length(X);
108 extraZeros = 0;
109 if (mod(Dl,L) == 0)
    X = X;
else
    Dlnew = length(X);
    while (mod(Dlnew,L) ~= 0)
        X = [X 0];
        Dlnew = length(X);
        extraZeros = extraZeros + 1;
    end
end
118

```

```

120 %%%
121 %
122 %%%%%% SECTION 3
123 %
124 % MATLAB approach of overlap-save method (First create matrix with
125 % overlap and then perform convolution)
126 zerosForFilter = zeros(1,N-M);
127 h1=[h zerosForFilter];
128 H1 = fft(h1);
129
130 x1=X;
131 nr=ceil((length(x1))/L);
132
133 tic
134 for k=1:nr
135     Ma(k,:)=x1(((k-1)*L+1):k*L);
136     if k==1
137         Ma1(k,:)=[zeros(1,overlap) Ma(k,:)];
138     else
139         tempVectorM = Ma1(k-1,:);
140         overlapData = tempVectorM(L+1:end);
141         Ma1(k,:)=[overlapData Ma(k,:)];
142     end
143     auxfft = fft(Ma1(k,:));
144     auxMult = auxfft.*H1;
145     Ma2(k,:)=ifft(auxMult);
146 end
147
148 Ma3=Ma2(:,N-L+1:end);
149 y1=Ma3';
150 y=y1(:)';
151 y = y(1:end - extraZeros);
152 toc
153 %%
154 %%%%%% SECTION 4
155 %
156 % Read overlap-save data file generated by C++ program and compare with
157 fullData = load('overlap_save_data.txt');
158 A=1;
159 B=A+1;
160 l=1;

```

```

162 Z=zeros(length(fullData)/2,1);
163 while (l<=length(Z))
164 Z(l) = fullData(A)+fullData(B)*l;
165 A = A+2;
166 B = B+2;
167 l=l+1;
168 end
169
170 figure;
171 plot(t,real(y))
172 hold on
173 plot(t,real(Z),'o')
174 axis([min(t) max(t) 1.1*min(y) 1.1*max(y)]);
175 xlabel('t (Seconds)')
176 ylabel('y(t)')
177 title ('Comparision of overlapSave method of MATLAB and C++ ')
178 legend('MATLAB overlapSave','C++ overlapSave')
179 grid on
180 %%
181 %
182 %%%%%%% SECTION 5
183 %
184 %
185 % Our MATLAB and C++ implementation test with the built-in conv function of
186 % MATLAB.
187 tic
188 P = conv(Xref,h);
189 toc
190 figure
191 plot(t, P(1:size(Z,1)), 'r')
192 hold on
193 plot(t,real(Z),'o')
194 title ('Comparision of MATLAB function conv() and overlaSave')
195 axis([min(t) max(t) 1.1*min(real(Z)) 1.1*max(real(Z))]);
196 xlabel('t (Seconds)')
197 ylabel('y(t)')
198 legend('MATLAB function : conv(X,h)', 'C++ overlapSave')
199 grid on

```

Listing 9.3: overlapSave\_test.m code

**Step 3 :** Choose for a sig and filt value between [1 7] and [1 3] respectively and run the first three sections namely **section 1**, **section 2** and **section 3**.

This will generate a `time_domain_data.txt` and `time_domain_filter.txt` file in the same folder which contains the time domain signal and filter data respectively.

**Step 4 :** Find the `overlapSave_test.vcxproj` file in the same folder and open it.

In this project file, find *overlapSave\_test.cpp* in *SourceFiles* section and click on it. This file is an example of using *overlapSave* function. Basically, *overlapSave\_test.cpp* file consists of four sections:

**Section 1 :** It reads the *time\_domain\_data.txt* and *time\_domain\_filter.txt* files.

**Section 2 :** It converts signal and filter data into complex form.

**Section 3 :** It calls the *overlapSave* function to perform convolution.

**Section 4 :** It saves the result in the text file namely *overlap\_save\_data.txt*.

```

1 # include "overlap_save_20180208.h"

3 # include <complex>
# include <fstream>
5 # include <iostream>
# include <math.h>
7 # include <stdio.h>
# include <string>
9 # include <sstream>
# include <algorithm>
11 # include <vector>

13 using namespace std;

15 int main()
{
    ////////////////////////////////////////////////////////////////// Section 1 //////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////// Read the time_domain_data.txt and time_domain_filter.txt files //////////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////////////////////
19 ifstream inFile;
21 double ch;
22 vector <double> inTimeDomain;
23 inFile.open("time_domain_data.txt");

25 // First data (at 0th position) applied to the ch it is similar to the "cin".
26 inFile >> ch;

27 // It'll count the length of the vector to verify with the MATLAB
28 int count = 0;

30 while (!inFile.eof())
{
    // push data one by one into the vector
32 inTimeDomain.push_back(ch);

34 // it'll increase the position of the data vector by 1 and read full vector.s
35 inFile >> ch;
36 count++;
37 }

39 inFile.close(); // It is mandatory to close the file at the end.

```

```

43 ifstream inFileFilter ;
44 double chFilter;
45 vector <double> inTimeDomainFilter;
46 inFileFilter .open("time_domain_filter.txt");
47 inFileFilter >> chFilter;
48 int countFilter = 0;
49
50 while (! inFileFilter .eof())
51 {
52     inTimeDomainFilter.push_back(chFilter);
53     inFileFilter >> chFilter;
54     countFilter++;
55 }
56 inFileFilter .close();
57
58 ////////////////////////////////////////////////////////////////// Section 2 //////////////////////////////////////////////////////////////////
59 ////////////////////////////////////////////////////////////////// Real to complex conversion //////////////////////////////////////////////////////////////////
60 //////////////////////////////////////////////////////////////////
61 ////////////////////////////////////////////////////////////////// For signal data //////////////////////////////////////////////////////////////////
62 vector <complex<double>> inTimeDomainComplex(inTimeDomain.size());
63 vector <complex<double>> fourierTransformed;
64 vector <double> re(inTimeDomain.size());
65 vector <double> im(inTimeDomain.size());
66
67 for (unsigned int i = 0; i < inTimeDomain.size(); i++)
68 {
69     // Real data of the signal
70     re[i] = inTimeDomain[i];
71
72     // Imaginary data of the signal
73     im[i] = 0;
74 }
75 // Next, Real and Imaginary vector to complex vector conversion
76 inTimeDomainComplex = reImVect2ComplexVector(re, im);
77
78 ////////////////////////////////////////////////////////////////// For filter data //////////////////////////////////////////////////////////////////
79 vector <complex<double>> inTimeDomainFilterComplex(inTimeDomainFilter.size());
80 vector <double> reFilter(inTimeDomainFilter.size());
81 vector <double> imFilter(inTimeDomainFilter.size());
82
83 for (unsigned int i = 0; i < inTimeDomainFilter.size(); i++)
84 {
85     reFilter [i] = inTimeDomainFilter[i];
86     imFilter[i] = 0;
87 }
88 inTimeDomainFilterComplex = reImVect2ComplexVector(reFilter, imFilter);
89
90 ////////////////////////////////////////////////////////////////// Section 3 //////////////////////////////////////////////////////////////////
91 ////////////////////////////////////////////////////////////////// Overlap & save //////////////////////////////////////////////////////////////////
92 //////////////////////////////////////////////////////////////////
93 vector <complex<double>> y;
94 y = overlapSave(inTimeDomainComplex, inTimeDomainFilterComplex);

```

