NetXPTO - LinkPlanner

Armando Nolasco Pinto

February 20, 2019

Contents

Chapter 1

Preface

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, δ. 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.

3.3.1 Circular Buffer

The signals use a circular buffer to store data. Because standard C++ do not have a circular buffer container (at least up to ISO C++17) one was developed. The circular buffer was developed using the same principles and style of the other STL containers, trying to make sure that the future integration of a standard circular buffer in our code will as easy as possible. In this development we use the following references [1, 2, 3]. In [1] a simple circular buffer implementation is presented, in [2] a standard like version of a circular buffer is discussed, and in [3] a comparative assessment is presented considering different implementation strategies. We try to follow [2] as possible.

A circular buffer is a fixed-size container that works in a circular way, the default buffer size is 512. A circular buffer uses a begin and a end pointer to control where data is going to be retrieved (consumed) or added. A full buffer flag is also used to signal the full buffer situation.

Initially, the begin and the end are made to coincide and the full flag is set to false. This is the empty buffer state. When data is added, the end pointer advances. After adding data if the end and the begin pointer coincide the buffer is full.

When data is retrieved, the begin pointer advances. After retrieving data if the begin and end pointer coincide the buffer is empty.

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 Inicialization #############################
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**
logFileName	string	"log.txt"
logValue	bool	true

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

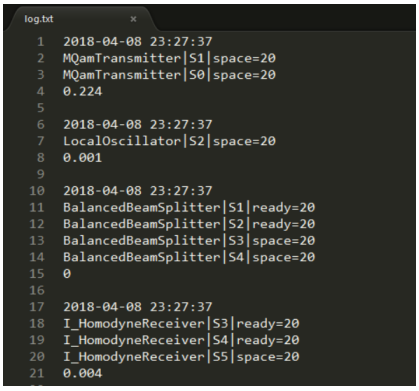

Figure 3.2: Output File Example

Figure 3.3 shows a portion of code that consists in the declaration and inicialization of the I_HomodyneReceiver block. In line 97, we can see that the block has 2 input signals, $S3$ and $S4$, and is assigned 1 output signal, $S5$. Going back to Figure 3.2 we can observe that $S3$ and $S4$ have 20 samples ready to be processed and the buffer of $S5$ 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*

References

[1] URL: https://embeddedartistry.com/blog/2017/4/6/circular-buffers-in-cc.

[2] URL: https://www.boost.org/doc/libs/1_68_0/doc/html/circular_buffer.html.

[3] URL: https://www.codeproject.com/Articles/1185449/Performance-of-a-Circular-Buffer-vs-Vector-Deque-a.

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

We are going to explain the use of the IPS using as an example the PBSK system. 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.

3.6 Documentation

As in any large software system documentation it is critical. The documentation is going to be developed in Latex using WinEdt as the recommend editor. The bibliography is per section, for this to work replace the bibtex by biber, go to the WinEdt Options->Execution Modes->Bibtex and replace bibtex.exe by biber.exe.

Chapter 4

Development Cycle

The NetXPTO-LinkPlanner is a open source project with its core implemented using ISO C++. At the present the followed standard is the ISO C++14.

The developed environment has been the Visual Studio Community 2017, namely release 15.5 and beyond. The Git has been used as the version control system. The NetXPTO-LinkPlanner repository is located in the GitHub site http://github.com/netxpto/linkplanner. Master branch should be considered a functional beta version of the software. Periodically new releases are delivered from the master branch under the branch name R<Release Year>-<Release Number>. The design and integration of the system has been performed by Prof. Armando Pinto.

Chapter 5

Visualizer

The visualizer is based on a customization of the the Matlab SPTOOL (https://www.mathworks.com/help/signal/ref/sptool.html) application.

Chapter 6

Case Studies

Chapter 7

Library

7.1 Binary Source

Header File	:	binary_source_*.h
Source File	:	binary_source_*.cpp
Version	:	20180118 (Armando Pinto)
	:	20180523 (André Mourato)

This block generates a sequence of binary values (1 or 0) and it can work in four different modes:

1. Random
2. PseudoRandom
3. DeterministicCyclic
4. DeterministicAppendZeros
5. AsciiFileAppendZeros
6. AsciiFileCyclic

Signals

Number of Input Signals	0
Type of Input Signals	-
Number of Output Signals	$\geq$
Type of Output Signals	Binary

Table 7.1: Binary source signals

Input Parameters

Parameter	Type	Values	Default
mode	string	Random, PseudoRandom, DeterministicCyclic, DeterministicAppendZeros, AsciiFileAppendZeros, AsciiFileCyclic	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	$1.0/100e9$
asciiFilePath	string	any	"file_input_data.txt"

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-*probabilityOfZero*.

Pseudorandom Mode Generates a pseudorandom sequence with period $2^{patternLength} - 1$.

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)

Illustrative Examples

Random Mode

PseudoRandom Mode 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.

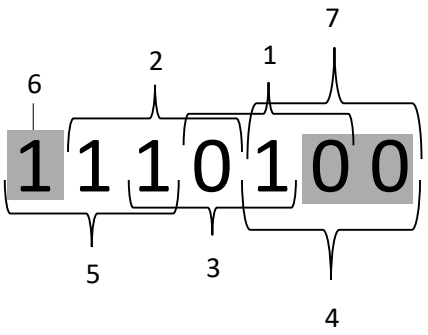

Figure 7.1: Example of a pseudorandom sequence with a pattern length equal to 3.

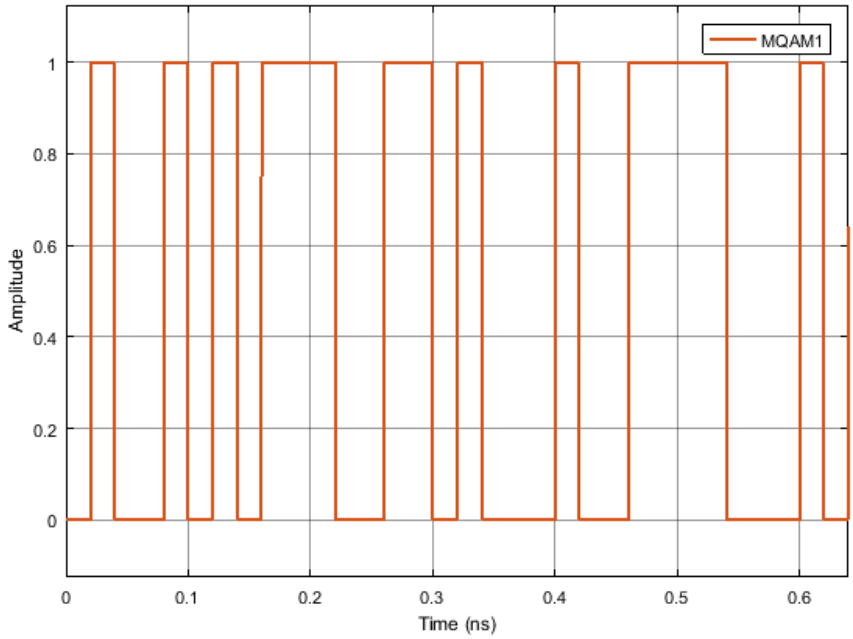

Figure 7.2: Binary signal generated by the block operating in the *Deterministic Append Zeros* mode with a binary sequence 01000...

DeterministicCyclic Mode Take the *bit stream* '0100011101010101'. The generated binary signal is displayed in.

DeterministicAppendZeros Mode Take the *bit stream* '0100011101010101'. The generated binary signal is displayed in 7.2.

Suggestions for future improvement

Implement an input signal that can work as trigger.

7.2 Discrete To Continuous Time

Header File	:	discrete_to_continuous_time.h
Source File	:	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 7.3: 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

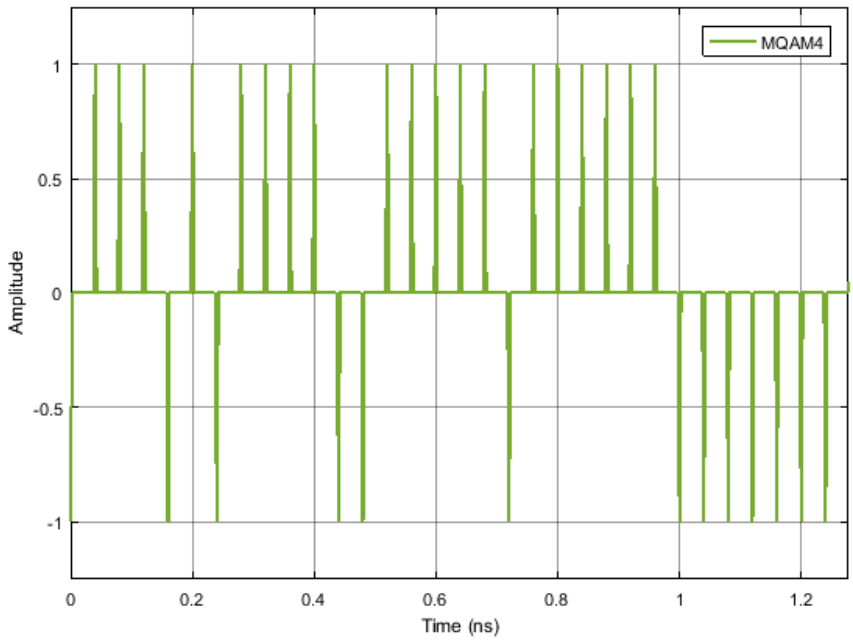

Figure 7.3: Example of the type of signal generated by this block for a binary sequence 0100...

7.3 IQ Modulator

Header File	:	iq_modulator.h
Source File	:	iq_modulator.cpp
Source File	:	20180130
Source File	:	20180828 (Romil Patel)

Version 20180130

This blocks accepts one inupt 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 7.4: 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 transmited to the receptor. The binary signal is sent to the Bit Error Rate (BER) meaurement 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

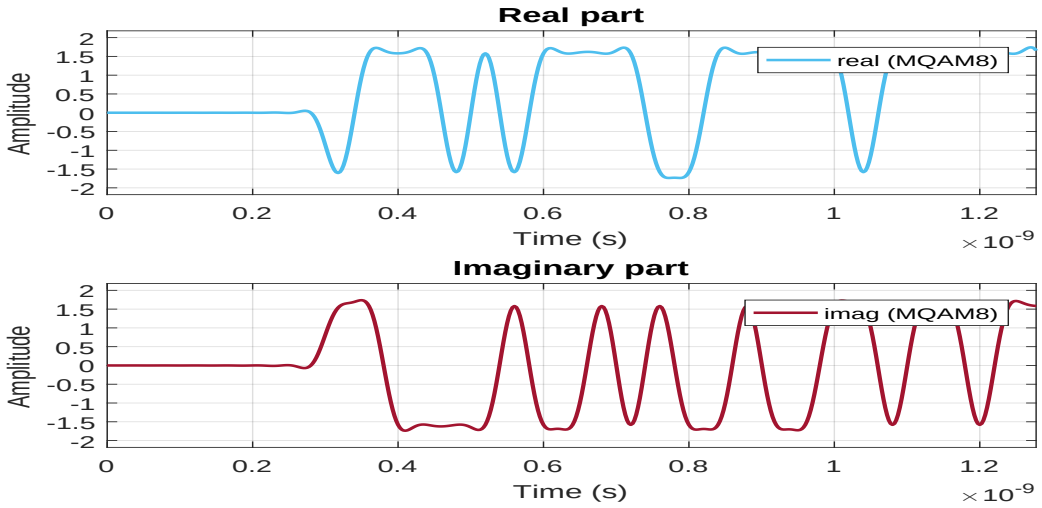

Figure 7.4: Example of a signal generated by this block for the initial binary signal 0100...

Version 20180828

Input Parameters:

—NA—

Input Signals:

Number: 1, 2,3
Type: RealValue

Output Signals:

Number: 4
Type: RealValue

Functional Description

This blocks has three inputs and one output. Port number 1 and 2 accept the real and imaginary data respectively and port 3 accepts the local oscillator as an input to the IQ modulator. This model serves as an ideal IQ modulator without noise and introduction of nonlinearity.

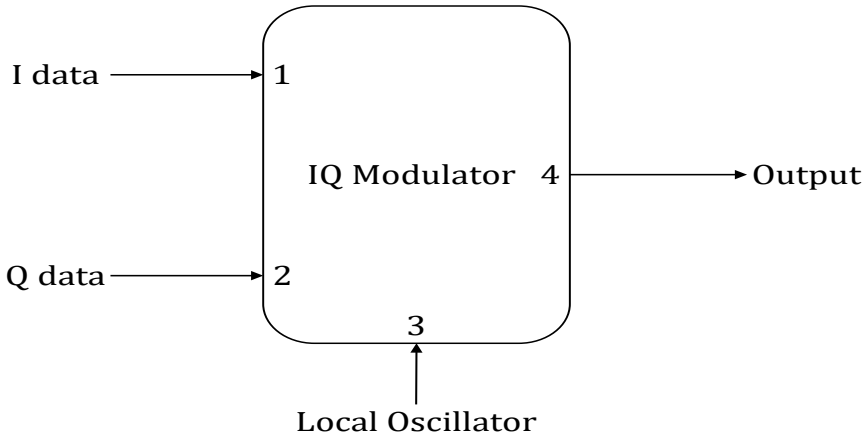

Figure 7.5: IQ Modulator block

IQ MZM Description

The detailed expatiation of the MZM starts with the phase modulator (see Figure **??**). The transfer function of the phase modulator can be given as,

$$E_{out}(t) = E_{in}(t) \cdot e^{j\phi_{PM}(t)} = E_{in}(t) \cdot e^{j\frac{u(t)}{V_\pi}\pi}$$

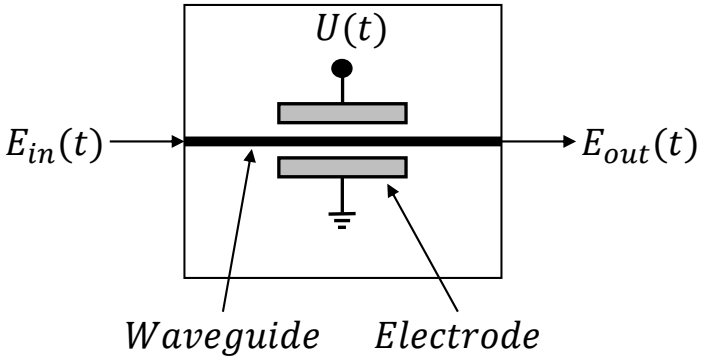

Figure 7.6: Phase Modulator

Two phase modulators can be placed in parallel using an interferometric structure as shown in Figure **??**. The incoming light is split into two branches, different phase shifts applies to each path, and then recombined. The output is a result of interference, ranging from constructive (the phase of the light in each branch is the same) to destructive (the phase in each branch differs by π). The transfer function of the structure can be given as,

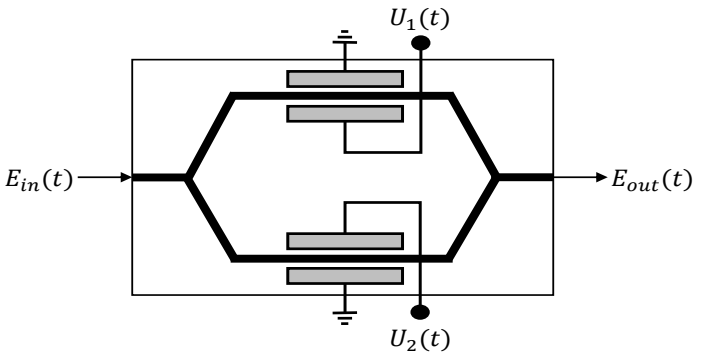

Figure 7.7: Mach-Zehnder Modulator

$$\frac{E_{out}(t)}{E_{in}(t)} = \frac{1}{2} \cdot (e^{j\phi_1(t)} + e^{j\phi_2(t)}) \tag{7.1}$$

Where, $\phi_1(t) = \frac{u_1(t)}{V_{\pi_1}}\pi$ and $\phi_2(t) = \frac{u_2(t)}{V_{\pi_2}}\pi$. if the inputs are set to $u_1 = u_2$ (push-push operation) then it provides the pure phase modulation at the output. Alternatively, if the inputs are set to $u_1 = -u_2$ (push-pull operation) then it provides pure amplitude modulation at the output.

The structure of the IQ MZM can be represented shown in Figure **??** where the incoming source light spitted into two portions. The first portion will drive the MZM of the I-channel and other portion will drive MZM Q-channel data. In the Q-channel, before feeding it to the MZM, it passed though the phase modulator to provide a $\pi/2$ phase shift to the carrier. The output of the MZM combined to form the electrical field $E_{out}(t)$ [1]. The transfer function of

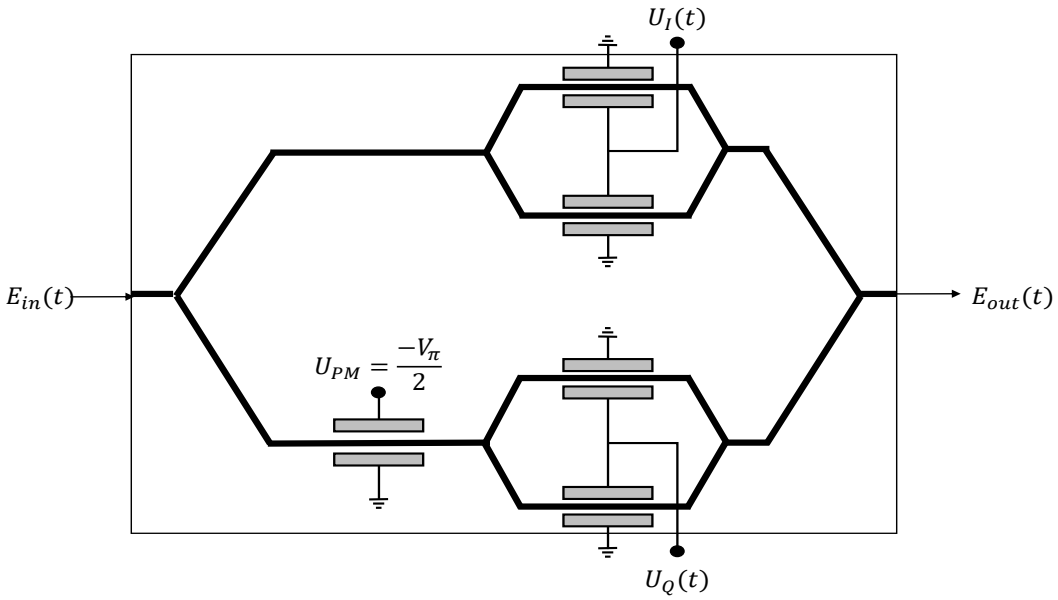

Figure 7.8: IQ Mach-Zehnder Modulator

the IQ MZM can be written as,

$$E_{out}(t) = \frac{1}{2}E_{in}(t)\left[cos\left(\frac{\pi U_I(t)}{2V_\pi}\right) + j \cdot cos\left(\frac{\pi U_Q(t)}{2V_\pi}\right)\right] \tag{7.2}$$

The black box model of the IQ MZM in the simulator can be depicted as,

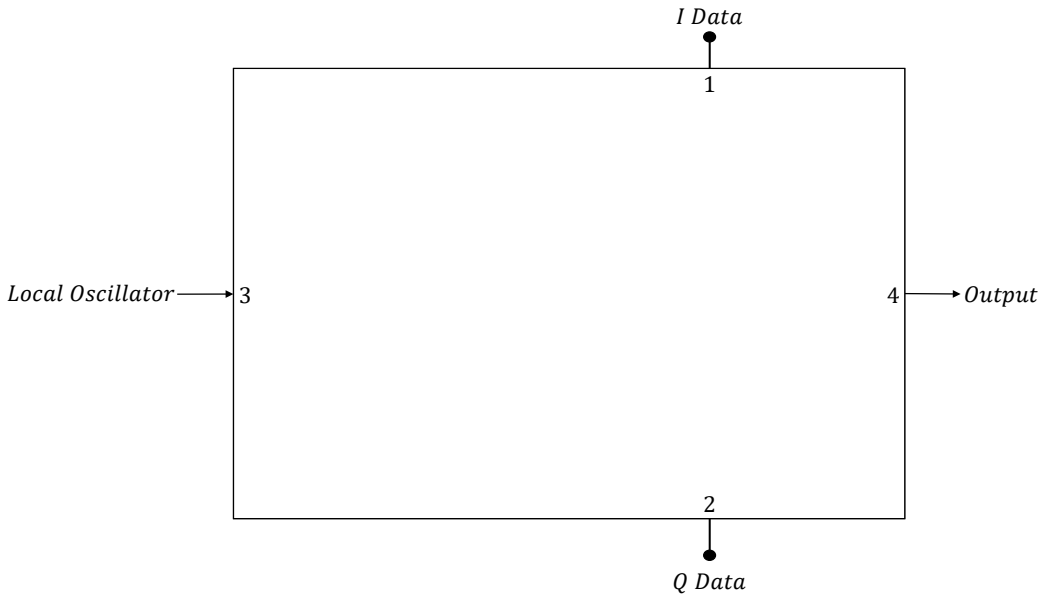

Figure 7.9: Simulation model of the IQ Mach-Zehnder Modulator

References

[1] *National Programme on Technology Enhanced Learning (NPTEL) :: Electronics & Communication Engineering : Optical communications*. URL: https://nptel.ac.in/courses/117104127/5.

7.4 Local Oscillator

Header File	:	local_oscillator.h
Source File	:	local_oscillator.cpp
Version	:	20180130
Version	:	20180828 (Romil Patel)

Version 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	$\in \left[0, \frac{\pi}{2}\right]$	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 7.5: 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

Version 20180828

7.5 MQAM Mapper

Header File : m_qam_mapper.h
Source File : 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 7.6: 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 atributes 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 ilustrated in figure 7.10.

Input Signals

Number : 1

Type : Binary (DiscreteTimeDiscreteAmplitude)

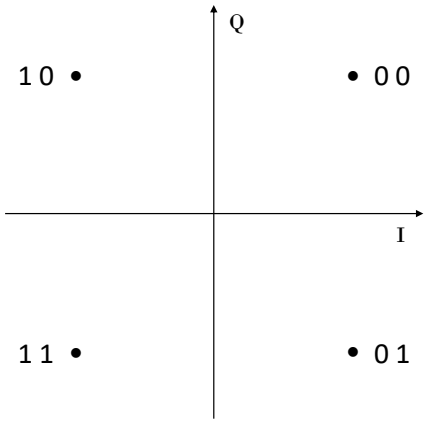

Figure 7.10: Constellation used to map the signal for m=4

Output Signals

Number : 2

Type : Sequence of 1's and -1's (DiscreteTimeDiscreteAmplitude)

Example

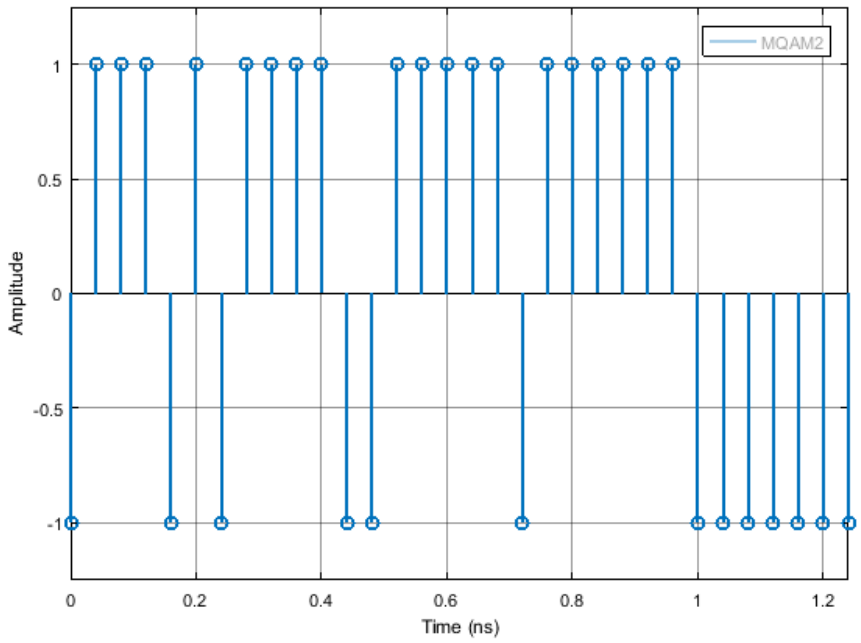

Figure 7.11: Example of the type of signal generated by this block for the initial binary signal 0100...

7.6 Pulse Shaper

Header File	:	pulse_shaper.h
Source File	:	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
rollOfFactor	real	$\in [0, 1]$	0.9

Table 7.7: 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 trough 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 *rollOfFactor* 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

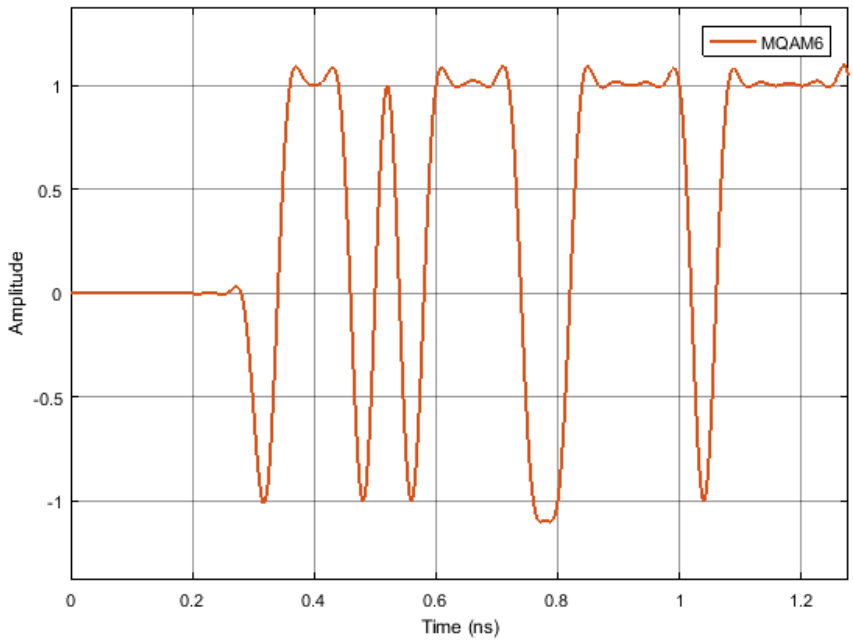

Figure 7.12: Example of a signal generated by this block for the initial binary signal 0100...

Sugestions for future improvement

Include other types of filters.

7.7 Sink

Header File	:	sink_*.h
Source File	:	sink_*.cpp
Version	:	20180523 (André Mourato)

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
displayNumberOfSamples	bool	true/false	true

Table 7.8: Sink input parameters

Methods

Sink(vector<Signal *> &InputSig, vector<Signal *> &OutputSig)

bool runBlock(void)

void setAsciiFilePath(string newName)

string getAsciiFilePath()

void setNumberOfBitsToSkipBeforeSave(long int newValue)

long int getNumberOfBitsToSkipBeforeSave()

void setNumberOfBytesToSaveInFile(long int newValue)

long int getNumberOfBytesToSaveInFile()

void setNumberOfSamples(long int nOfSamples)

long int getNumberOfSamples const()

void setDisplayNumberOfSamples(bool opt)

bool getDisplayNumberOfSamples const()

Functional Description

The Sink block discards all elements contained in the signal passed as input. After being executed the input signal's buffer will be empty.

Chapter 8

Mathlab Tools

8.1 Generation of AWG Compatible Signals

Students Name	:	Francisco Marques dos Santos
	:	Romil Patel
Goal	:	Convert simulation signals into waveform files compatible with the laboratory's Arbitrary Waveform Generator
Version	:	sgnToWfm.m (**Student Name**: Francisco Marques dos Santos)
	:	sgnToWfm_20171119.m (**Student Name**: 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 (Tekatronix AWG70002A). In order to recreate it appropriately, the signal must be real, not exceed $8 * 10^9$ samples and have a sampling rate equal or bellow 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
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).

Decimation factor calculation

The flowchart for calculating the decimation factor is as follows:

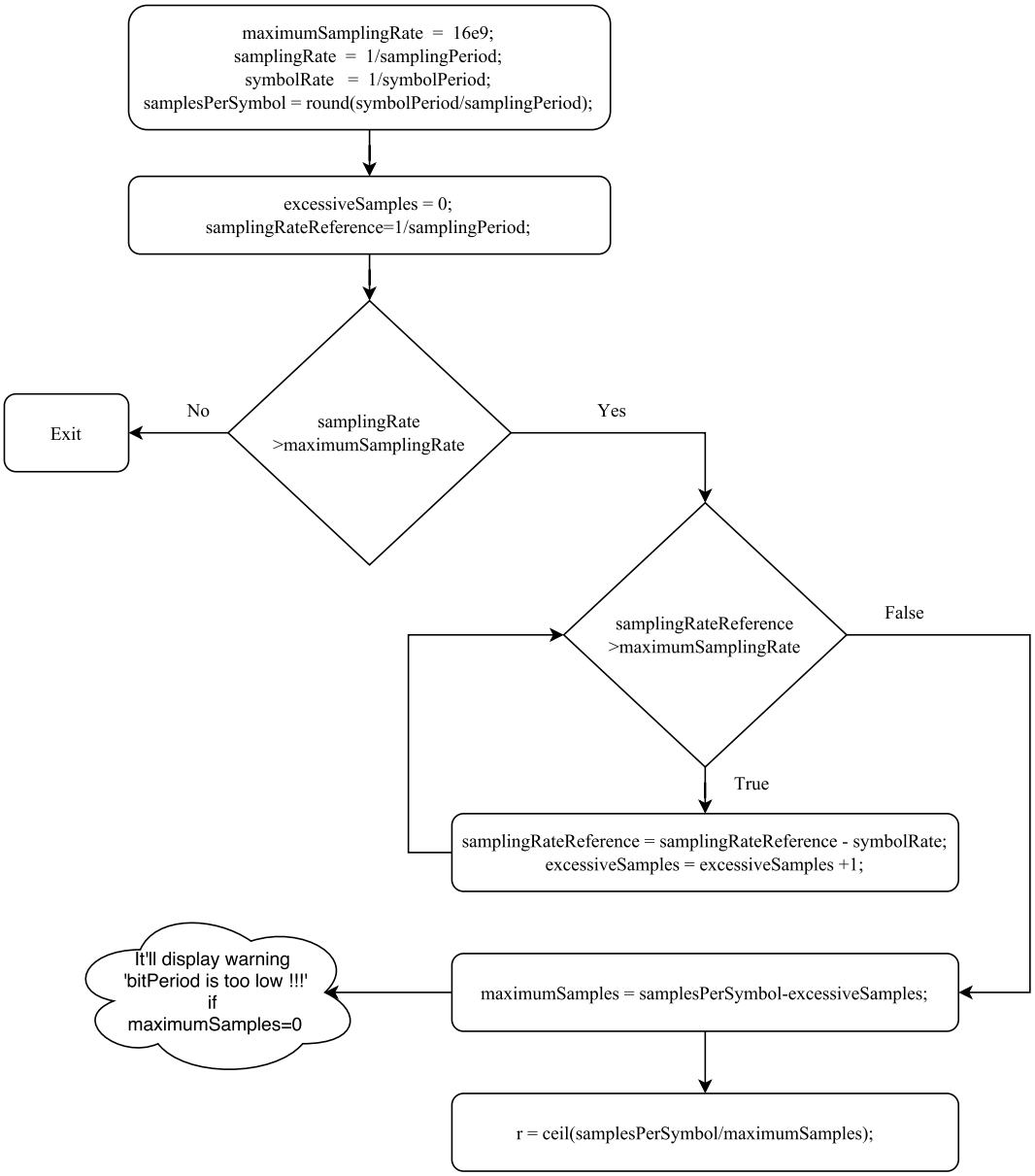

Figure 8.1: Flowchart to calculate decimation factor

»»»»> Develop.Romil

8.1.3 Loading a signal to the Tekatronix AWG70002A

The AWG we will be using is the Tekatronix 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).