```

95 ////////////////////////////////////////////////////////////////// Section 4 //////////////////////////////////////////////////////////////////
97 ////////////////////////////////////////////////////////////////// Save data //////////////////////////////////////////////////////////////////
99 ofstream outFile;
101 complex<double> outFileData;
102 outFile.precision(20);
103 outFile.open("overlap_save_data.txt");
104
105 for (unsigned int i = 0; i <y.size(); i++)
106 {
107     outFile << y[i].real() << endl;
108     outFile << y[i].imag() << endl;
109 }
110 outFile.close();
111 cout << "Execution finished! Please hit enter to exit." << endl;
112 getchar();
113 return 0;
}

```

Listing 9.4: overlapSave\_test.cpp code

**Step 5 :** Now, go to the **overlapSave\_test.m** and run section 4 and 5.

It'll display the graphs of comparative analysis of the MATLAB and C++ implementation of overlapSave program and also compares results with the MATLAB conv() function.

### Resultant analysis of various test signals

1. Signal with two sinusoids and random noise
2. Mixed signal2

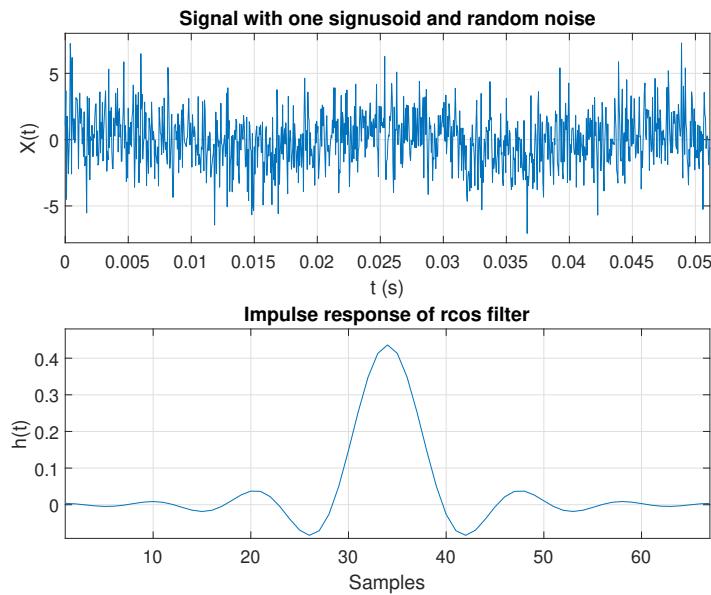


Figure 9.17: Random noise and two sinusoids signal & Impulse response of rcos filter

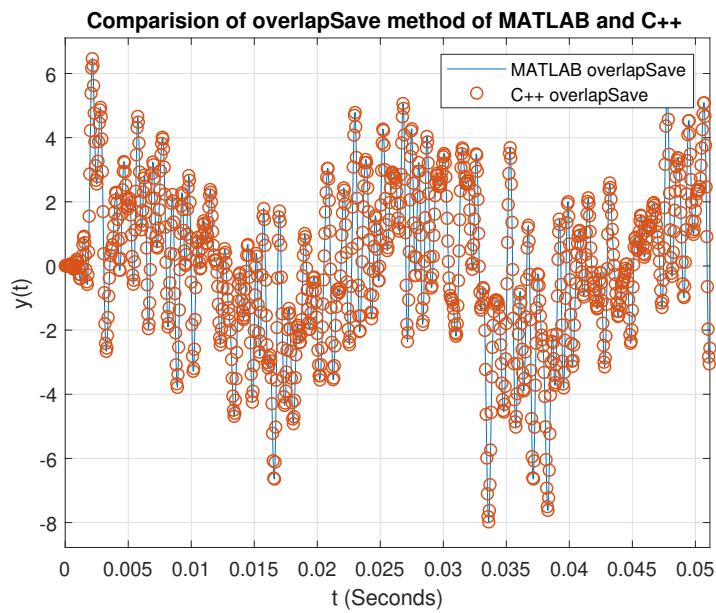


Figure 9.18: MATLAB and C++ comparison

### 3. Sinusoid with exponent

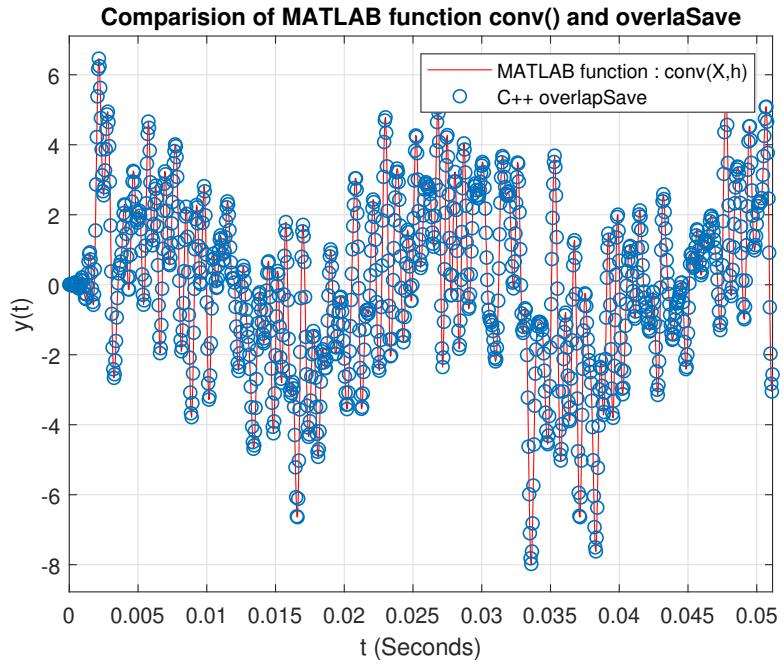


Figure 9.19: MATLAB function `conv()` and C++ `overlapSave` comparison

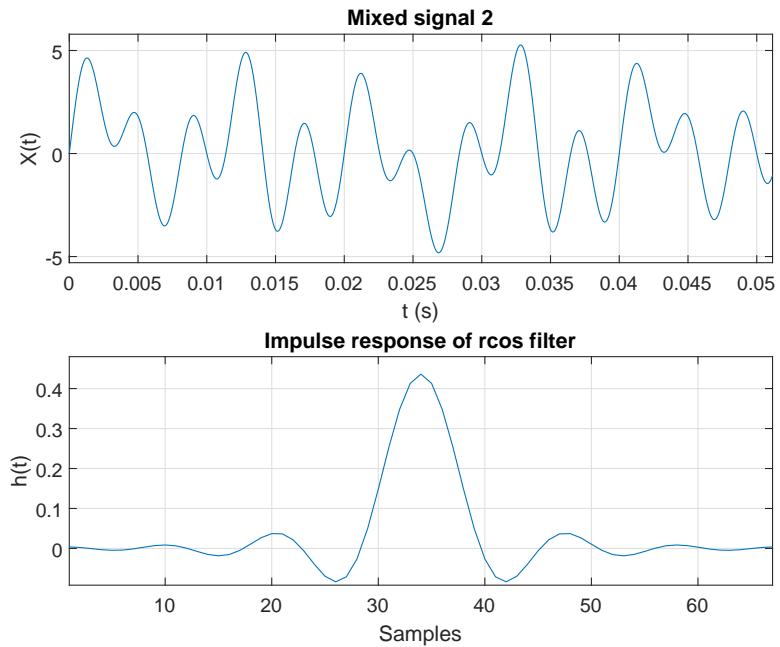


Figure 9.20: Mixed signal2 & Impulse response of rcos filter

### Test example of real-time overlap-save function with Netxpto simulator

This section explains the steps of comparing real-time overlap-save method with the time-domain filtering. The structure of the real-time overlap-save function

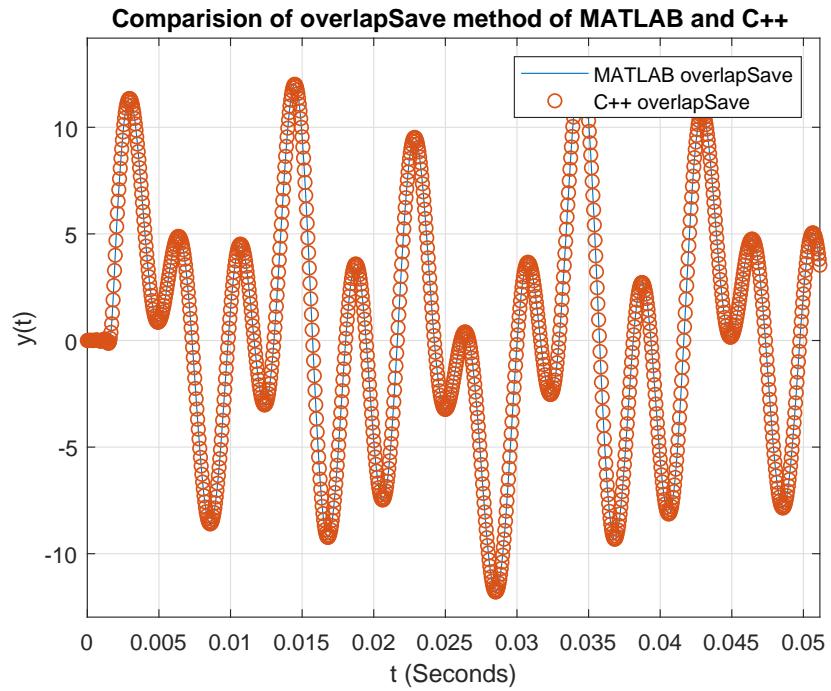


Figure 9.21: MATLAB and C++ comparison

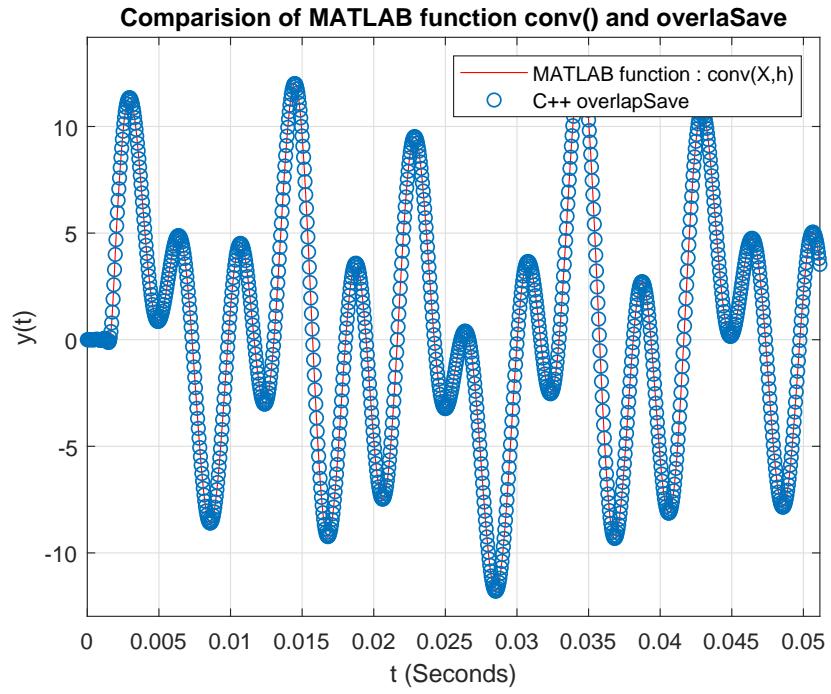


Figure 9.22: MATLAB function conv() and C++ overlapSave comparison

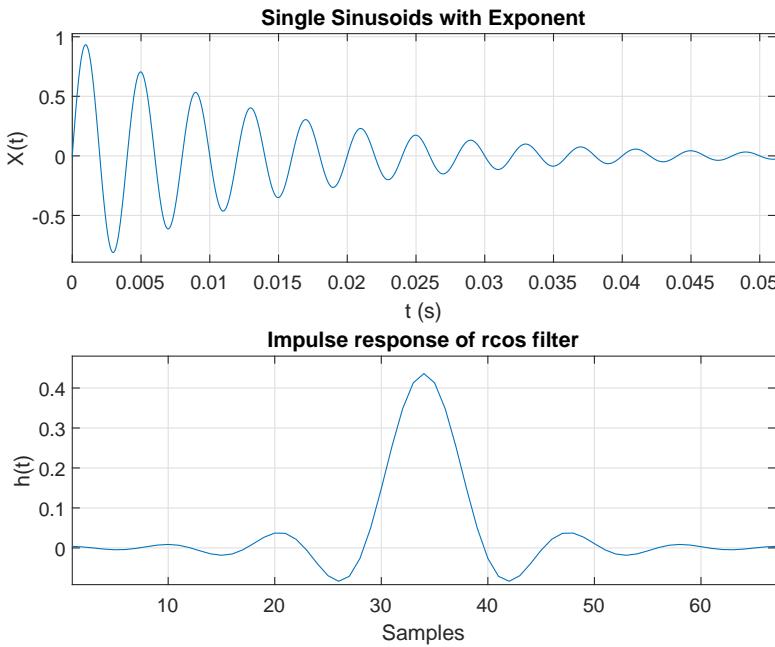


Figure 9.23: Sinusoid with exponent & Impulse response of Gaussian filter

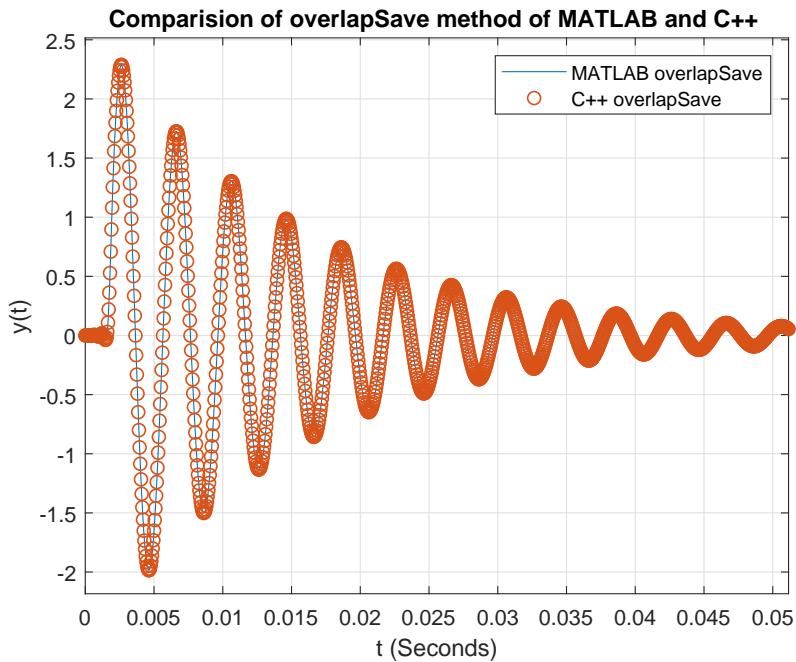


Figure 9.24: MATLAB and C++ comparison

$overlapSave(x_{m-1}(n), x_m(n), h(n))$  requires an impulse response  $h(n)$  of the filter. There are two methods to feed the impulse response to the real-time overlap-save function:

**Method 1.** The impulse response  $h(n)$  of the filter can be fed using the time-domain impulse

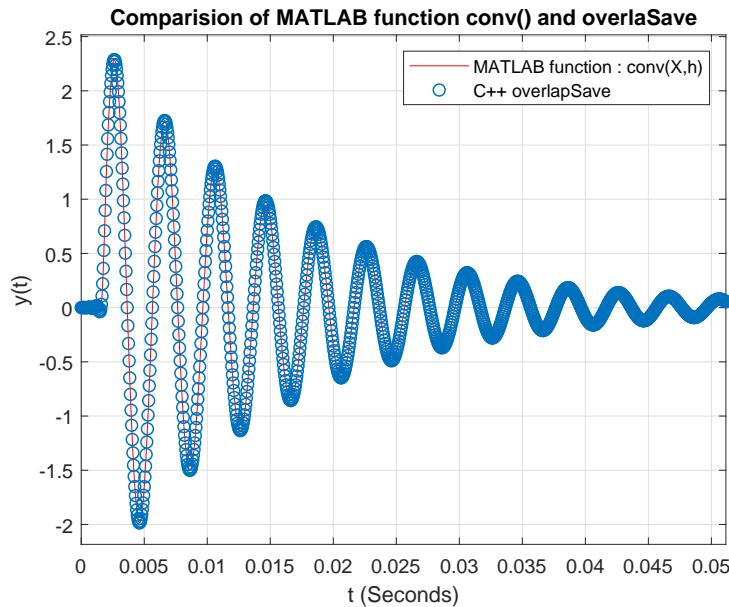


Figure 9.25: MATLAB function `conv()` and C++ `overlapSave` comparison

response formula of the filter.

**Method 2.** Write the transfer function of the filter and convert it into the impulse response using Fourier transform method.

Here, this example uses the method 2 to feed the impulse response of the filter. In order to compare the result, follow the steps given below:

**Step 1 :** Open the folder namely `overlapSaveRealTime_test` by following the path `"/algorithms/overlapSave/overlapSaveRealTime_test"`.

**Step 2 :** Find the `overlapSaveRealTime_test.vcxproj` file and open it.

In this project file, find `filter_20180306.cpp` in *SourceFiles* section and click on it. This file includes the several definitions of the two different filter class namely **FIR\_Filter** and **FD\_Filter** for filtering in time-domain and frequency-domain respectively. In this file, **FD\_Filter::runBlock** displays the logic of real-time overlap-save method.

```

1 # include "filter_20180306.h"
2
3 //////////////////////////////////////////////////// FIR_Filter //////////////////////////////////// TIME DOMAIN
4
5
6 void FIR_Filter :: initializeFIR_Filter (void) {
7
8     outputSignals[0]->symbolPeriod = inputSignals[0]->symbolPeriod;
9     outputSignals[0]->samplingPeriod = inputSignals[0]->samplingPeriod;
10

```



```

62 //////////////////////////////////////////////////////////////////
63 void FD_Filter:: initializeFD_Filter (void)
64 {
65     outputSignals[0] ->symbolPeriod = inputSignals[0] ->symbolPeriod;
66     outputSignals[0] ->samplingPeriod = inputSignals[0] ->samplingPeriod;
67     outputSignals[0] ->samplesPerSymbol = inputSignals[0] ->samplesPerSymbol;
68
69     if (!getSeeBeginningOfTransferFunction()) {
70         int aux = (int) ((double)transferFunctionLength) / 2 + 1;
71         outputSignals[0] ->setFirstValueToBeSaved(aux);
72     }
73
74     if (saveTransferFunction)
75     {
76         ofstream fileHandler("./signals/" + transferFunctionFilename, ios::out);
77         fileHandler << "// ### HEADER TERMINATOR ###\n";
78
79         double samplingPeriod = inputSignals[0] ->samplingPeriod;
80         t_real fWindow = 1 / samplingPeriod;
81         t_real df = fWindow / transferFunction.size();
82
83         t_real f;
84         for (int k = 0; k < transferFunction.size(); k++)
85         {
86             f = -fWindow / 2 + k * df;
87             fileHandler << f << " " << transferFunction[k] << "\n";
88         }
89         fileHandler.close();
90     }
91 }
92
93 bool FD_Filter::runBlock(void)
94 {
95     bool alive{ false };
96
97     int ready = inputSignals[0] ->ready();
98     int space = outputSignals[0] ->space();
99     int process = min(ready, space);
100    if (process == 0) return false;
101
102 ////////////////////////////////////////////////////////////////// previousCopy & currentCopy //////////////////////////////////////////////////////////////////
103 //////////////////////////////////////////////////////////////////
104    vector<double> re(process); // Get the Input signal
105    t_real input;
106    for (int i = 0; i < process; i++){
107        inputSignals[0] ->bufferGet(&input);
108        re.at(i) = input;
109    }
110
111    vector<t_real> im(process);
112    vector<t_complex> currentCopyAux = reImVect2ComplexVector(re, im);

```

```

114 vector<t_complex> pcInitialize(process); // For the first data block only
115 if (K == 0){ previousCopy = pcInitialize; }
116
117 // size modification of currentCopyAux to currentCopy.
118 vector<t_complex> currentCopy(previousCopy.size());
119 for (unsigned int i = 0; i < currentCopyAux.size(); i++){
120     currentCopy[i] = currentCopyAux[i];
121 }
122
123 /////////////////////////////////////////////////////////////////// Filter Data "hn" ///////////////////////////////////////////////////////////////////
124 ///////////////////////////////////////////////////////////////////
125 vector<t_complex> impulseResponse;
126 impulseResponse = transferFunctionToImpulseResponse(transferFunction);
127 vector<t_complex> hn = impulseResponse;
128
129 /////////////////////////////////////////////////////////////////// OverlapSave in Realtime ///////////////////////////////////////////////////////////////////
130 ///////////////////////////////////////////////////////////////////
131 vector<t_complex> OUTaux = overlapSave(currentCopy, previousCopy, hn);
132
133 previousCopy = currentCopy;
134 K = K + 1;
135
136 // Remove the size modified data (opposite to "currentCopyAux to currentCopy")
137 vector<t_complex> OUT;
138 for (int i = 0; i < process; i++){
139     OUT.push_back(OUTaux[previousCopy.size() + i]);
140 }
141
142 // Bufferput
143 for (int i = 0; i < process; i++){
144     t_real val;
145     val = OUT[i].real();
146     outputSignals[0]→bufferPut((t_real)(val));
147 }
148
149 return true;
150 }
```

Listing 9.5: filter\_20180306.cpp code

**Step 3 :** Next, open **overlapSaveRealTime\_test.cpp** file in the same project and run it. Graphically, this file represents the following Figure 9.26.

**Step 4 :** Open the MATLAB visualizer and compare the signal **S6.sgn** and **S7.sgn** as shown in Figure 9.27.

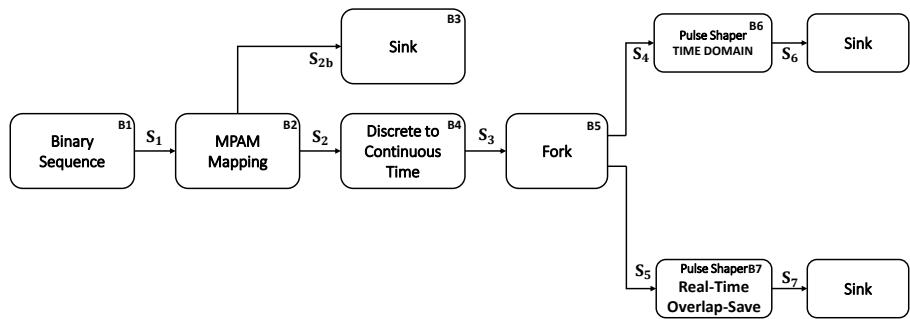


Figure 9.26: Real-time overlap-save example setup

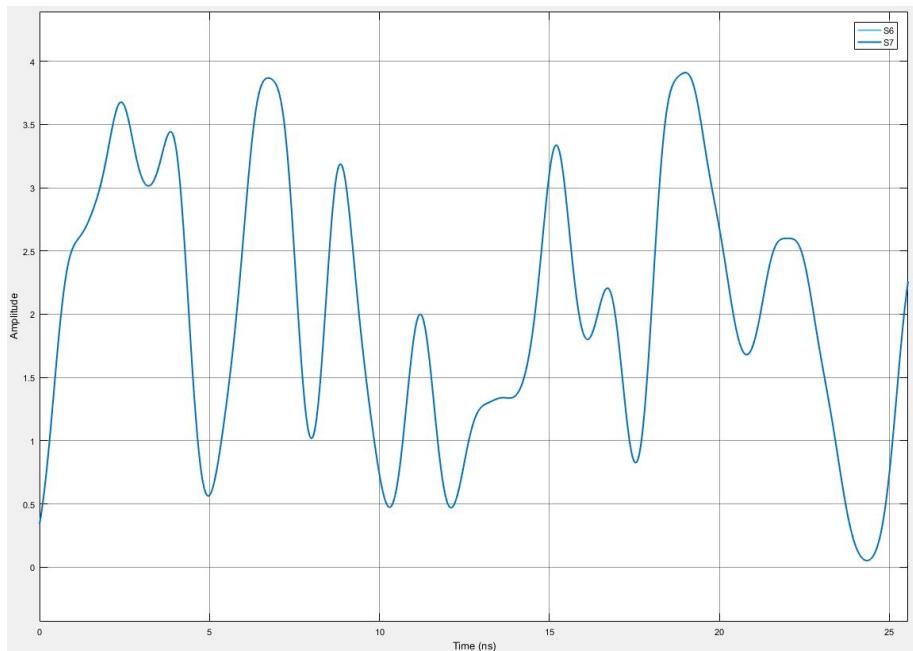


Figure 9.27: Comparison of signal S6 and S7

## References

- [1] Steven W. Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Pub, 1999. ISBN: 0966017676.
- [2] Richard E. Blahut and Richard E. *Fast algorithms for digital signal processing*. Addison-Wesley Pub. Co, 1985, p. 441. ISBN: 0201101556. URL: <https://dl.acm.org/citation.cfm?id=537283>.

### 9.3 Filter

<b>Header File</b>	:	filter_*.h
<b>Source File</b>	:	filter_*.cpp
<b>Version</b>	:	20180201 (Romil Patel)

In order to filter any signal, a new generalized version of the filter namely *filter\_\*.h* & *filter\_\*.cpp* is programmed which facilitate to filtering in both time and frequency domain. Basically, *filter\_\*.h* file contains the declaration two distinct class namely **FIR\_Filter** and **FD\_Filter** which help to perform filtering in time-domain (using impulse response) and frequency-domain (using transfer function), respectively (see Figure 9.28). The *filter\_\*.cpp* file contains the definitions of all the functions declared in the **FIR\_Filter** and **FD\_Filter**.

In the Figure 9.28, the function **bool runblock(void)** in the transfer function based **FD\_Filter**

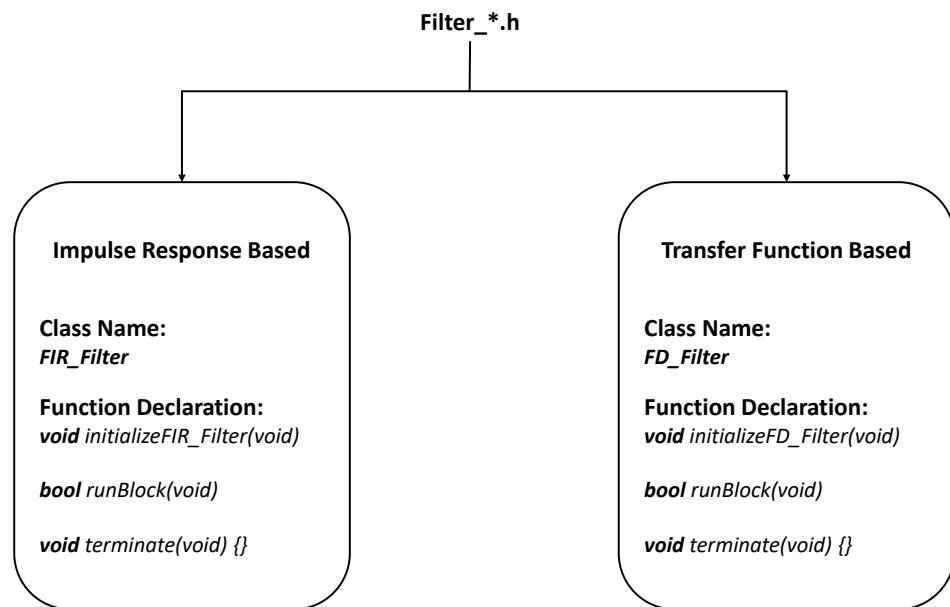


Figure 9.28: Filter class

class, is the declaration of the real-time overlap-save method for filtering in the frequency domain. On the other hand, the function **bool runblock(void)** in the **FIR\_Filter** class is the declaration of the function to facilitate filtering in the time domain [1, 2].

All those function declared in the *filter\_\*.h* file are defined in the *filter\_\*.cpp* file. The definition of **bool runblock(void)** function for both the classes are the following,

2     **bool** FIR\_Filter :: runBlock(**void**) {

4