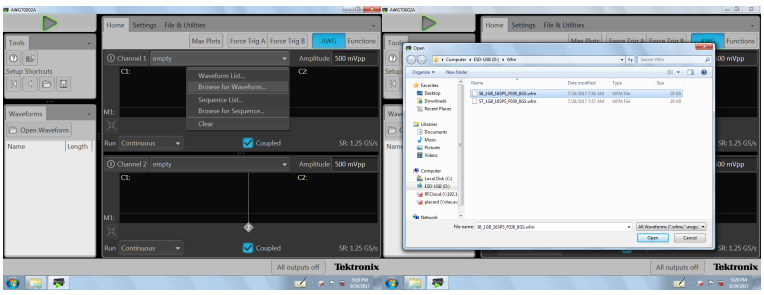

Figure 8.2: Selecting your waveform in the AWG

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).

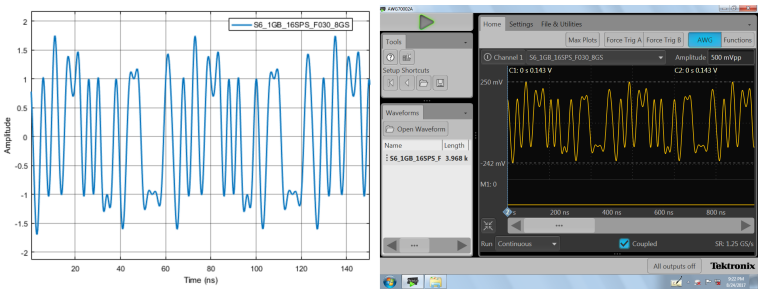

Figure 8.3: Comparison between the waveform in the AWG and the original signal before configuring the sampling rate

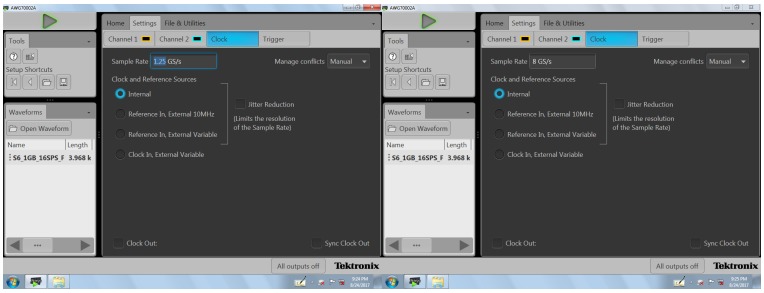

Figure 8.4: Configuring the right sampling rate

4. Generate the signal: Output the wave by enabling the channel you want and clicking on the play button.

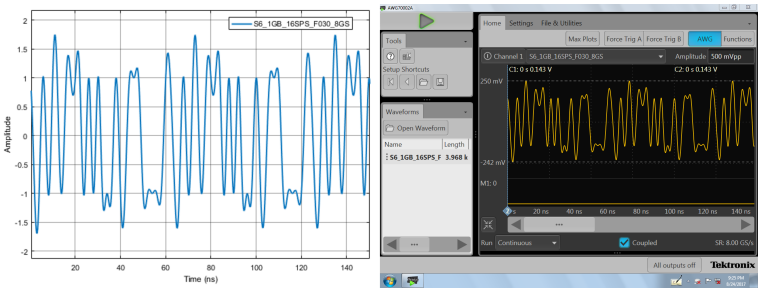

Figure 8.5: Comparison between the waveform in the AWG and the original signal after configuring the sampling rate

8.2 Polarization Analysis Signals

Students Name	:	Mariana Ramos
Goal	:	Analyse simulation Photon Stream signals into Stokes space using plot in Poincare sphere and Autocorrelation calculation and plot.
Version	:	jonesToStokes_20180614.m

This section shows some matlab functions to analyse photon stream signals from simulation.

8.2.1 jonesToStokes.m

Structure of a function

[] = jonesToStokes(fname,deltaP, filename,NumberOfSamples);

Inputs

fname: Input filename of the signal (*.sgn) you want to convert. It must be a photon stream signal (Type: PhotonStreamXY).

deltaP: Value of delta P (polarization linewidth which is a parameter that depends on optical fibre installation) used to perform the simulation.

filename: Input filename which contains the stokes parameters resulted from simulation (*.txt).

NumberOfSamples: Number of Samples to plot in Poincare sphere.

Outputs

Two graphics are plotted from the photon stream input signal:

Histogram of parameters $\vec{\alpha}$: Three plots of each α_i which were used as input parameters of the random matrix used to model the SOP drift block.

Poincare sphere: A plot of the time evolution of the photon stream into the Poincare sphere.

Functional Description

This matlab function converts the input signal (*.sgn) which is represented in Jones space in a signal represented in Stokes Space which allows us to plot the signal time evolution in Poincare sphere. Furthermore, the $\vec{\alpha}$ parameters obtained from simulation are also plotted in order to be sure that they were generated correctly by following a normal distribution with mean 0 and a standard deviation depending on these parameters values.

Students Name	:	Mariana Ramos
Version	:	ACF_20180614.m

8.2.2 ACF.m

Structure of a function

[] = ACF(fname,deltaP,N)

Inputs

fname: Input filename of the signal (*.sgn) you want to convert. It must be a photon stream signal (Type: PhotonStreamXY).

deltaP: Value of delta P (polarization linewidth which is a parameter that depends on optical fibre installation) used to perform the simulation.

N: Number of Samples used to plot the function of autocorrelation.

Outputs

This matlab function has two outputs:

.txt file with values of the autocorrelation calculated with data from simulation: A .txt file with the autocorrelation of the photon stream signal which inputs the function for N samples which is also an input of the function-.

ACF plot: A plot with numerical ACF calculated from simulation data and with the theoretical ACF calculated based on the value of Δp inserted as an input of the function.

Functional Description

This matlab function calculates the autocorrelation in time domain of the photon stream input signal as well as the theoretical autocorrelation based on the value of Δp inserted as an input of the function which must be the same used in simulation. This function outputs a txt file with the numerical ACF and plots a graphic with both theoretical and numerical autocorrelation.

Students Name	:	Mariana Ramos
Version	:	plotPhotonStream_20180102.m

8.2.3 plotPhotonStream_20180102.m

Structure of a function

[mag_x, mag_y, phase_dif, h] = plotPhotonStream_20180102(fname, opt, h)

Inputs

fname: Input filename of the signal (*.sgn) you want to convert. It must be a photon stream signal (Type: PhotonStreamXY).

opt: If opt equals 0, or no opt, we plot the absolute value of X and Y and the phase difference.If opt equals 1, we plot the amplitude of X and Y, this is only possible when X and Y are real values.

h: Number of the figure to plot.

Outputs

This matlab function has four outputs:

mag_x: Magnitude of the complex number X.

mag_y: Magnitude of the complex number Y.

phase_dif: Difference phase between the two complex numbers X and Y.

h: A plot of magnitude and phase difference of the two complex numbers depending on the choice of the input parameter **opt**.

Functional Description

This matlab function plots the magnitude of the two components of the input photon stream signal (*.sgn) and the phase difference between both components. This function allows us to visualize the photon stream signal over time.

Students Name	:	Mariana Ramos
Version	:	plot_sphere.m

8.2.4 plot_sphere.m

This function accepts the three stokes parameters S_1, S_2 and S_3. This function supports the other functions in this section allowing the plot of these parameters in Poincare sphere.

Chapter 9

Algorithms

9.1 Fast Fourier Transform

Header File	:	fft_*.h
Source File	:	fft_*.cpp
Version	:	20180201 (Romil Patel)

Algorithm

The algorithm for the FFT will be implemented according with the following expression,

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \ e^{i2\pi kn/N} \qquad 0 \leq k \leq N-1 \tag{9.1}$$

Similarly, for IFFT,

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k \ e^{-i2\pi kn/N} \qquad 0 \leq k \leq N-1 \tag{9.2}$$

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,

$$y = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x \ e^{m\ i2\pi kn/N} \qquad 0 \leq k \leq N-1 \tag{9.3}$$

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 $O(Nlog_2N)$, 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].

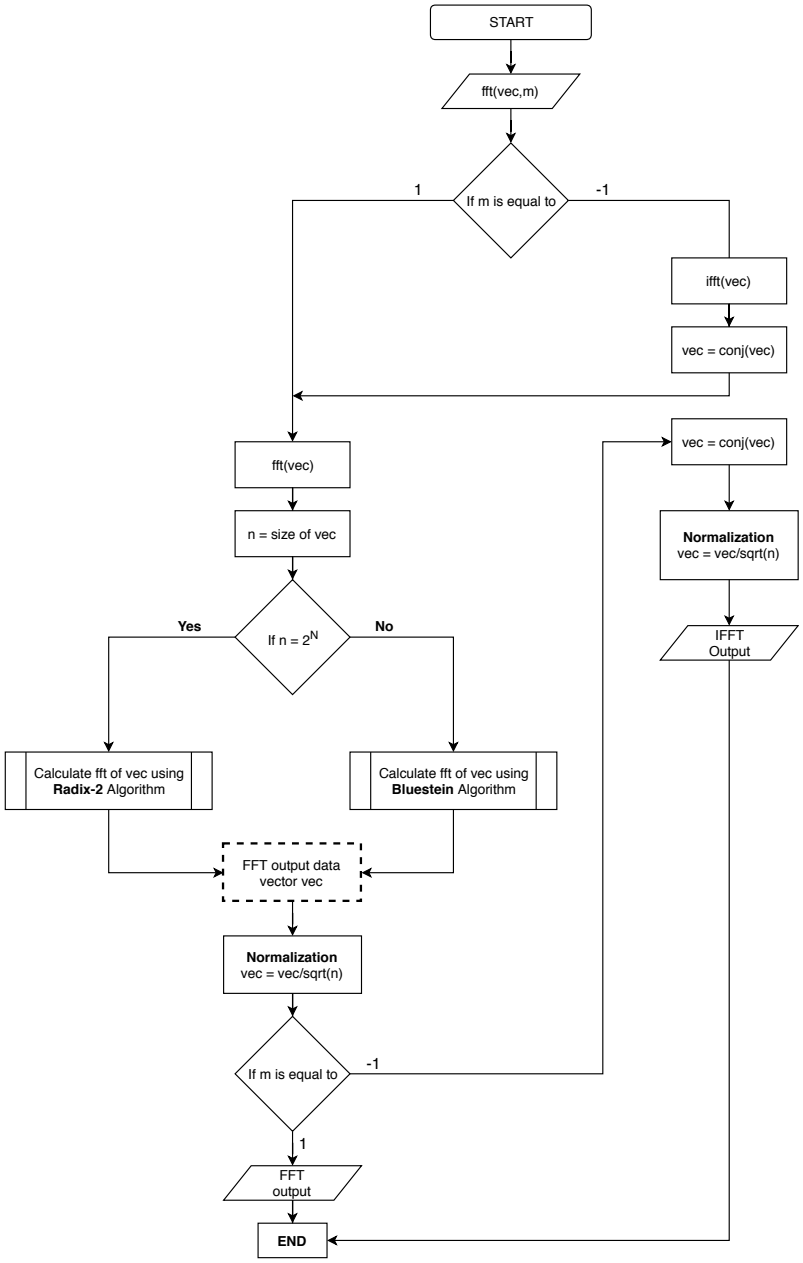

Figure 9.1: Top level architecture of FFT algorithm

Radix-2 algorithm

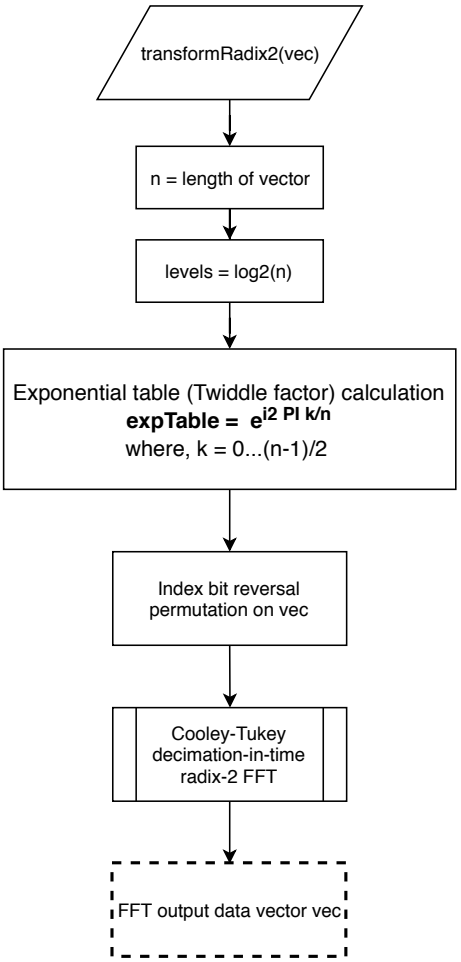

Figure 9.2: Radix-2 algorithm

Cooley-Tukey algorithm

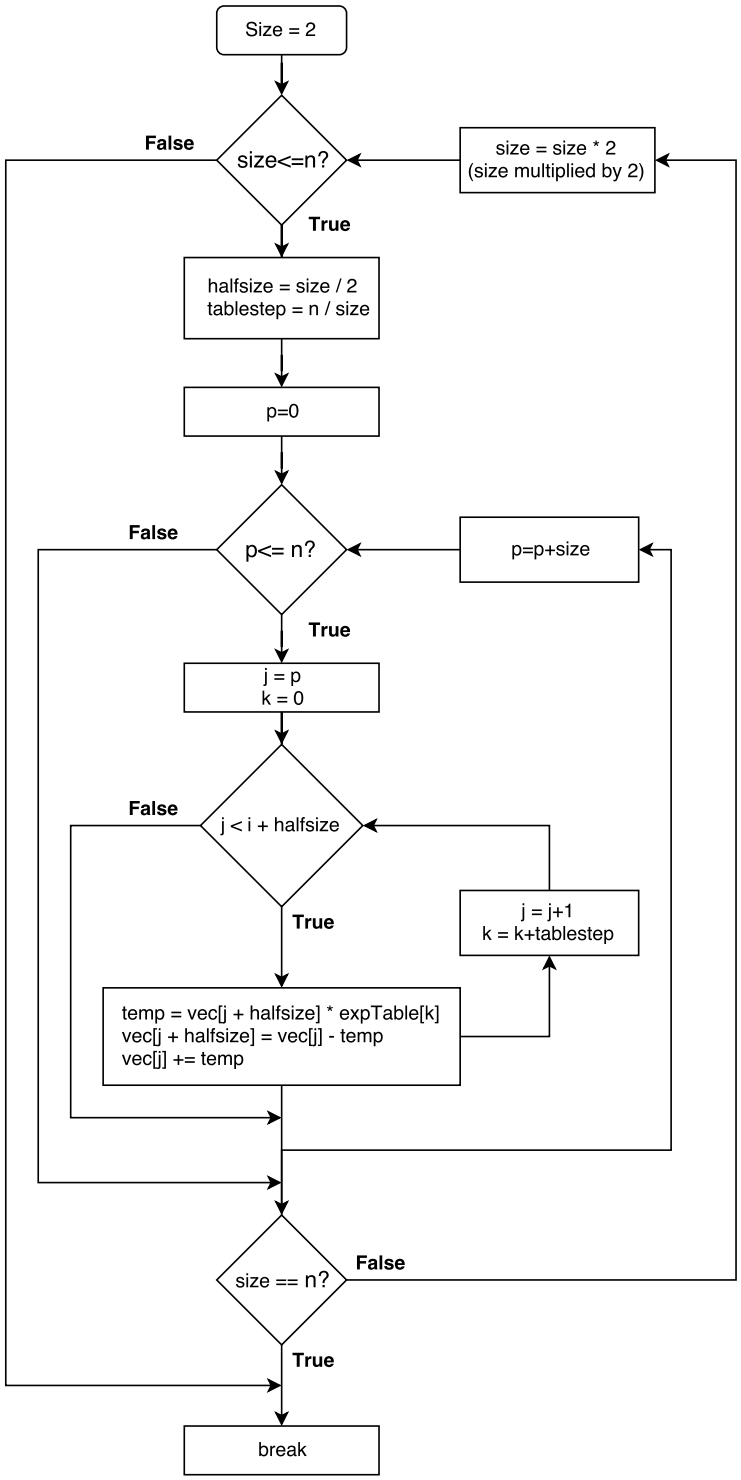

Figure 9.3: Cooley-Tukey algorithm

Bluestein algorithm

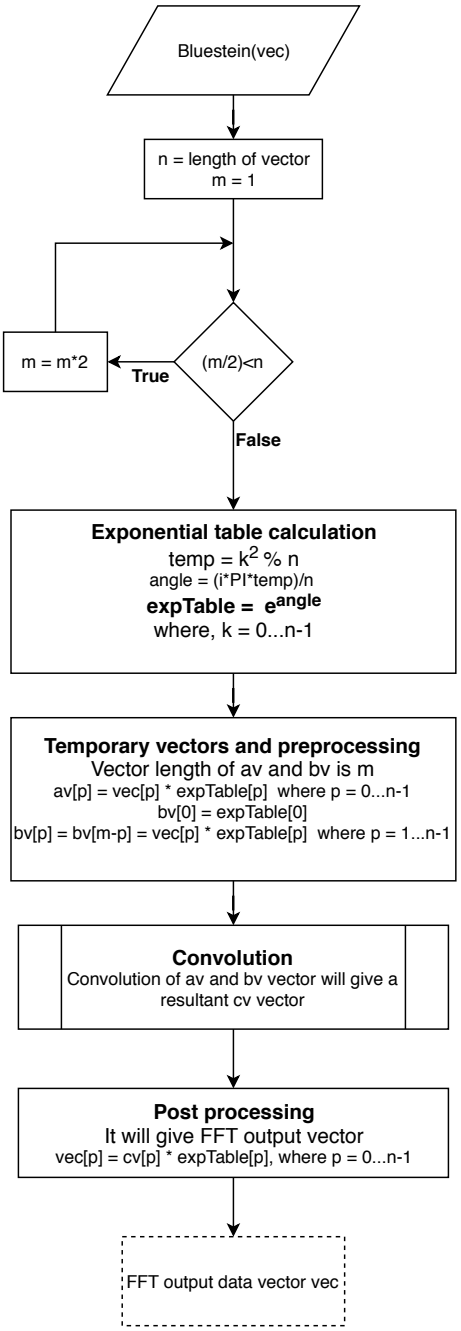

Figure 9.4: Bluestein algorithm

Convolution algorithm

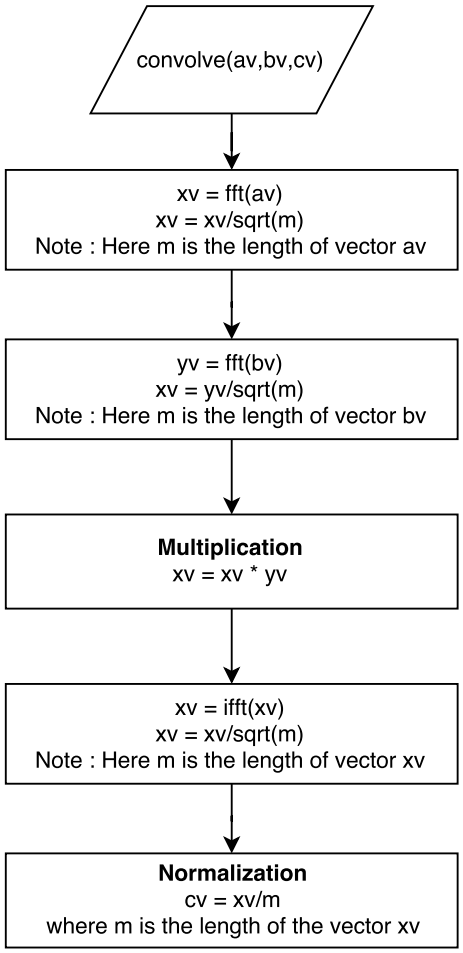

Figure 9.5: Circular convolution algorithm

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.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all
close all

Fs = 1e5;               % Sampling frequency
T = 1/Fs;               % Sampling period
L = 2^10;               % Length of signal
t = (0:L-1)*(5*T);      % Time vector
f = linspace(-Fs/2,Fs/2,L);

%Choose for sig a value between [1, 7]
sig = 2;
switch sig
    case 1
        signal_title = 'Signal with one signusoid and random noise';
        S = 0.7*sin(2*pi*50*t);
        X = S + 2*randn(size(t));
    case 2
        signal_title = 'Sinusoids with Random Noise';
        S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
        X = S + 2*randn(size(t));
    case 3
        signal_title = 'Single sinusoids';
        X = sin(2*pi*t);
    case 4
        signal_title = 'Summation of two sinusoids';
        X = sin(2*pi*t) + cos(2*pi*t);
    case 5
        signal_title = 'Single Sinusoids with Exponent';
```

```
        X = sin(2*pi*200*t).*exp(-abs(70*t));
    case 6
        signal_title = 'Mixed signal 1';
        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);
    case 7
        signal_title = 'Mixed signal 2';
        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);
    case 8
        signal_title = 'Sinusoid tone';
        X = cos(2*pi*100*t);
end

plot(t(1:end),X(1:end))
title(signal_title)
axis([min(t) max(t) 1.1*min(X) 1.1*max(X)]);
xlabel('t (s)')
ylabel('X(t)')
grid on

% dlmwrite will generate text file which represents the time domain signal.
% dlmwrite('time_function.txt', X, 'delimiter','\t');
fid=fopen('time_function.txt','w');
b=fprintf(fid,'%0.15f\n',X); % 15-Digit accuracy
fclose(fid);

tic
fy =fft(X);
toc
fy = fftshift(fy);
figure(2);
subplot(2,1,1)
plot(f,abs(fy));
axis([-Fs/(2*5) Fs/(2*5) 0 1.1*max(abs(fy))]);
xlabel('f');
ylabel('|Y(f)|');
title('MATLAB program Calculation : Magnitude');
grid on
subplot(2,1,2)
plot(f,phase(fy));
xlim([-Fs/(2*5) Fs/(2*5)]);
xlabel('f');
ylabel('phase(Y(f))');
title('MATLAB program Calculation : Phase');
grid on

%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Read C++ transformed data file
fullData = load('frequency_function.txt');
A=1;
B=A+1;
l=1;
Z=zeros(length(fullData)/2,1);
while (l<=length(Z))
Z(l) = fullData(A)+fullData(B)*1i;
A = A+2;
B = B+2;
l=l+1;
end

% % Comparsion of the MATLAB and C++ fft calculation.
figure;
subplot(2,1,1)
plot(f,abs( fftshift ( fft (X))))
hold on
%Multiplied by sqrt(n) to verify our C++ code with MATLAB implemenrtation.
%plot(f,(sqrt(length(Z))*abs( fftshift (Z))), '--o')
plot(f,abs( fftshift (Z)), '--o') % fftOptimized
axis([-Fs/(2*5) Fs/(2*5) 0 1.1*max(abs(fy))]);
xlabel('f (Hz)');
 title ('Main reference for Magnitude')
legend('MATLAB','C++')
grid on
subplot(2,1,2)
plot(f,phase( fftshift ( fft (X))))
hold on
plot(f,phase( fftshift (Z)),'--o')
xlim([-Fs/(2*5) Fs/(2*5)])
 title ('Main reference for Phase')
xlabel('f (Hz)');
legend('MATLAB','C++')
grid on

%
% % IFFT test comparision Plot
% figure; plot(X); hold on; plot(real(Z),'--o');
```

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.

```
# include "fft_20180208.h"

# include <complex>
# include <fstream>
# include <iostream>
# include <math.h>
# include <stdio.h>
# include <string>
# include <strstream>
# include <algorithm>
# include <vector>
#include <iomanip>

using namespace std;

int main()
{
  ////////////////////////////////// Section 1 ////////////////////////////////////
  /////////// Read the input text file (import "time_function.txt" ) ///////////
  ////////////////////////////////////////////////////////////////////////////////
  ifstream inFile;
  inFile.precision(15);
  double ch;
  vector <double> inTimeDomain;
  inFile.open("time_function.txt");

  // First data (at 0th position) appled to the ch it is similar to the "cin".
  inFile >> ch;

  // It'll count the length of the vector to verify with the MATLAB
  int count=0;

  while (!inFile.eof()){
    // push data one by one into the vector
    inTimeDomain.push_back(ch);

    // it'll increase the position of the data vector by 1 and read full vector.
    inFile >> ch;

    count++;
  }

  inFile.close(); // It is mandatory to close the file at the end.
```

```
44
    ////////////////////////////////// Section 2 ////////////////////////////////////
46  ////////////////////////////////// Calculate FFT ////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
48
    vector <complex<double>> inTimeDomainComplex(inTimeDomain.size());
50  vector <complex<double>> fourierTransformed;
    vector <double> re(inTimeDomain.size());
52  vector <double> im(inTimeDomain.size());

54  for (unsigned int i = 0; i < inTimeDomain.size(); i++)
    {
56    re[i] = inTimeDomain[i]; // Real data of the signal
    }
58
    // Next, Real and Imaginary vector to complex vector conversion
60  inTimeDomainComplex = reImVect2ComplexVector(re, im);

62  // calculate FFT
    clock_t begin = clock();
64  fourierTransformed = fft(inTimeDomainComplex,-1,1); // Optimized
    clock_t end = clock();
66  double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;