```

6   int ready = inputSignals[0] -> ready();
7   int space = outputSignals[0] -> space();
8   int process = min(ready, space);
9   if (process == 0) return false;
10
11  for (int i = 0; i < process; i++) {
12      t_real val;
13      (inputSignals[0]) -> bufferGet(&val);
14      if (val != 0) {
15          vector<t_real> aux(impulseResponseLength, 0.0);
16          transform(impulseResponse.begin(), impulseResponse.end(), aux.begin(), bind1st(multiplies<t_real>(), val));
17          transform(aux.begin(), aux.end(), delayLine.begin(), delayLine.begin(), plus<t_real>());
18      }
19      outputSignals[0] -> bufferPut((t_real)(delayLine[0]));
20      rotate(delayLine.begin(), delayLine.begin() + 1, delayLine.end());
21      delayLine[impulseResponseLength - 1] = 0.0;
22  }
23
24  return true;
25 };

```

Listing 9.6: Definition of **bool FIR\_Filter::runBlock(void)**

```

1  bool FD_Filter::runBlock(void)
2  {
3      bool alive{ false };
4
5      int ready = inputSignals[0] -> ready();
6      int space = outputSignals[0] -> space();
7      int process = min(ready, space);
8      if (process == 0) return false;
9
10     /////////////////////////////// previousCopy & currentCopy ///////////////////////////////
11     /////////////////////////////// previousCopy & currentCopy ///////////////////////////////
12     vector<double> re(process); // Get the Input signal
13     t_real input;
14     for (int i = 0; i < process; i++){
15         inputSignals[0] -> bufferGet(&input);
16         re.at(i) = input;
17     }
18
19     vector<t_real> im(process);
20     vector<t_complex> currentCopyAux = reImVect2ComplexVector(re, im);
21
22     vector<t_complex> pcInitialize(process); // For the first data block only
23     if (K == 0){ previousCopy = pcInitialize; }
24
25     // size modification of currentCopyAux to currentCopy.
26     vector<t_complex> currentCopy(previousCopy.size());
27     for (unsigned int i = 0; i < currentCopyAux.size(); i++){
28         currentCopy[i] = currentCopyAux[i];

```