68  ////////////////////////////////// Section 3 ////////////////////////////////////
    //////// Save FFT calculated data (export "frequency_function.txt" ) /////////
70  ////////////////////////////////////////////////////////////////////////////////
    ofstream outFile;
72  complex<double> outFileData;
    outFile.open("frequency_function.txt");
74  outFile.precision(15);
    for (unsigned int i = 0; i <fourierTransformed.size(); i++){
76    outFile << fourierTransformed[i].real() << endl;
      outFile << fourierTransformed[i].imag() << endl;
78  }
    outFile.close();
80
    ////////////////////////////////// Section 4 ////////////////////////////////////
82  ////////////////////////////////// Display Section //////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
84  for (unsigned int i = 0; i < fourierTransformed.size(); i++){
      cout << fourierTransformed[i] << endl; // Display all FFT calculated data
86  }
    cout << "\n\nTime elapsed to calculate FFT : " << elapsed_secs << " seconds"<< endl;
88  cout <<"\nTotal length of of data :"<< count << endl;
    getchar();
90  return 0;
}
```

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

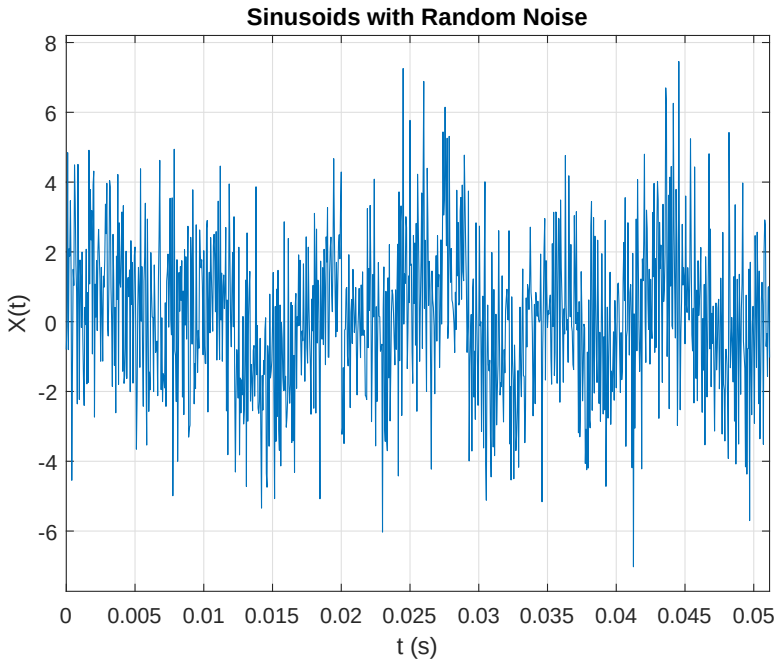

Figure 9.6: Random noise and two sinusoids

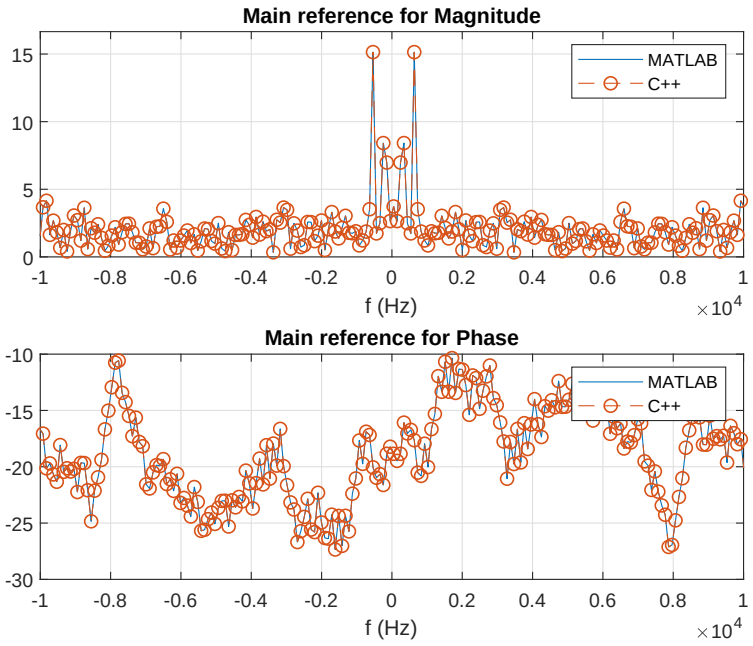

Figure 9.7: MATLAB and C++ comparison

2. Sinusoid with an exponent

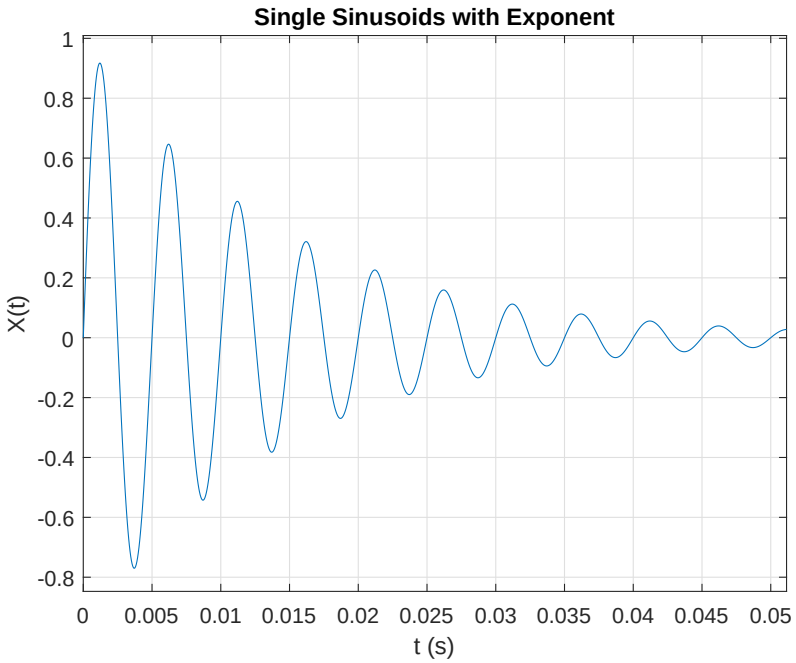

Figure 9.8: Sinusoids with exponent

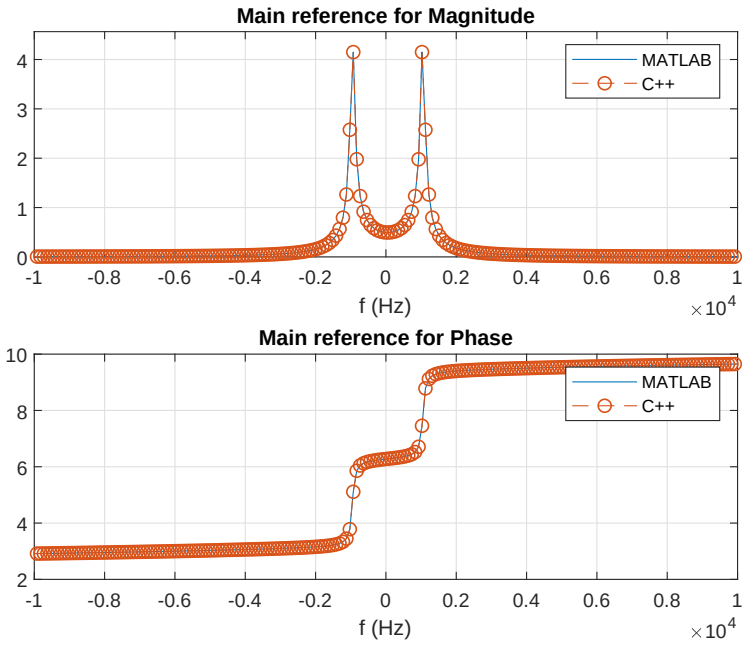

Figure 9.9: MATLAB and C++ comparison

3. Mixed signal

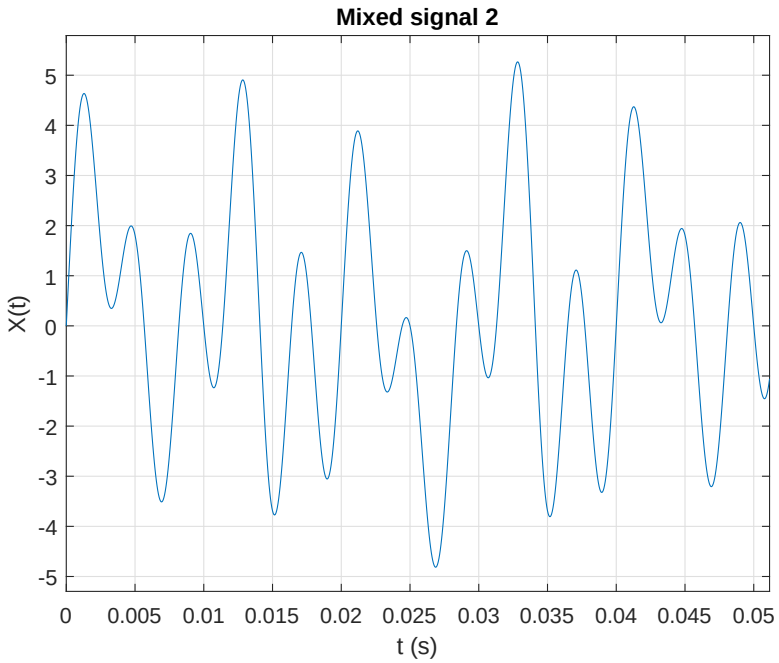

Figure 9.10: mixed signal

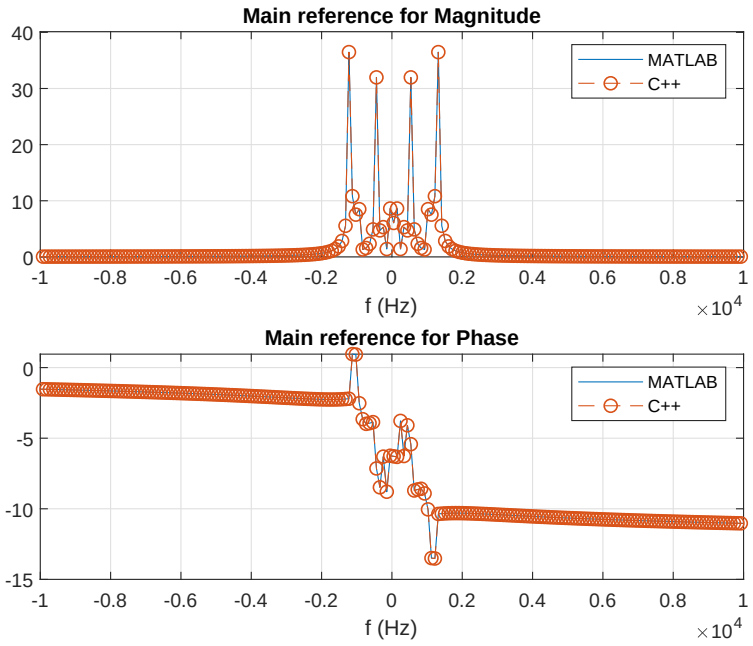

Figure 9.11: MATLAB and C++ comparison

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,

$$X_k = \sum_{n=0}^{N-1} x_n \ e^{m\ i2\pi kn/N} \qquad 0 \leq k \leq N-1 \tag{9.4}$$

Similarly, for IFFT,

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \ e^{m\ i2\pi kn/N} \qquad 0 \leq k \leq N-1 \tag{9.5}$$

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

Header File	:	overlap_save_*.h
Source File	:	overlap_save_*.cpp
Version	:	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,

$$y_i(n) = x_i(n) \circledast h(n). \tag{9.6}$$

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.

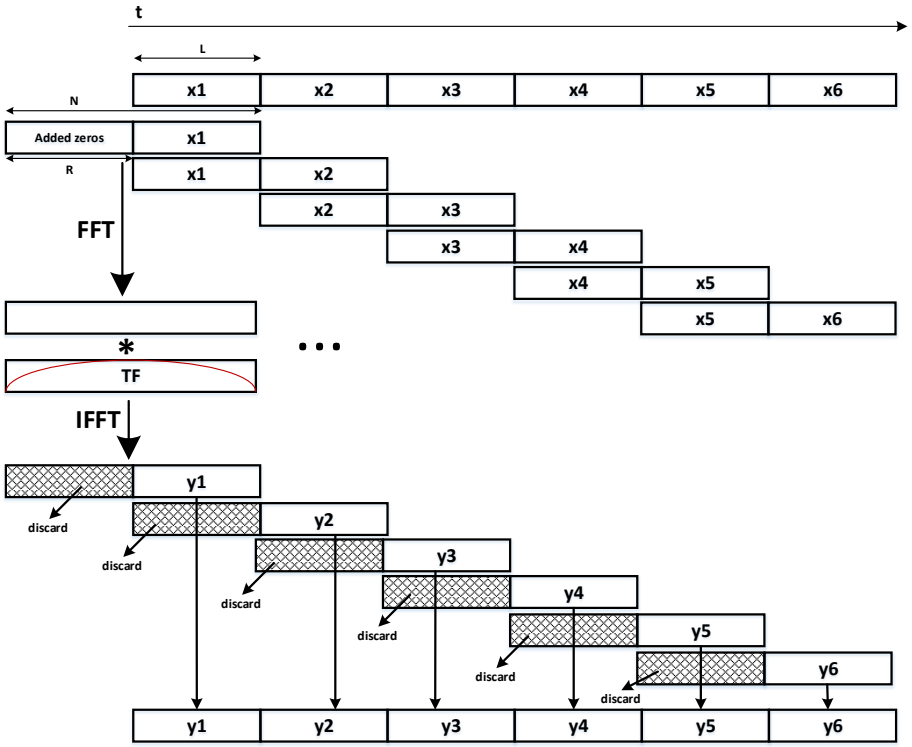

Figure 9.12: Illustration of Overlap-save method.

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,

$$y(n) = overlapSave(x(n), h(n))$$

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,

$$y(n) = overlapSave(x_m(n), x_{m-1}(n), h(n))$$

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 singal $x(n)$ 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

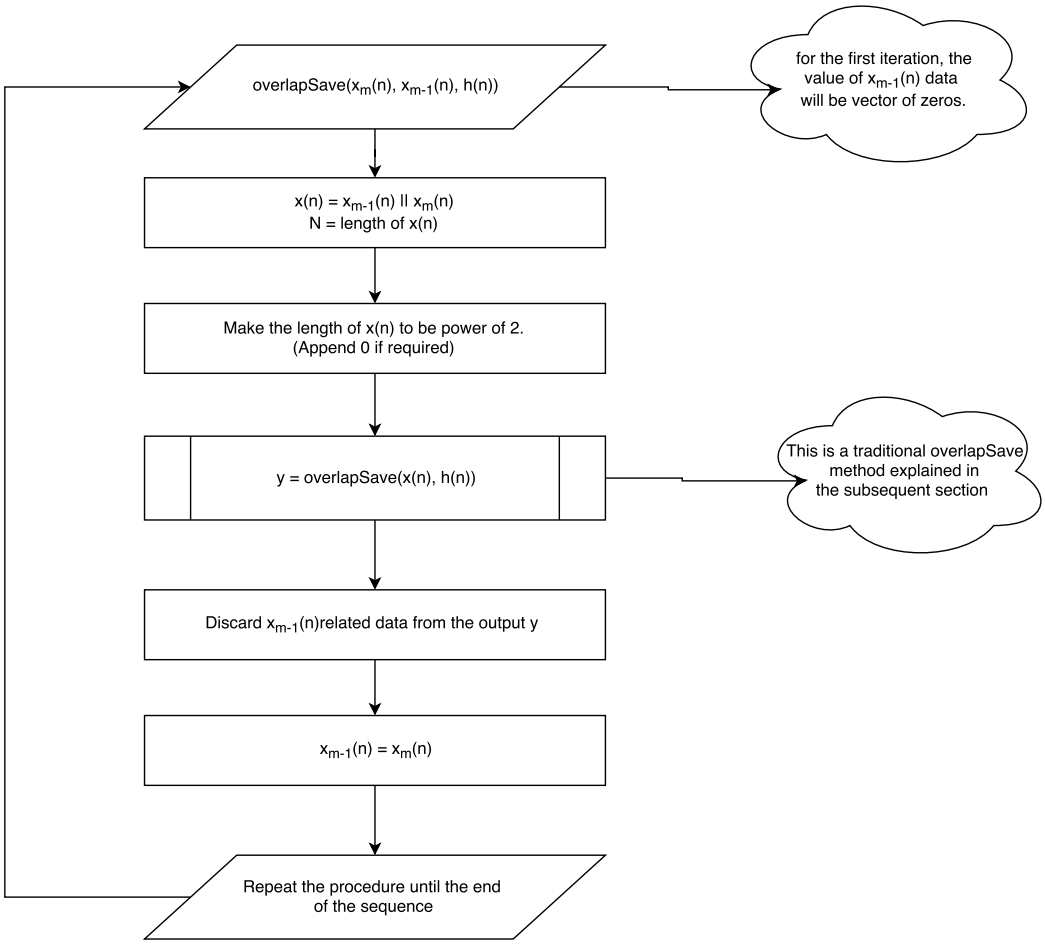

Figure 9.13: Flowchart for real-time overlap-save method

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

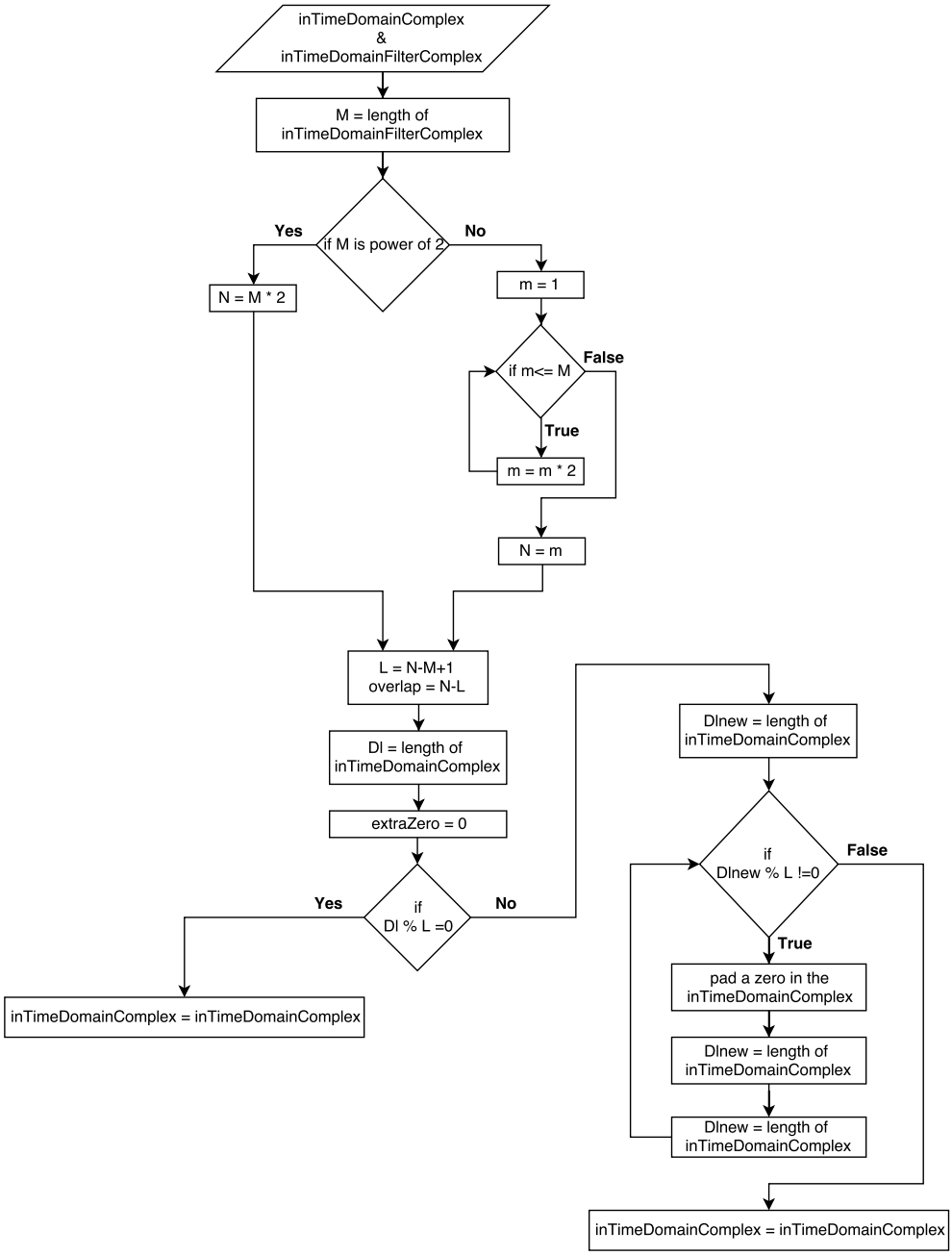

Figure 9.14: Flowchart for calculating length of FFT, data block and filter

2. Create matrix with overlap

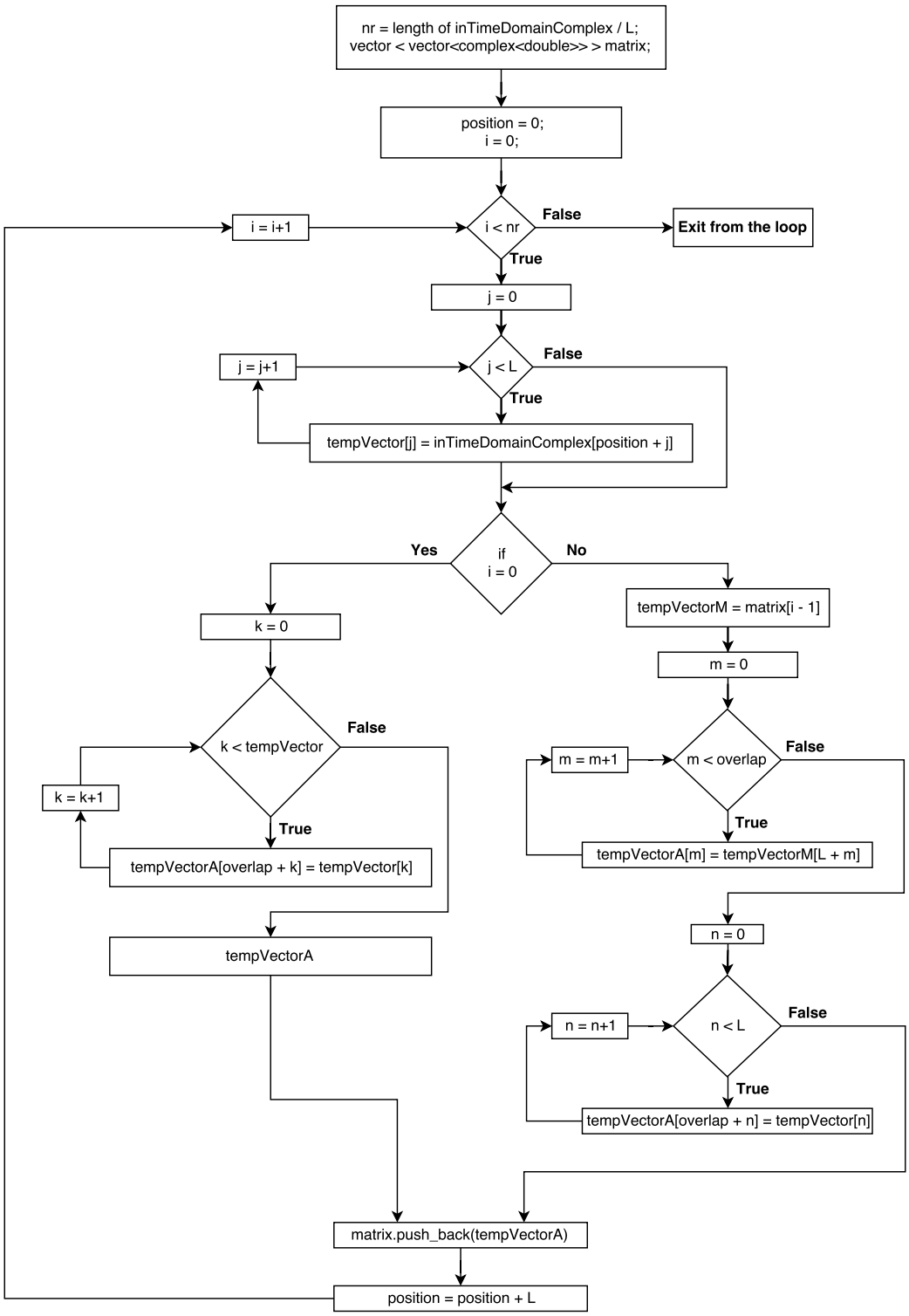

Figure 9.15: Flowchart of creating matrix with overlap

3. Convolution between filter and data blocks

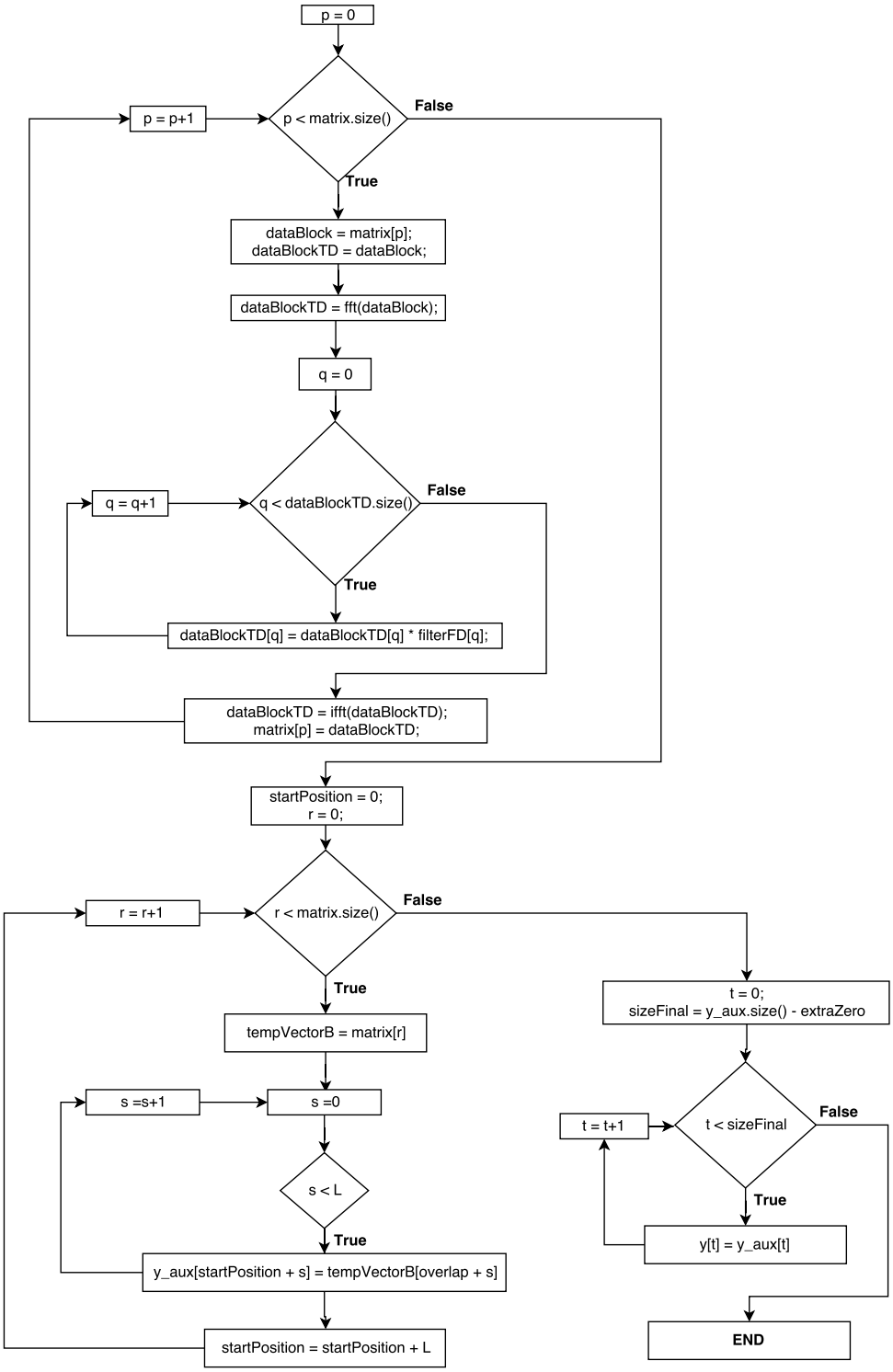

Figure 9.16: Flowchart of the convolution

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().

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generate signal and filter data and save it as a .txt file .
clc
clear all
close all

Fs = 1e5;                % Sampling frequency
T = 1/Fs;                % Sampling period
L = 2^10;                 % Length of signal
t = (0:L-1)*(5*T);       % Time vector
f = linspace(-Fs/2,Fs/2,L);

%Choose for sig a value between [1, 7]
sig = 5;
switch sig
    case 1
        signal_title = 'Signal with one signusoid and random noise';
       S = 0.7*sin(2*pi*50*t);
```

```
21          X = S + 2*randn(size(t));
    case 2
23          signal_title  = 'Sinusoids with Random Noise';
            S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
25          X = S + 2*randn(size(t));
    case 3
27          signal_title  = 'Single sinusoids';
            X = sin(2*pi*t);
29  case 4
            signal_title  = 'Summation of two sinusoids';
31          X = sin(2*pi*1205*t) + cos(2*pi*1750*t);
    case 5
33          signal_title  = 'Single Sinusoids with Exponent';
            X = sin(2*pi*250*t).*exp(-70*abs(t));
35  case 6
            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);
    case 7
39          signal_title  = 'Mixed signal 2';
            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  end

43  Xref = X;
    % dlmwrite will generate text file which represents the time domain signal.
45  %dlmwrite('time_domain_data.txt', X, 'delimiter','\t');
    fid=fopen('time_domain_data.txt','w');
47  fprintf(fid,'%.20f\n',X); % 12-Digit accuracy
    fclose(fid);
49

51  % Choose for filt a value between [1, 3]
     filt = 1;
53  switch filt
    case 1
55          filter_type  = 'Impulse response of rcos filter';
            h = rcosdesign(0.25,11,6);
57  case 2
            filter_type  = 'Impulse response of rrcos filter';
59          h = rcosdesign(0.25,11,6,'sqrt');
    case 3
61          filter_type  = 'Impulse response of Gaussian filter';
            h = gaussdesign(0.25,11,6);
63  end

65  %dlmwrite('time_domain_filter.txt', h, 'delimiter','\t');
    fid=fopen('time_domain_filter.txt','w');
67  fprintf(fid,'%.20f\n',h); % 20-Digit accuracy
    fclose(fid);
69

71  figure;
```

```
subplot(211)
plot(t,X)
grid on
 title ( signal_title )
axis([min(t) max(t) 1.1*min(X) 1.1*max(X)]);
xlabel('t (s)')
ylabel('X(t)')

subplot(212)
plot(h)
grid on
 title ( filter_type )
axis([1 length(h) 1.1*min(h) 1.1*max(h)]);
xlabel('Samples')
ylabel('h(t)')

%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculate the length of FFT, data blocks and filter
M = length(h);

 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

L = N -M+1;    % Size of data block (50% of overlap)
overlap = N - L;  % size of overlap
Dl = length(X);
extraZeros = 0;
 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
```

```
%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATLAB approach of overlap-save method (First create matrix with
% overlap and then perform convoluion)
zerosForFilter = zeros(1,N-M);
h1=[h zerosForFilter];
H1 = fft(h1);

x1=X;
nr=ceil((length(x1))/L);

tic
for k=1:nr
    Ma(k,:)=x1(((k-1)*L+1):k*L);
    if k==1
        Ma1(k,:)=[zeros(1,overlap) Ma(k,:)];
    else
        tempVectorM = Ma1(k-1,:);
        overlapData = tempVectorM(L+1:end);
        Ma1(k,:)=[overlapData Ma(k,:)];
    end
    auxfft = fft(Ma1(k,:));
    auxMult = auxfft.*H1;
    Ma2(k,:)=ifft(auxMult);
end

Ma3=Ma2(:,N-L+1:end);
y1=Ma3';
y=y1(:)';
y = y(1:end - extraZeros);
toc
%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read overlap-save data file generated by C++ program and compare with
fullData = load('overlap_save_data.txt');
A=1;
B=A+1;
l=1;
```

```
Z=zeros(length(fullData)/2,1);
while (l<=length(Z))
Z(l) = fullData(A)+fullData(B)*1i;
A = A+2;
B = B+2;
l=l+1;
end

figure;
plot(t,real(y))
hold on
plot(t,real(Z),'o')
axis([min(t) max(t) 1.1*min(y) 1.1*max(y)]);
xlabel('t (Seconds)')
ylabel('y(t)')
 title ('Comparision of overlapSave method of MATLAB and C++ ')
legend('MATLAB overlapSave','C++ overlapSave')
grid on
%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Our MATLAB and C++ implementation test with the built-in conv function of
% MATLAB.
 tic
P = conv(Xref,h);
toc
figure
plot(t, P(1:size(Z,1)),'r')
hold on
plot(t,real(Z),'o')
 title ('Comparision of MATLAB function conv() and overlaSave')
axis([min(t) max(t) 1.1*min(real(Z)) 1.1*max(real(Z))]);
xlabel('t (Seconds)')
ylabel('y(t)')
legend('MATLAB function : conv(X,h)','C++ overlapSave')
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*.

```
# include "overlap_save_20180208.h"

# include <complex>
# include <fstream>
# include <iostream>
# include <math.h>
# include <stdio.h>
# include <string>
# include <strstream>
# include <algorithm>
# include <vector>

using namespace std;

int main()
{
  ////////////////////////////////// Section 1 ////////////////////////////////////
  ////// Read the time_domain_data.txt and time_domain_filter.txt files ///////
  ////////////////////////////////////////////////////////////////////////////////
  ifstream inFile;
  double ch;
  vector <double> inTimeDomain;
  inFile.open("time_domain_data.txt");

  // First data (at 0th position) appled to the ch it is similar to the "cin".
  inFile >> ch;

  // It'll count the length of the vector to verify with the MATLAB
  int count = 0;

  while (!inFile.eof())
  {
    // push data one by one into the vector
    inTimeDomain.push_back(ch);

    // it'll increase the position of the data vector by 1 and read full vector.s
    inFile >> ch;
    count++;
  }

  inFile.close(); // It is mandatory to close the file at the end.
```

```
43  ifstream inFileFilter;
    double chFilter;
45  vector <double> inTimeDomainFilter;
    inFileFilter.open("time_domain_filter.txt");
47  inFileFilter >> chFilter;
    int countFilter = 0;
49
    while (!inFileFilter.eof())
51  {
      inTimeDomainFilter.push_back(chFilter);
53    inFileFilter >> chFilter;
      countFilter++;
55  }
    inFileFilter.close();
57
    ////////////////////////////////// Section 2 ////////////////////////////////////
59  ////////////////////////// Real to complex conversion ///////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
61  ////////////// For signal data //////////////
    vector <complex<double>> inTimeDomainComplex(inTimeDomain.size());
63  vector <complex<double>> fourierTransformed;
    vector <double> re(inTimeDomain.size());
65  vector <double> im(inTimeDomain.size());

67  for (unsigned int i = 0; i < inTimeDomain.size(); i++)
    {
69    // Real data of the signal
      re[i] = inTimeDomain[i];
71
      // Imaginary data of the signal
73    im[i] = 0;
    }
75  // Next, Real and Imaginary vector to complex vector conversion
    inTimeDomainComplex = reImVect2ComplexVector(re, im);
77
    ////////////// For filter data //////////////
79  vector <complex<double>> inTimeDomainFilterComplex(inTimeDomainFilter.size());
    vector <double> reFilter(inTimeDomainFilter.size());
81  vector <double> imFilter(inTimeDomainFilter.size());

83  for (unsigned int i = 0; i < inTimeDomainFilter.size(); i++)
    {
85    reFilter[i] = inTimeDomainFilter[i];
      imFilter[i] = 0;
87  }
    inTimeDomainFilterComplex = reImVect2ComplexVector(reFilter, imFilter);
89
    ////////////////////////////////// Section 3 ////////////////////////////////////
91  ////////////////////////////// Overlap & save ///////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
93  vector <complex<double>> y;
    y = overlapSave(inTimeDomainComplex, inTimeDomainFilterComplex);
```

```
95
    ////////////////////////////////// Section 4 ////////////////////////////////////
97  ////////////////////////////////// Save data ////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
99  ofstream outFile;
    complex<double> outFileData;
101 outFile.precision(20);
    outFile.open("overlap_save_data.txt");
103
    for (unsigned int i = 0; i <y.size(); i++)
105 {
      outFile << y[i].real() << endl;
107   outFile << y[i].imag() << endl;
    }
109 outFile.close();

111 cout << "Execution finished! Please hit enter to exit." << endl;
    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

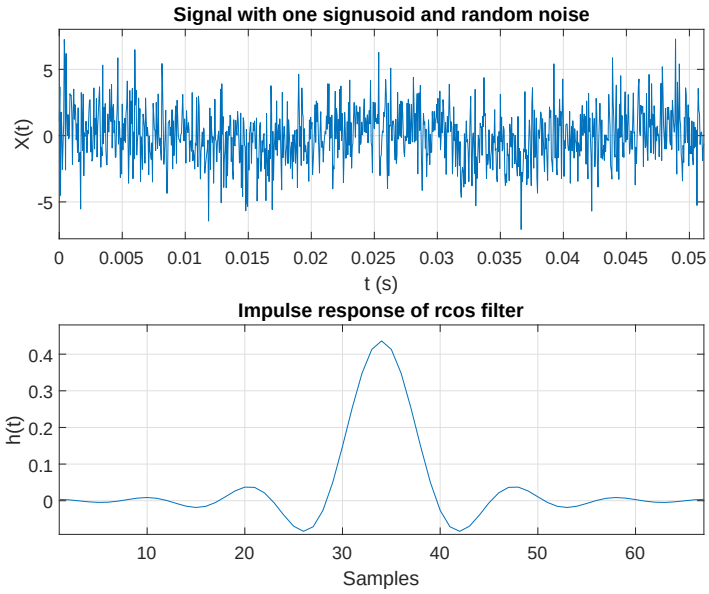

Figure 9.17: Random noise and two sinusoids signal & Impulse response of rcos filter