```

30   }
31   /////////////////////////////////////////////////////////////////// Filter Data "hn" ///////////////////////////////////////////////////////////////////
32   /////////////////////////////////////////////////////////////////// vector<t_complex> impulseResponse;
33   impulseResponse = transferFunctionToImpulseResponse(transferFunction);
34   vector<t_complex> hn = impulseResponse;
35
36   /////////////////////////////////////////////////////////////////// OverlapSave in Realtime ///////////////////////////////////////////////////////////////////
37   /////////////////////////////////////////////////////////////////// vector<t_complex> OUTaux = overlapSave(currentCopy, previousCopy, hn);
38
39   previousCopy = currentCopy;
40   K = K + 1;
41
42   // Remove the size modified data (opposite to "currentCopyAux to currentCopy")
43   vector<t_complex> OUT;
44   for (int i = 0; i < process; i++){
45     OUT.push_back(OUTaux[previousCopy.size() + i]);
46   }
47
48   // Bufferput
49   for (int i = 0; i < process; i++){
50     t_real val;
51     val = OUT[i].real();
52     outputSignals[0]→bufferPut((t_real)(val));
53   }
54
55   return true;
56 }
57
58 }
```

Listing 9.7: Definition of **bool FD\_Filter::runBlock(void)**

Both the class of the filter discussed above are the root class for the filtering operation in time and frequency domain. To perform filtering operation, we have to include *filter\_\*.h* and *filter\_\*.cpp* in the project. These filter root files require either *impulse response* or *transfer function* of the filter to perform filtering operation in time domain and frequency domain respectively. In the next section, we'll discuss an example of pulse shaping filtering using the proposed filter root class.

### Example of pulse shaping filtering

This section explains how to use **FIR\_Filter** and **FD\_Filter** class for the pulse shaping using the impulse response and the transfer function, respectively and it also compares the resultant output of both methods. The impulse response for the **FIR\_Filter** class will be generated by a *pulse\_shaper.cpp* file and the transfer function for the **FD\_Filter** will be generated by a *pulse\_shaper\_fd\_20180306.cpp* file and applied to the **bool runblock(void)** block as shown in Figure 9.29.

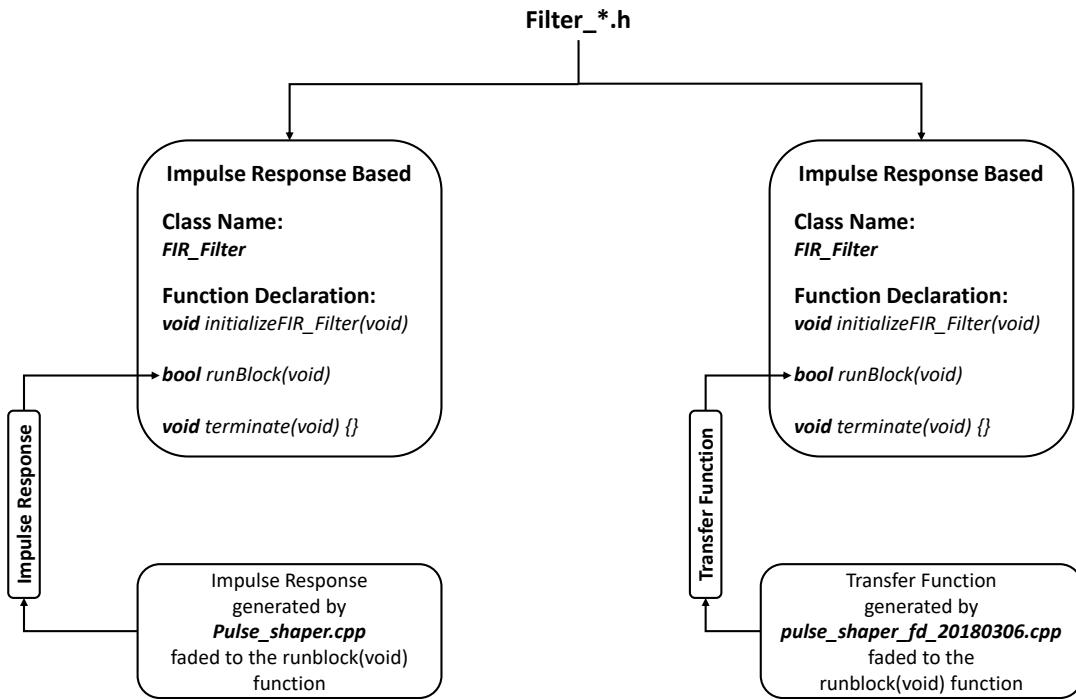


Figure 9.29: Pulse shaping using filter\_\* .h

### Example of pulse shaping filtering : Procedural steps

This section explains the steps of filtering a signal with the various pulse shaping filter using its impulse response and transfer function as well. It also displays the comparison between the resultant output generated by both the methods. In order to conduct the experiment, follow the steps given below:

**Step 1 :** In the directory, open the folder namely **filter\_test** by following the path "/algorithms/filter/filter\_test".

**Step 2 :** Find the **filter\_test.vcxproj** file in the same folder and open it.

In this project file, find **filter\_test.cpp** in *SourceFiles* section and click on it. This file represents the simulation set-up as shown in Figure 9.30.

**Step 3 :** Check how **PulseShaper** and **PulseShaperFd** blocks are implemented.

Check the appendix for the various types of pulse shaping techniques and what are the different parameters used to adjust the shape of the pulse shaper.

**Step 4 :** Run the **filter\_test.cpp** code and compare the signals **S6.sgn** and **S7.sgn** using visualizer.

Here, we have used three different types of pulse shaping filter namely, raised cosine, root raised cosine and Gaussian pulse shaper. The following Figure 9.31, 9.32 and 9.33 display

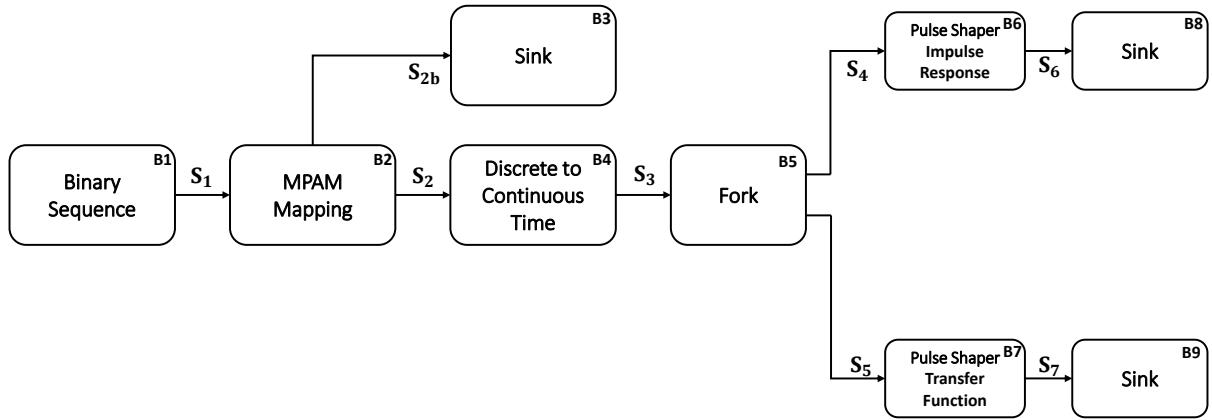


Figure 9.30: Filter test setup

the comparison of the output signals **S6.sgn** and **S7.sgn** for the raised cosine, root raised cosine and Gaussian pulse shaping filter, respectively.

### Case 1 : Raised cosine

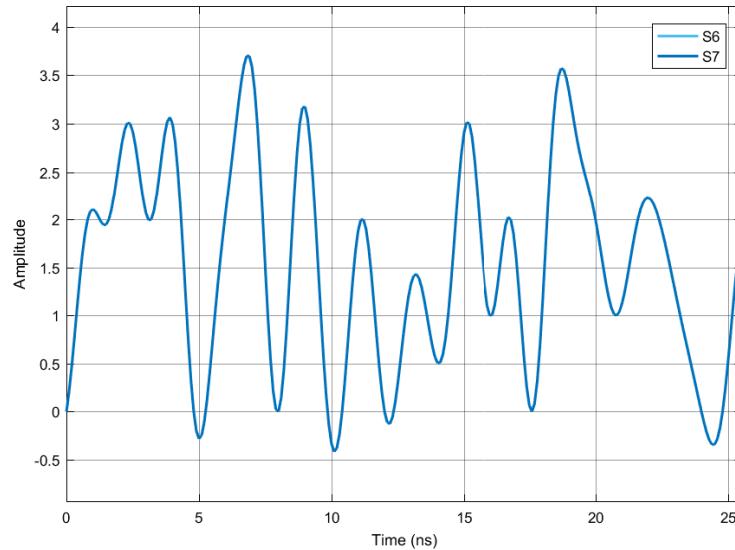


Figure 9.31: Raised cosine pulse shaping results comparison

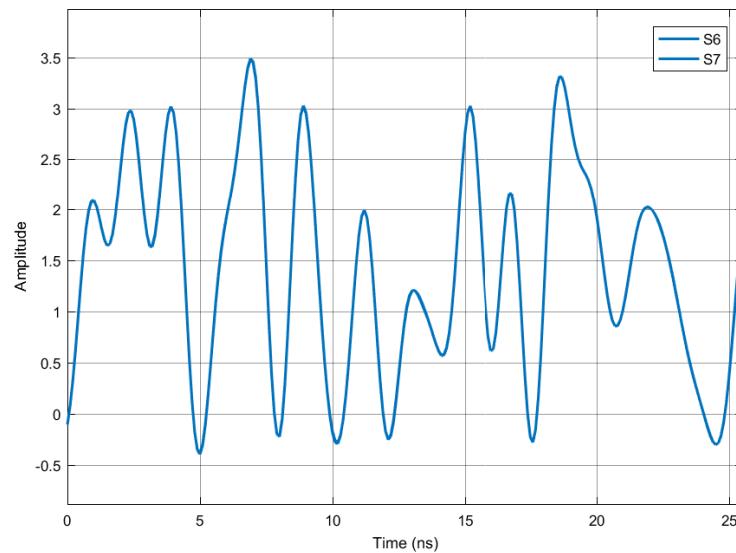
**Case 2 : Root raised cosine**

Figure 9.32: Root raised cosine pulse shaping result

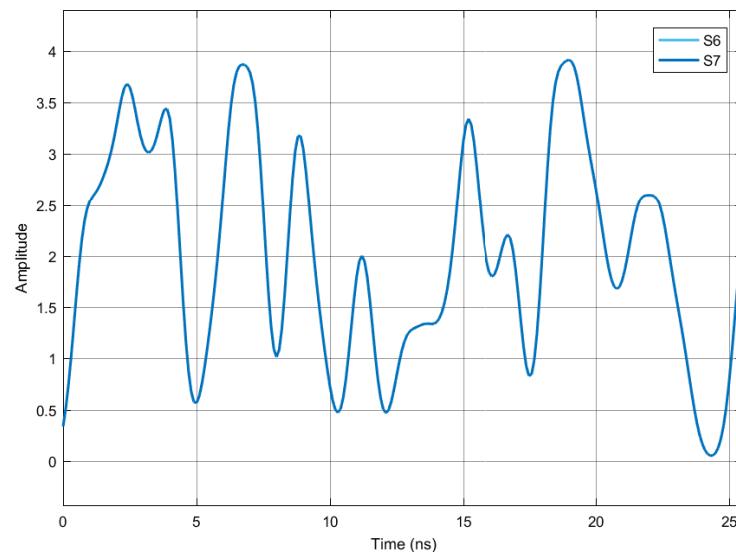
**Case 3 : Gaussian**

Figure 9.33: Gaussian pulse shaping results comparison

## APPENDICES

### A. Raised cosine pulse shaper

The raised cosine pulse shaping filter has a transfer function given by,

$$H_{RC}(f) = \begin{cases} 1 & \text{for } |f| \leq \frac{1-\beta}{2T_s} \\ \frac{1}{2} \left[ 1 + \cos \left( \frac{\pi T_s}{\beta} \left[ |f| - \frac{1-\beta}{2T_s} \right] \right) \right] & \text{for } \frac{1-\beta}{2T_s} < |f| \leq \frac{1+\beta}{2T_s} \\ 0 & \text{otherwise} \end{cases} \quad (9.7)$$

The parameter,  $\beta$  is the roll-off factor of the raised cosine filter. The impulse response of the raised cosine filter is given by,

$$h_{RC}(t) = \frac{\sin(\pi t/T_s)}{\pi t/T_s} \frac{\cos(\pi \beta t/T_s)}{1 - 4\beta^2 t^2/T_s^2} \quad (9.8)$$

### B. Root raised cosine pulse shaper

The raised cosine pulse shaping filter has a transfer function given by,

$$H_{RC}(f) = \begin{cases} 1 & \text{for } |f| \leq \frac{1-\beta}{2T_s} \\ \sqrt{\frac{1}{2} \left[ 1 + \cos \left( \frac{\pi T_s}{\beta} \left[ |f| - \frac{1-\beta}{2T_s} \right] \right) \right]} & \text{for } \frac{1-\beta}{2T_s} < |f| \leq \frac{1+\beta}{2T_s} \\ 0 & \text{otherwise} \end{cases} \quad (9.9)$$

The parameter,  $\beta$  is the roll-off factor of the raised cosine filter. The impulse response of the root raised cosine filter is given by,

$$h_{RRC}(t) = \begin{cases} \frac{1}{T_s} \left( 1 + \beta \left( \frac{4}{\pi} - 1 \right) \right) & \text{for } t = 0 \\ \frac{\beta}{T_s \sqrt{2}} \left[ \left( 1 + \frac{2}{\pi} \right) \sin \left( \frac{\pi}{4\beta} \right) + \left( 1 - \frac{2}{\pi} \right) \cos \left( \frac{\pi}{4\beta} \right) \right] & \text{for } t = \frac{T_s}{4\beta} \\ \frac{1}{T_s} \frac{\sin \left[ \pi \frac{t}{T_s} (1 - \beta) \right] + 4\beta \frac{t}{T_s} \cos \left[ \pi \frac{t}{T_s} (1 + \beta) \right]}{\pi \frac{t}{T_s} \left[ 1 - \left( 4\beta \frac{t}{T_s} \right)^2 \right]} & \text{otherwise} \end{cases} \quad (9.10)$$

### C. Gaussian pulse shaper

The Gaussian pulse shaping filter has a transfer function given by,

$$H_G(f) = \exp(-\alpha^2 f^2) \quad (9.11)$$

The parameter  $\alpha$  is related to  $B$ , the 3-dB bandwidth of the Gaussian shaping filter is given by,

$$\alpha = \frac{\sqrt{\ln 2}}{\sqrt{2}B} = \frac{0.5887}{B} \quad (9.12)$$

From the equation 9.12, as  $\alpha$  increases, the spectral occupancy of the Gaussian filter decreases. The impulse response of the Gaussian filter can be given by,

$$h_G(t) = \frac{\sqrt{\pi}}{\alpha} \exp\left(-\frac{\pi^2}{\alpha^2} t^2\right) \quad (9.13)$$

From the equation 9.12, we can also write that,

$$\alpha = \frac{0.5887}{BT_s} T_s \quad (9.14)$$

Where,  $BT_s$  is the 3-dB bandwidth-symbol time product which ranges from  $0 \leq BT_s \leq 1$  given as the input parameter for designing the Gaussian pulse shaping filter.

## References

- [1] Sen M. (Sen-Maw) Kuo, Bob H. Lee, and Wenshun. Tian. *Real-time digital signal processing : fundamentals, implementations and applications*. ISBN: 9781118414323. URL: <https://www.wiley.com/en-us/Real+Time+Digital+Signal+Processing%7B%5C%7D3A+Fundamentals%7B%5C%7D2C+Implementations+and+Applications%7B%5C%7D2C+3rd+Edition-p-9781118414323>.