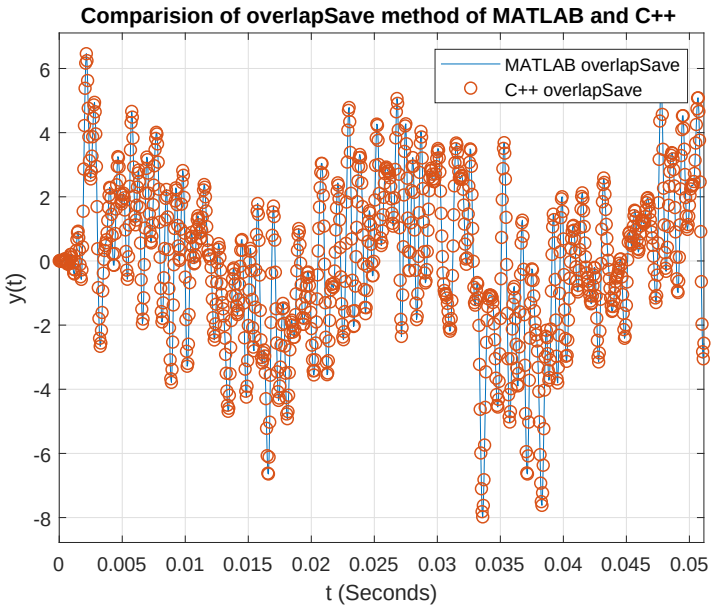

Figure 9.18: MATLAB and C++ comparison

3. Sinusoid with exponent

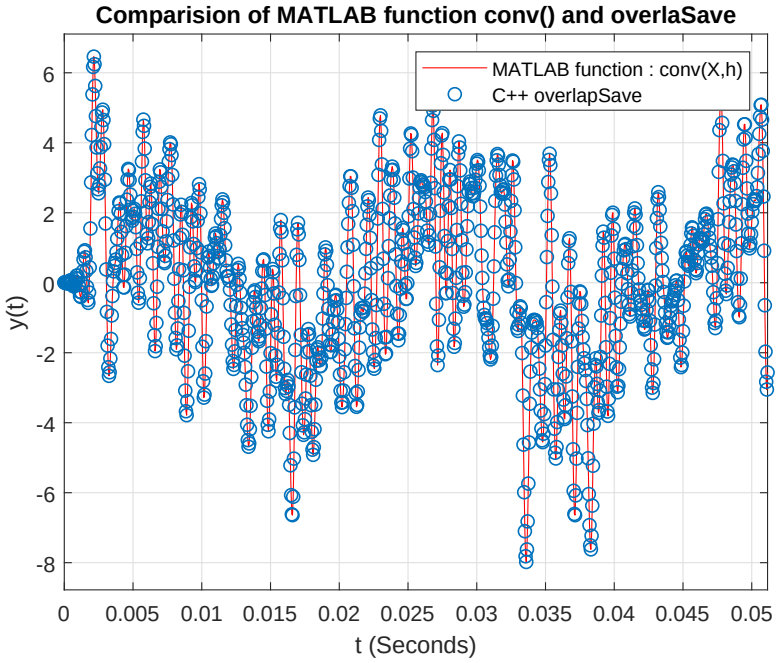

Figure 9.19: MATLAB function conv() and C++ overlapSave comparison

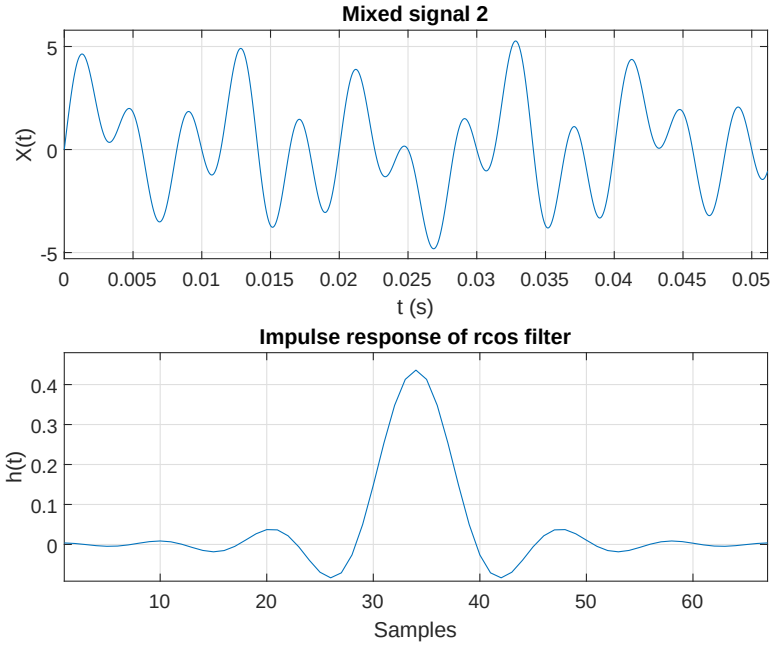

Figure 9.20: Mixed signal2 & Impulse response of rcos filter

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

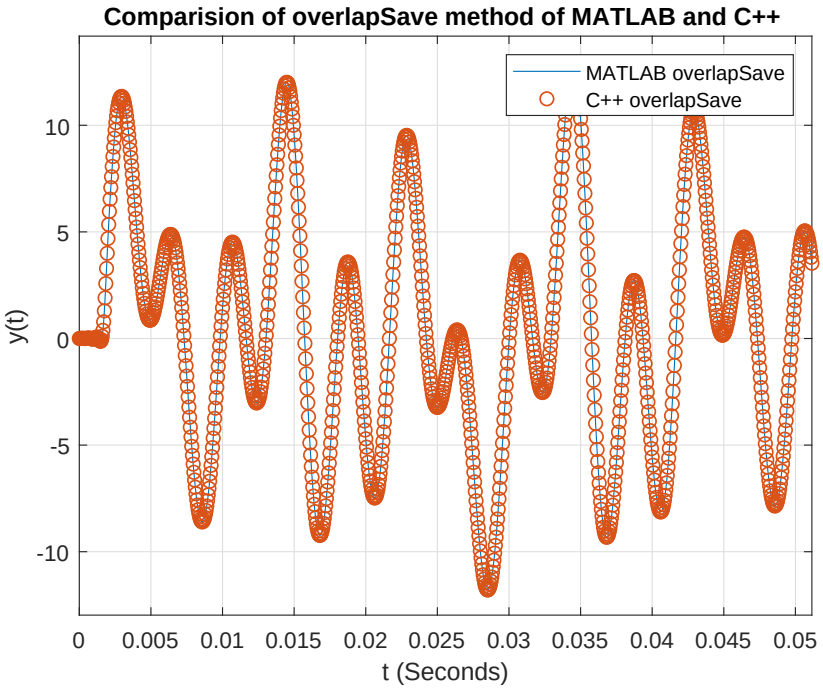

Figure 9.21: MATLAB and C++ comparison

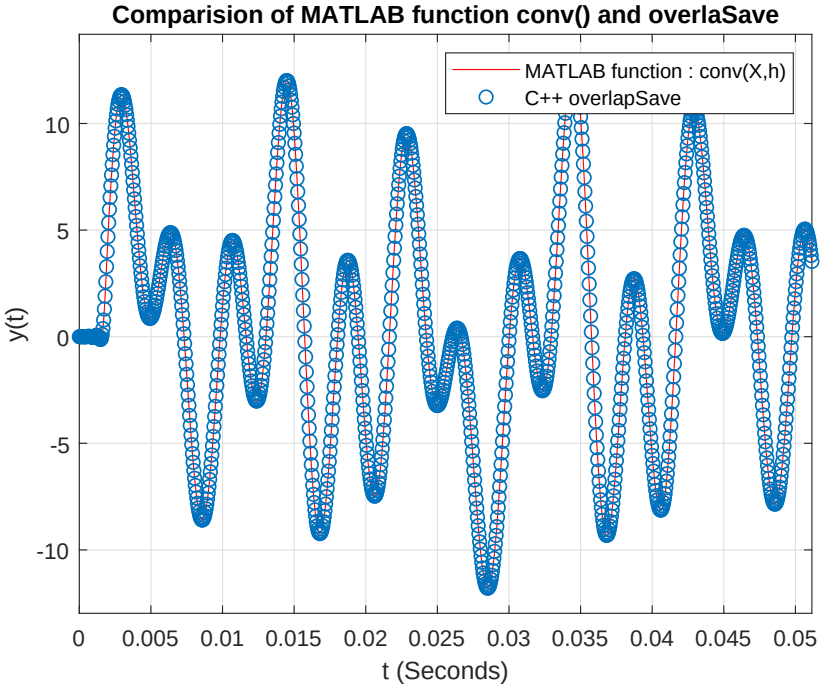

Figure 9.22: MATLAB function conv() and C++ overlapSave comparison

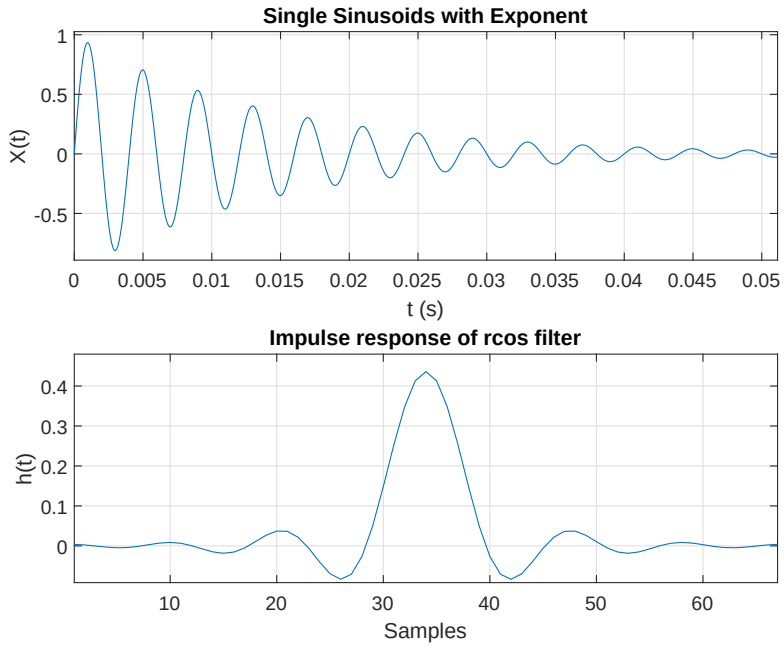

Figure 9.23: Sinusoid with exponent & Impulse response of Gaussian filter

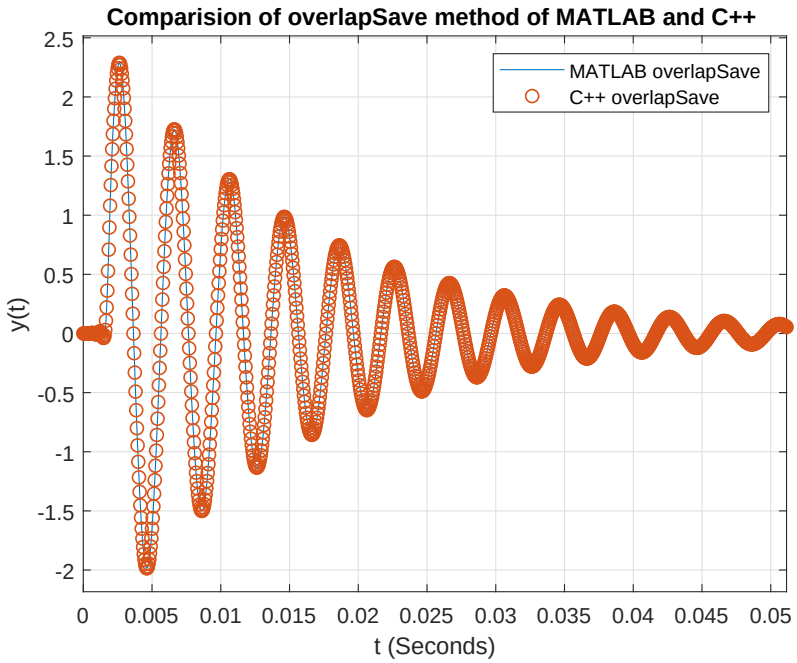

Figure 9.24: MATLAB and C++ comparison

$overlapSave(x_{m-1}(n), x_m(n), h(n))$ 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