- [2] Theodore S. Rappaport. *Wireless communications : principles and practice*. Prentice Hall PTR, 2002, p. 707. ISBN: 0130422320.

## 9.4 Hilbert Transform

<b>Header File</b>	:	hilbert_filter_*.h
<b>Source File</b>	:	hilbert_filter_*.cpp
<b>Version</b>	:	20180306 (Romil Patel)

### What is the purpose of Hilbert transform?

The Hilbert transform facilitates the formation of analytical signal. An analytic signal is a complex-valued signal that has no negative frequency components, and its real and imaginary parts are related to each other by the Hilbert transform.

$$s_a(t) = s(t) + i\hat{s}(t) \quad (9.15)$$

where,  $s_a(t)$  is an analytical signal and  $\hat{s}(t)$  is the Hilbert transform of the signal  $s(t)$ . Such analytical signal can be used to generate Single Sideband Signal (SSB) signal.

### Transfer function for the discrete Hilbert transform

There are two approached to generate the analytical signal using Hilbert transformation method. First method generates the analytical signal  $S_a(f)$  directly, on the other hand, second method will generate the  $\hat{s}(f)$  signal which is multiplied with  $i$  and added to the  $s(f)$  to generate the analytical signal  $S_a(f)$ .

#### Method 1 :

The discrete time analytical signal  $S_a(t)$  corresponding to  $s(t)$  is defined in the frequency domain as [1] (This method requires MATLAB Hilbert transform definition)

$$S_a(f) = \begin{cases} 2S(f) & \text{for } f > 0 \\ S(f) & \text{for } f = 0 \\ 0 & \text{for } f < 0 \end{cases} \quad (9.16)$$

which is inverse transformed to obtain an analytical signal  $S_a(t)$ .

#### Method 2 :

The discrete time Hilbert transformed signal  $\hat{s}(f)$  corresponding to  $s(f)$  is defined in the frequency domain as [2]

$$\hat{S}(f) = \begin{cases} i S(f) & \text{for } f > 0 \\ 0 & \text{for } f = 0 \\ -i S(f) & \text{for } f < 0 \end{cases} \quad (9.17)$$

which is inverse transformed to obtain a Hilbert transformed signal  $\hat{S}(t)$ . To generate an analytical signal,  $\hat{S}(t)$  is added to the  $S(t)$  to get the equation 9.15.

### Real-time Hilbert transform : Proposed logical flow

To understand the new proposed method, consider that the signal consists of 2048 samples and the **bufferLength** is 512. Therefore, by considering the **bufferLength**, we will process the whole signal in four consecutive blocks namely *A*, *B*, *C* and *D*; each with the length of 512 samples as shown in Figure 9.34. The filtering process will start only after acquiring first

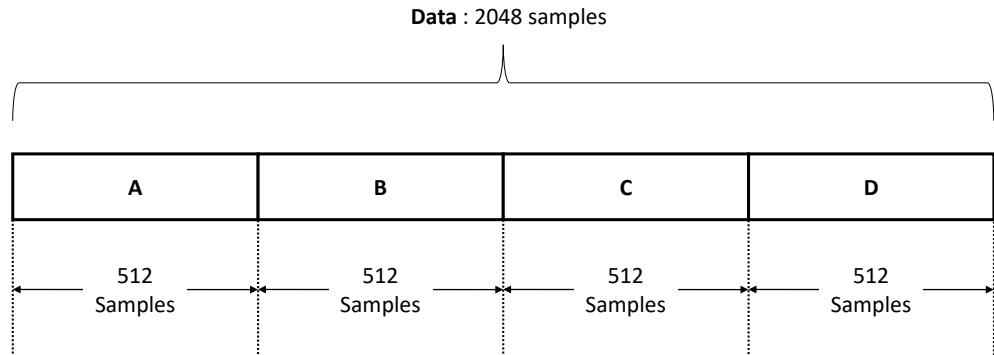


Figure 9.34: Logical flow

two blocks *A* and *B* (see iteration 1 in Figure 9.35), which introduces delay in the system. In the iteration 1,  $x(n)$  consists of 512 front Zeros, block *A* and block *B* which makes the total length of the  $x(n)$  is  $512 \times 3 = 1536$  symbols. After applying filter to the  $x(n)$ , we will capture the data which corresponds to the block *A* only and discard the remaining data from each side of the filtered output.

In the next iteration 2, we'll use **previousCopy** *A* and *B* along with the **currentCopy** "C"

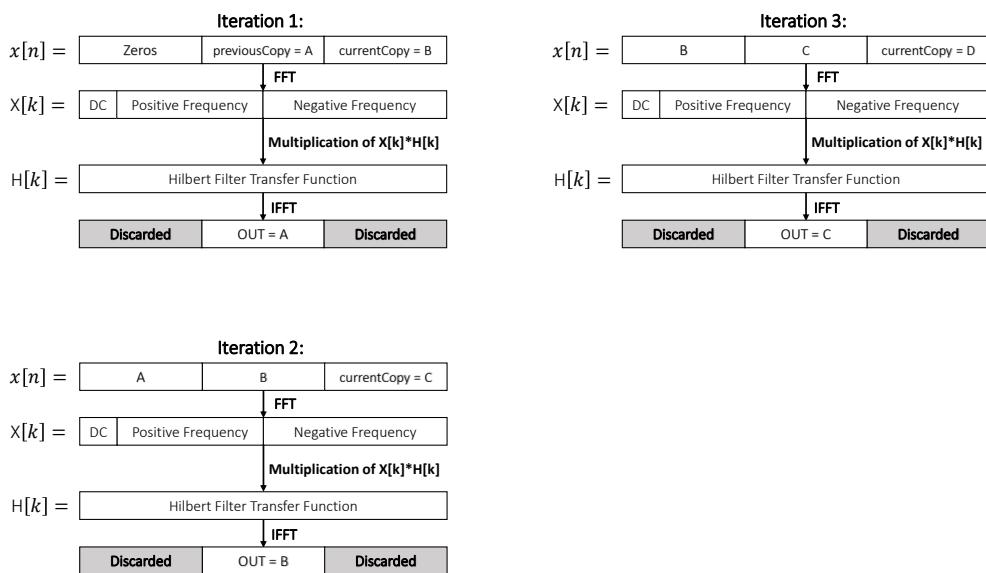


Figure 9.35: Logical flow of real-time Hilbert transform

and process the signal same as we did in iteration and we will continue the procedure until the end of the sequence.

### Real-time Hilbert transform : Test setup

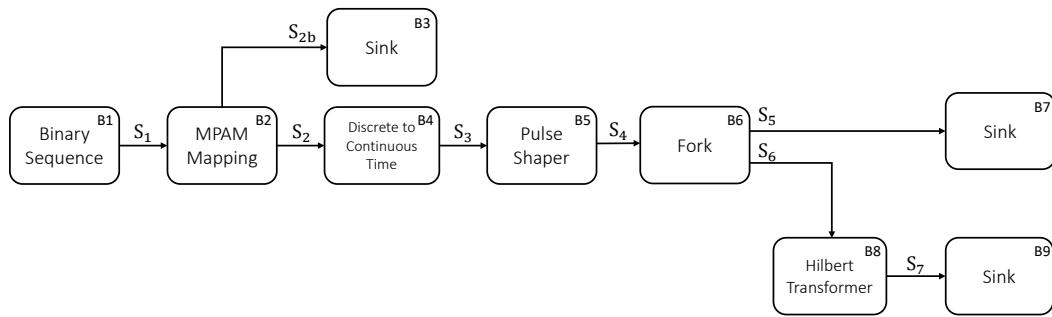


Figure 9.36: Test setup for the real time Hilbert transform

### Real-time Hilbert transform : Results

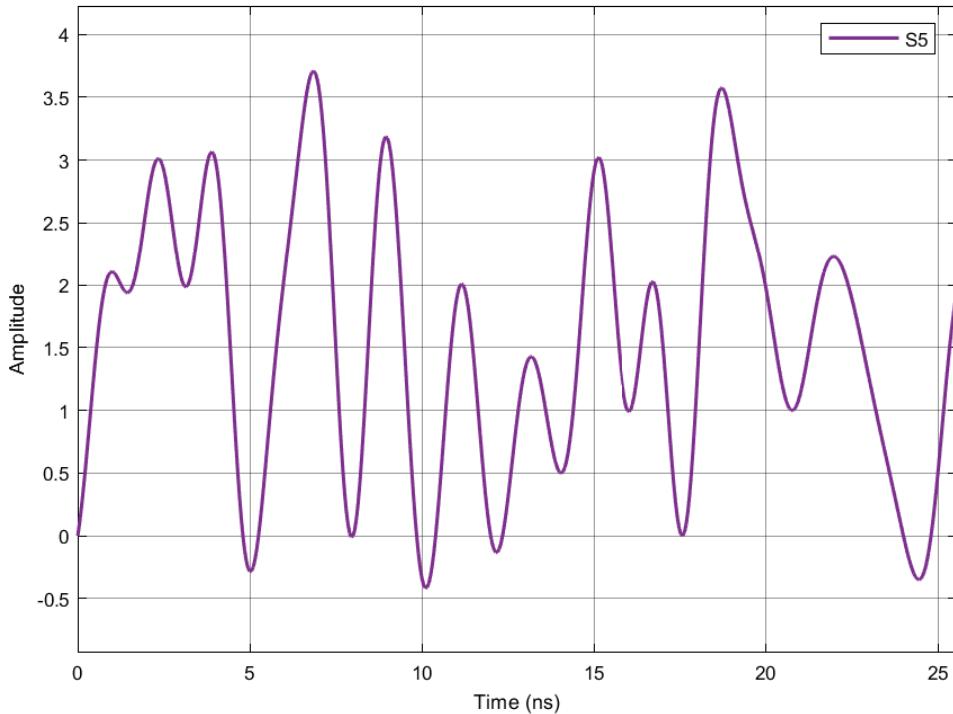


Figure 9.37:  $S_5$  signal

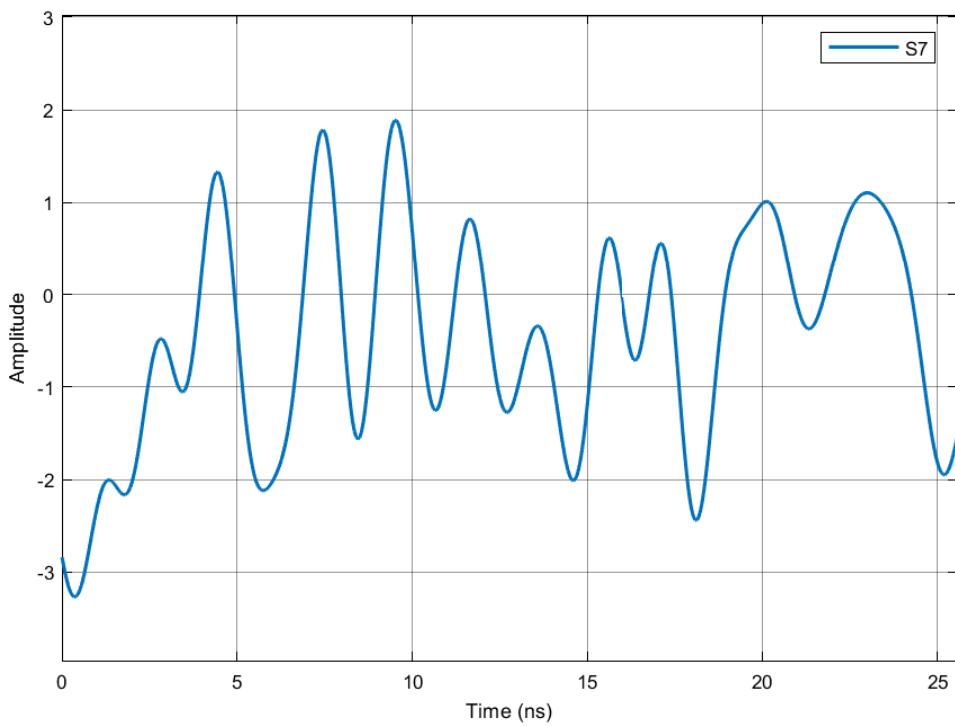


Figure 9.38:  $S7$  signal

**Remark :** Here, we have used method 2 to generate analytical signal using Hilbert transform. If you want to use method 1 then you should use  $ifft$  in place of  $fft$  and vice-versa.

## References

- [1] S.L. Marple. "Computing the discrete-time 'analytic' signal via FFT". In: *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136)*. Vol. 2. IEEE Comput. Soc, pp. 1322–1325. ISBN: 0-8186-8316-3. DOI: [10.1109/ACSSC.1997.679118](https://doi.org/10.1109/ACSSC.1997.679118). URL: <http://ieeexplore.ieee.org/document/679118/>.
- [2] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-time signal processing*. Prentice Hall, 1999, p. 870. ISBN: 0137549202.

## **Chapter 10**

---

## **Code Development Guidelines**

## Chapter 11

# Building C++ Projects Without Visual Studio

---

This is a guide on how to build C++ projects without having Microsoft Visual Studio installed. All the necessary files will be available in the `\msbuild\` folder on this repository.

## 11.1 Installing Microsoft Visual C++ Build Tools

Run the file `visualcppbuildtools_full.exe` and follow all the setup instructions;

## 11.2 Adding Path To System Variables

Please follow this step-by-step tutorial carefully.

1. Open the **Control Panel**.
2. Select the option **System and Security**.
3. Select the option **System**.
4. Select **Advanced System Settings** on the menu on the left side of the window.
5. This should have opened another window. Click on **Environment variables**.
6. Check if there is a variable called **Path** in the **System Variables** (bottom list).
7. If it doesn't exist, create a new variable by pressing **New** in **System Variables** (bottom list). Insert the name **Path** as the name of the variable and enter the following value `C:\Windows\Microsoft.Net\Framework\v4.0.30319`. Jump to step 10.
8. If it exists, click on the variable **Path** and press **Edit**. This should open another window;
9. Click on **New** to add another value to this variable. Enter the following value: `C:\Windows\Microsoft.Net\Framework\v4.0.30319`.
10. Press **Ok** and you're done.

## 11.3 How To Use MSBuild To Build Your Projects

You are now able to build (compile and link) your C++ projects without having Visual Studio installed on your machine. To do this, please follow the instructions below:

1. Open the **Command Line** and navigate to your project folder (where the .vcxproj file is located).
2. Enter the command:  
`msbuild <filename> /tv:14.0 /p:PlatformToolset=v140,TargetPlatformVersion=8.1,OutDir=".\"`, where <filename> is your .vcxproj file.

After building the project, the .exe file should be automatically generated to the current folder.

The signals will be generated into the sub-directory `\signals\`, which must already exist.

## 11.4 Known Issues

### 11.4.1 Missing ucrtbased.dll

In order to solve this issue, please follow the instructions below:

1. Navigate to `C:\Program Files (x86)\Windows Kits\10\bin\x86\ucrt\`
2. Copy the following file: `ucrtbased.dll`
3. Paste this file in the following folder: `C:\Windows\System32\`
4. Paste this file in the following folder: `C:\Windows\SysWOW64\`

**Attention:**you need to paste the file in BOTH of the folders above.

## Chapter 12

### Git Helper

Git creates and maintains a database that store versions of a repository, i.e. versions of a folder. To create this database for a specific folder the Git application must be installed on the computer. Open the Git console program and go to the specific folder and execute the following command:

```
git init
```

The Git database is created and stored in the folder `.git` in the root of your repository. The Git commands allow you to manipulate this database.

#### 12.1 Data Model

To understand Git is fundamental to understand the Git data model.

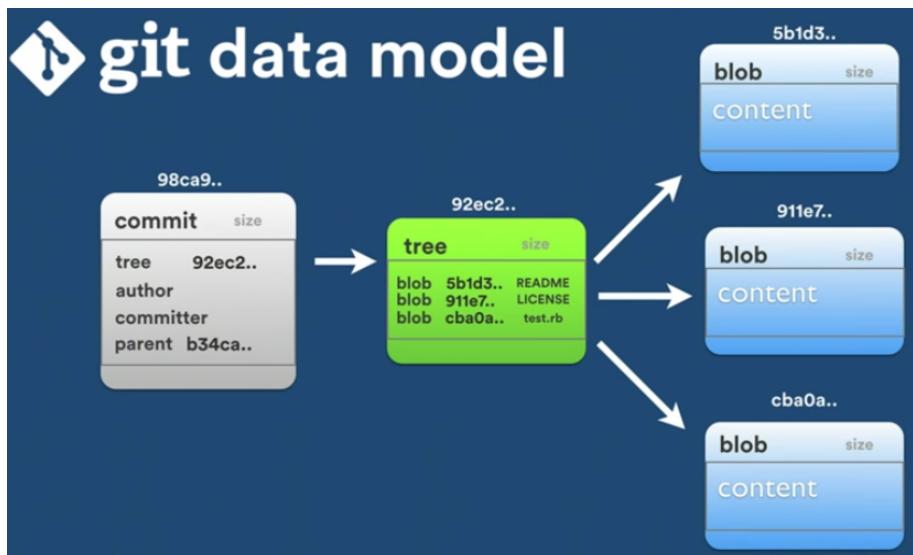


Figure 12.1: Git data model.

Git manipulates the following objects:

- commits - text files that store a description of the repository;
- trees - text files that store a description of a folder;

- blobs - the files that exist in your repository.

The objects are stored in the folder `.git/objects`. Each store object is identified by its SHA1 hash value, i.e. 20 bytes which identifies unequivocally the object. Note that 20 bytes can be represented by a 40 characters hexadecimal string. The identifier of each object is the 40 characters hexadecimal string. Each particular object is stored in a sub-folder inside the `.git/objects`. The name of the sub-folder is the two most significative characters of the SHA1 hash value. The name of the file that is inside the sub-folder is the remaining thirty eight characters of the SHA1 hash value. The Git stores all committed versions of a file. The Git maintains a content-addressable file systems, i.e. a file system in which the files can be accessed based on its content.

A commit object is identified by a SHA1 hash value, and has the following information: a pointer for a tree (the root of your repository), a pointer for the previous commit, the author of the commit, the committer and a commit message. The author is the person who did the work. The committer is the person who validate the work and who apply the work by doing the commit. By doing this difference git allow both to get the credit, the author and the committer. Example of a commit file contend:

```
tree 2c04e4bad1e2bcc0223239e65c0e6e822bba4f16
parent bd3c8f6fed39a601c29c5d101789aaa1dab0f3cd
author NetXPTO <netxpto@gmail.com> 1514997058 +0000
committer NetXPTO <netxpto@gmail.com> 1514997058 +0000
```

Here goes the commit message.

A tree object is identified by a SHA1 hash value, and has a list of blobs and trees that are inside that tree. Example of a tree file contend:

100644	blob	bdb0cabc87cf50106df6e15097dff816c8c3eb34	.gitattributes
100644	blob	50492188dc6e12112a42de3e691246dafdad645b	.gitignore
100644	blob	8f564c4b3e95add1a43e839de8adbfd1ceccf811	bfg-1.12.16.jar
040000	tree	de44b36d96548240d98cb946298f94901b5f5a05	doc
040000	tree	8b7147dbfdc026c78fee129d9075b0f6b17893be	garbage
040000	tree	bdfcd8ef2786ee5f0f188fc04d9b2c24d00d2e92	include
040000	tree	040373bd71b8fe2fe08c3a154cada841b3e411fb	lib
040000	tree	7a5fce17545e55d2faa3fc3ab36e75ed47d7bc02	msbuild
040000	tree	b86efba0767e0fac1a23373aaf95884a47c495c5	mtools
040000	tree	1f981ea3a52bccf1cb00d7cb6dfdc687f33242ea	references
040000	tree	86d462afd7485038cc916b62d7cbfc2a41e8cf47	sdf
040000	tree	13bfce10b78764b24c1e3dfbd0b10bc6c35f2f7b	things_to_do
040000	tree	232612b8a5338ea71ab6a583d477d41f17ebae32	visualizerXPTO
040000	tree	1e5ee96669358032a4a960513d5f5635c7a23a90	work_in_progress

A blob is identified by a SHA1 hash value, and has the file contend compressed. A git header

and tailor is added to each file and the file is compressed using the zlib library. The git header is just the file type, file size and the \NUL character, for instance "blob 13\NUL", the tailor is just the \n character. The blob is stored as a binary file.

## 12.2 Refs

SHA1 hash values are hard to memorize by humans. To make life easier to humans we use refs. A ref associate a name, easier to memorize by humans, with a SHA1 hash value. Therefore refs are pointers to objects. Refs are implemented by text files, the name of the file is the name of the ref and inside the file is a string with the SHA1 hash value.

There are different type of refs. Some are static, for instance the tags, others are actualized automatically, for instance the branches.

## 12.3 Tags

A tag is just a ref for a specific commit. A tag do not change over time.

## 12.4 Branch

A branch is a ref that points for a commit that is originated by a divergence from a common point. A branch is automatically actualized so that it always points for the most recent commit of that branch.

## 12.5 Heads

Heads is a pointer for the commit where you are.

## 12.6 Database Folders and Files

### 12.6.1 Objects Folder

Git stores the database and the associated information in a set of folders and files inside the folder `.git` in the root of your repository.

The folder `.git/objects` stores information about all objects (commits, trees and blobs). The objects are stored in files inside folders. The name of the folders are the 2 first characters of the SHA1 40 characters hexadecimal string. The name of the files are the other 38 hexadecimal characters of the SHA1. The information is compressed to save same space but it can be access using some applications.

### 12.6.2 Refs Folder

The `.git/refs` folder has inside the following folders `heads`, `remotes`, and `tags`. The `heads` has inside a ref for all local branches of your repository. The `remotes` folder has inside a set of folders with the name of all remote repositories, inside each folder is a ref for all branches in that remote repository. The `tag` folder has a ref for each tag.

## 12.7 Git Spaces

Git uses several spaces.

- workspace - is your directories where you are working;
- index - when you record changes before commit them;
- blobs - the files that exist in your repository.

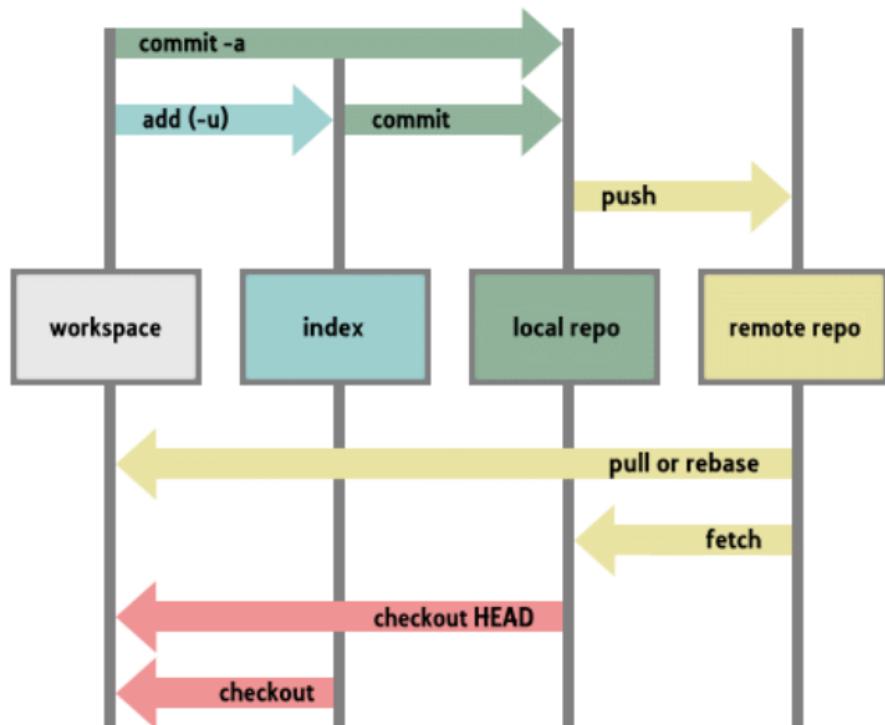


Figure 12.2: Git spaces.

## 12.8 Workspace

## 12.9 Index

## 12.10 Merge

Merge is a fundamental concept to git. It is the way you consolidate your work.

### 12.10.1 Fast-Forward Merge

### 12.10.2 Resolve

### 12.10.3 Recursive

### 12.10.4 Octopus

### 12.10.5 Ours

### 12.10.6 Subtree

### 12.10.7 Custom

## 12.11 Commands

### 12.11.1 Porcelain Commands

#### **git add**

*git add*, store a file that was changed to the index.

#### **git bisect**

#### **git branch**

*git branch -set-upstream-to=<remote>/<branch> <local branch>*, links a local branch with a branch in a remote repository.

#### **git cat-file**

*git cat-file -t <hash>*

shows the type of the objects

*git cat-file -p <hash>*

shows the contend of the object

#### **git clone**

#### **git diff**

*git diff* shows the changes in the working space.

*git diff -name-only* shows the changes in the working space, only file names.

*git diff -cached* shows the changes in the index space.

*git diff -cached -name-only* shows the changes in the index space, only file names.

#### **git fetch -all**

#### **git init**

*git init*, used to initialize a git repository. It creates the *.git* folder and all its subfolders and files. The subfolders are *objects*, *refs*, ... There are also the following files *HEAD*.

### **git log**

shows a list of commits from reverse time order (goes back on time), i.e. shows the history of your repository. The history of a repository can be represented by a directed acyclic graph (dag for short), pointing in the forward direction in time.

options

*-graph*

shows a graphical representation of the repository commits history.

### **git rebase**

*git rebase <branch2 or commit2>*, finds a common point between the current branch and branch2 or commit2, reapply all commits of your current branch from that divergent points on top of branch2 or commit2, one by one.

### **git reset**

*git reset --soft HEAD 1*, moves one commit back but keeps all modified files.

*git reset --hard HEAD 1*, moves one commit back and cleans all the modified files.

### **git reflog**

Keep a log file with all commands from the last 90 days.

### **git show**

Shows what is new in the last commit.

### **git stash**

Stash is a global branch where you can store the present state.

*git stash*, save the present state of your repository.

*git stash -list*, shows what is in the stash.

### **git status**

## 12.11.2 Pluming Commands

### **git count-object**

*git count-object -H*, counts all object and shows the result in a readable form (-H, human).

### **git gc**

Garbage collector. Eliminates all objects that are not referenced, i.e. has no reference associated with.

*git gc --prune=all*

### **git hash-object**

*git hash-object -w <file>*, calculates the SHA1 hash value of a file and write it in the *.git/objects* folder.

### **git cat-files**

*git cat-files -p <sha1>*, shows the contend of a file in a readable format (flag -p, pretty format).

*git cat-files -t <sha1>*, shows the type of a file.

### **git update-index**

*git update-index --add <file name>*, creates the hash and adds the <file\_name> to the index.

**git ls-files**

*git ls-files -stage*, shows all files that you are tracking.

**git write-tree****git commit-tree****git update-ref**

*git update-ref refs/heads/<branch name> <commit sha1 value>*, creates a branch that points to the <commit sha1 value>.

**git verify-pack**

## 12.12 The Configuration Files

There is a config file for each repository that is stored in the *.git/* folder with the name *config*.

There is a config file for each user that is stored in the *c:/users/<user name>/* folder with the name *.gitconfig*.

To open the *c:/users/<user name>/.gitconfig* file type:

**git config --global -e**

## 12.13 Pack Files

Pack files are binary files that git uses to save data and compress your repository. Pack files are generated periodically by git or with the use of gc command.

## 12.14 Applications

### 12.14.1 Meld

### 12.14.2 GitKraken

## 12.15 Error Messages

### 12.15.1 Large files detected

Clean the repository with the **BFG Repo-Cleaner**.

Run the Java program:

```
java -jar bfg-1.12.16.jar --strip-blobs-bigger-than 100M
```

This program is going to remote from your repository all files larger than 100MBytes. After

do:

```
git push --force.
```

## Chapter 13

# Simulating VHDL Programs with GHDL

This guide will help you simulate VHDL programs with the open-source simulator GHDL.

### 13.1 Adding Path To System Variables

Please follow this step-by-step tutorial:

1. Open the **Control Panel**.
2. Select the option **System and Security**.
3. Select the option **System**.
4. Select **Advanced System Settings** on the menu on the left side of the window.
5. This should have opened another window. Click on **Environment variables**.
6. Check if there is a variable called **Path** in the **System Variables** (bottom list).
7. **If it doesn't exist**, create a new variable by pressing **New** in **System Variables** (bottom list). Insert the name **Path** as the name of the variable and enter your absolute path to the folder `\LinkPlanner\vhdl_simulation\ghdl\bin`.  
Example: `C:\repos\LinkPlanner\vhdl_simulation\ghdl\bin`.  
Jump to step 10.
8. **If it exists**, click on the variable **Path** and press **Edit**. This should open another window;
9. Click on **New** to add another value to this variable. Enter your absolute path to the folder `\LinkPlanner\vhdl_simulation\ghdl\bin`.  
Example: `C:\repos\LinkPlanner\vhdl_simulation\ghdl\bin`.
10. Press **Ok** and you're done.

## 13.2 Using GHDL To Simulate VHDL Programs

This guide will only cover the simulation of the VHDL module in this repository. This simulation will take an .sgn file and output its binary information, removing the header. There are two ways to simulate this module.

### 13.2.1 Requirements

Place a .sgn file in the directory `\vhdl_simulation\input_files\` and rename it to `SIGNAL.sgn`

### 13.2.2 Option 1

Execute the batch file `simulation.bat`, located in the directory `\vhdl_simulation\` in this repository.

### 13.2.3 Option 2

Open the **Command Line** and navigate to your project folder (where the .vhd file is located). Execute the following commands:

```
ghdl -a -std=08 signal_processing.vhd
ghdl -a -std=08 vhdl_simulation.vhd
ghdl -e -std=08 vhdl_simulation
ghdl -r -std=08 vhdl_simulation
```

**Additional information:** The first two commands are used to compile the program and will generate .cf files. Do not remove these file until the simulation is complete.

The third command is used to elaborate the simulation.

The last command is used to run the simulation. If you want to simulate the same program again, you will just need to execute this command (as long as you don't delete the .cf files).

### 13.2.4 Simulation Output

The simulation will output the file `SIGNAL.sgn`. This file will contain all the processed binary information of the input file, with the header.