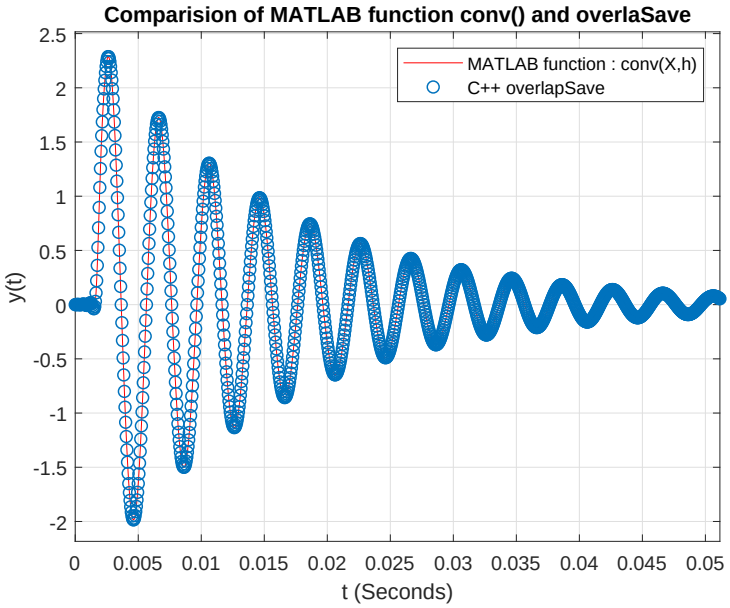

Figure 9.25: MATLAB function conv() and C++ overlapSave comparison

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.

```
# include "filter_20180306.h"

////////////////////////////////////////////////////////////
////////////////////////// FIR_Filter ////////////////////////// TIME DOMAIN
////////////////////////////////////////////////////////////

void FIR_Filter :: initializeFIR_Filter (void) {

  outputSignals[0]->symbolPeriod = inputSignals[0]->symbolPeriod;
  outputSignals[0]->samplingPeriod = inputSignals[0]->samplingPeriod;
```

```
  outputSignals[0]->samplesPerSymbol = inputSignals[0]->samplesPerSymbol;

  if (!getSeeBeginningOfImpulseResponse()) {
    int aux = (int) (((double)impulseResponseLength) / 2) + 1;
    outputSignals[0]->setFirstValueToBeSaved(aux);
  }

  delayLine.resize(impulseResponseLength, 0);

  if (saveImpulseResponse) {
    ofstream fileHandler("./signals/" + impulseResponseFilename, ios::out);
    fileHandler << "// ### HEADER TERMINATOR ###\n";

    t_real t;
    double samplingPeriod = inputSignals[0]->samplingPeriod;
    for (int i = 0; i < impulseResponseLength; i++) {
      t = -impulseResponseLength / 2 * samplingPeriod + i * samplingPeriod;
      fileHandler << t << " " << impulseResponse[i] << "\n";
    }
    fileHandler.close();
  }

};


bool FIR_Filter::runBlock(void) {

  int ready = inputSignals[0]->ready();
  int space = outputSignals[0]->space();
  int process = min(ready, space);
  if (process == 0) return false;

  for (int i = 0; i < process; i++) {
    t_real val;
    (inputSignals[0])->bufferGet(&val);
    if (val != 0) {
      vector<t_real> aux(impulseResponseLength, 0.0);
      transform(impulseResponse.begin(), impulseResponse.end(), aux.begin(), bind1st(multiplies<t_real>(),
     val));
      transform(aux.begin(), aux.end(), delayLine.begin(), delayLine.begin(), plus<t_real>());
    }
    outputSignals[0]->bufferPut((t_real)(delayLine[0]));
    rotate(delayLine.begin(), delayLine.begin() + 1, delayLine.end());
    delayLine[impulseResponseLength - 1] = 0.0;
  }

  return true;
};


////////////////////////////////////////////////////////////
////////////////////////// FD_Filter ////////////////////////// FREQUENCY DOMAIN
```

```
62 ////////////////////////////////////////////////////////////
   void FD_Filter:: initializeFD_Filter (void)
64 {
     outputSignals[0]->symbolPeriod = inputSignals[0]->symbolPeriod;
66   outputSignals[0]->samplingPeriod = inputSignals[0]->samplingPeriod;
     outputSignals[0]->samplesPerSymbol = inputSignals[0]->samplesPerSymbol;
68
     if (!getSeeBeginningOfTransferFunction()) {
70     int aux = (int) (((double)transferFunctionLength) / 2) + 1;
       outputSignals[0]->setFirstValueToBeSaved(aux);
72   }

74   if (saveTransferFunction)
     {
76     ofstream fileHandler("./signals/" + transferFunctionFilename, ios::out);
       fileHandler << "// ### HEADER TERMINATOR ###\n";
78
       double samplingPeriod = inputSignals[0]->samplingPeriod;
80     t_real fWindow = 1 / samplingPeriod;
       t_real df = fWindow / transferFunction.size();
82
       t_real f;
84     for (int k = 0; k < transferFunction.size(); k++)
       {
86       f = -fWindow / 2 + k * df;
         fileHandler << f << " " << transferFunction[k] << "\n";
88     }
       fileHandler.close();
90   }
   }
92
   bool FD_Filter::runBlock(void)
94 {
     bool alive{ false };
96
     int ready = inputSignals[0]->ready();
98   int space = outputSignals[0]->space();
     int process = min(ready, space);
100  if (process == 0) return false;

102  ////////////////////// previousCopy & currentCopy ////////////////////
     ////////////////////////////////////////////////////////////////////
104  vector<double> re(process); // Get the Input signal
     t_real input;
106  for (int i = 0; i < process; i++){
       inputSignals[0]->bufferGet(&input);
108    re.at(i) = input;
     }
110
     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
      if  (K == 0){  previousCopy = pcInitialize;  }
116
      // size  modification of  currentCopyAux to currentCopy.
118   vector<t_complex> currentCopy(previousCopy.size());
      for  (unsigned int i  = 0;  i  < currentCopyAux.size(); i++){
120     currentCopy[i] = currentCopyAux[i];
      }
122
      ////////////////////////// Filter Data "hn" ////////////////////////
124   ////////////////////////////////////////////////////////////////////
      vector<t_complex> impulseResponse;
126   impulseResponse = transferFunctionToImpulseResponse(transferFunction);
      vector<t_complex> hn = impulseResponse;
128
      ////////////////////// OverlapSave in Realtime ////////////////////
130   ////////////////////////////////////////////////////////////////////
      vector<t_complex> OUTaux = overlapSave(currentCopy, previousCopy, hn);
132
      previousCopy = currentCopy;
134   K = K + 1;

136   // Remove the size modified data (opppsite to "currentCopyAux to currentCopy")
      vector<t_complex> OUT;
138   for  (int  i  = 0;  i  < process;  i++){
        OUT.push_back(OUTaux[previousCopy.size() + i]);
140   }

142   // Bufferput
      for  (int  i  = 0;  i  < process;  i++){
144     t_real  val;
        val  = OUT[i].real();
146     outputSignals[0]->bufferPut((t_real)(val));
      }
148
      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 files 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.

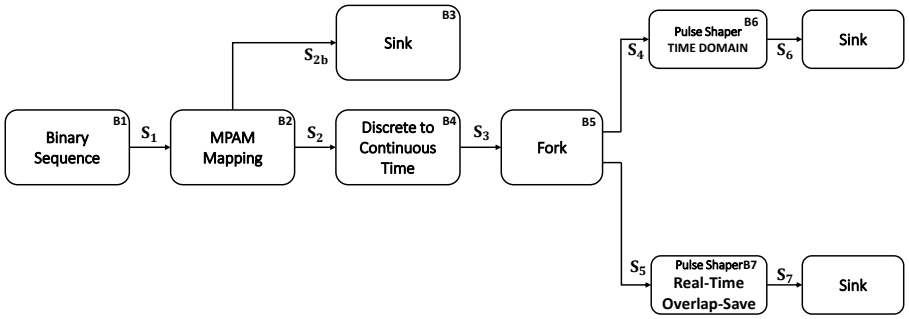

Figure 9.26: Real-time overlap-save example setup

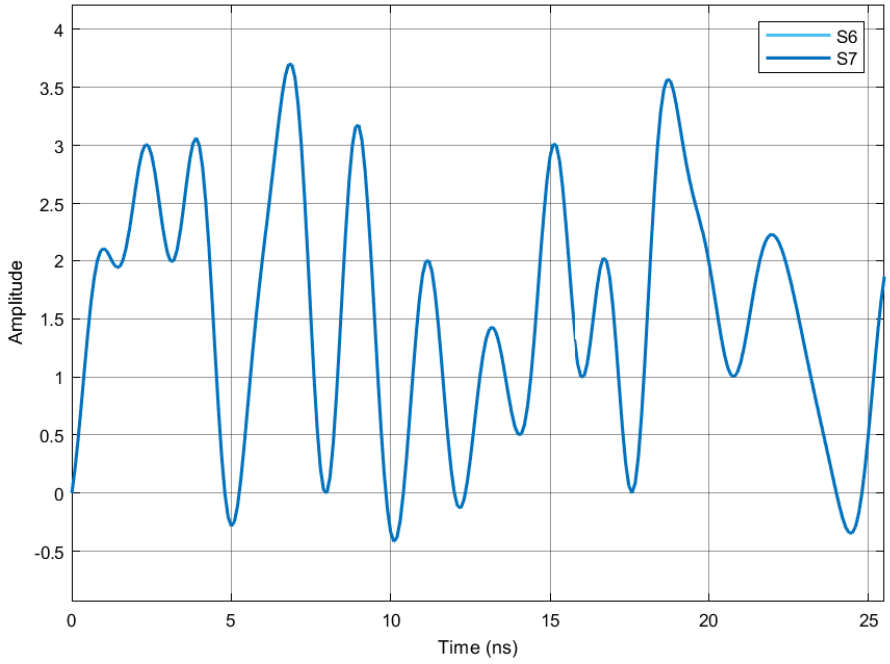

Figure 9.27: Comparison of signal S6 and S7

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

Header File	:	filter_*.h
Source File	:	filter_*.cpp
Version	:	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**

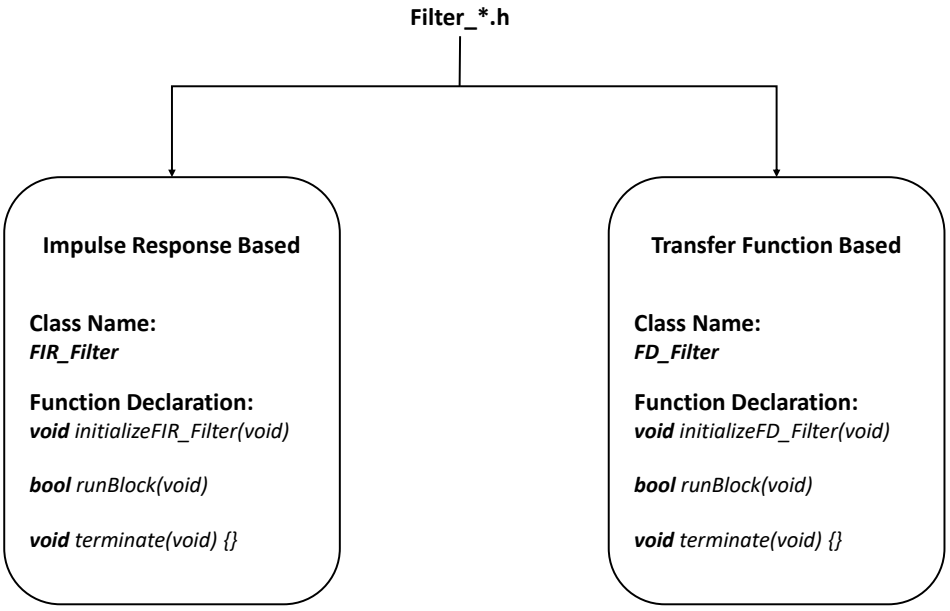

Figure 9.28: Filter class

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
```

```
  int ready = inputSignals[0]->ready();
  int space = outputSignals[0]->space();
  int process = min(ready, space);
  if (process == 0) return false;

  for (int i = 0; i < process; i++) {
    t_real val;
    (inputSignals[0])->bufferGet(&val);
    if (val != 0) {
      vector<t_real> aux(impulseResponseLength, 0.0);
      transform(impulseResponse.begin(), impulseResponse.end(), aux.begin(), bind1st(multiplies<t_real>(),
     val));
      transform(aux.begin(), aux.end(), delayLine.begin(), delayLine.begin(), plus<t_real>());
    }
    outputSignals[0]->bufferPut((t_real)(delayLine[0]));
    rotate(delayLine.begin(), delayLine.begin() + 1, delayLine.end());
    delayLine[impulseResponseLength - 1] = 0.0;
  }

  return true;
};
```

Listing 9.6: Definition of **bool FIR_Filter::runBlock(void)**

```
bool FD_Filter::runBlock(void)
{
  bool alive{ false };

  int ready = inputSignals[0]->ready();
  int space = outputSignals[0]->space();
  int process = min(ready, space);
  if (process == 0) return false;

  ////////////////////// previousCopy & currentCopy ////////////////////
  ///////////////////////////////////////////////////////////////////////
  vector<double> re(process); // Get the Input signal
  t_real input;
  for (int i = 0; i < process; i++){
    inputSignals[0]->bufferGet(&input);
    re.at(i) = input;
  }

  vector<t_real> im(process);
  vector<t_complex> currentCopyAux = reImVect2ComplexVector(re, im);

  vector<t_complex> pcInitialize(process);  // For the first data block only
  if (K == 0){ previousCopy = pcInitialize; }

  // size modification of currentCopyAux to currentCopy.
  vector<t_complex> currentCopy(previousCopy.size());
  for (unsigned int i = 0; i < currentCopyAux.size(); i++){
    currentCopy[i] = currentCopyAux[i];
```

```
    }

    ////////////////////////// Filter Data "hn" ////////////////////////
    ////////////////////////////////////////////////////////////////////
    vector<t_complex> impulseResponse;
    impulseResponse = transferFunctionToImpulseResponse(transferFunction);
    vector<t_complex> hn = impulseResponse;

    ////////////////////// OverlapSave in Realtime ////////////////////
    ////////////////////////////////////////////////////////////////////
    vector<t_complex> OUTaux = overlapSave(currentCopy, previousCopy, hn);

    previousCopy = currentCopy;
    K = K + 1;

    // Remove the size modified data (opppsite to "currentCopyAux to currentCopy")
    vector<t_complex> OUT;
    for (int i = 0; i < process; i++){
      OUT.push_back(OUTaux[previousCopy.size() + i]);
    }

    // Bufferput
    for (int i = 0; i < process; i++){
      t_real val;
      val = OUT[i].real();
      outputSignals[0]->bufferPut((t_real)(val));
    }

    return true;
}
```

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.

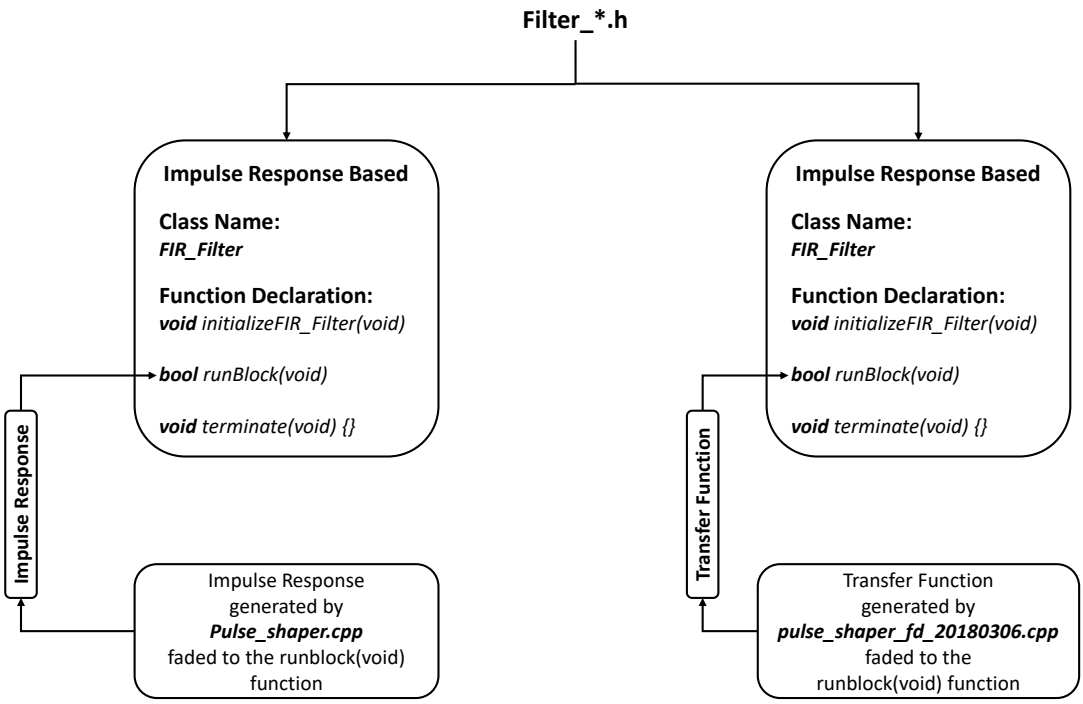

Figure 9.29: Pulse shaping using **filter_*.h**

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 sapping 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

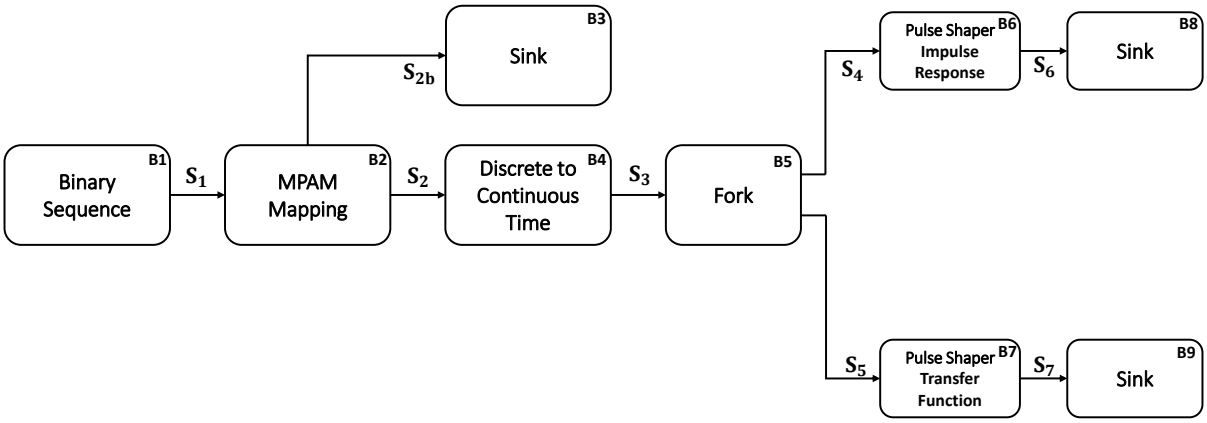

Figure 9.30: Filter test setup

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

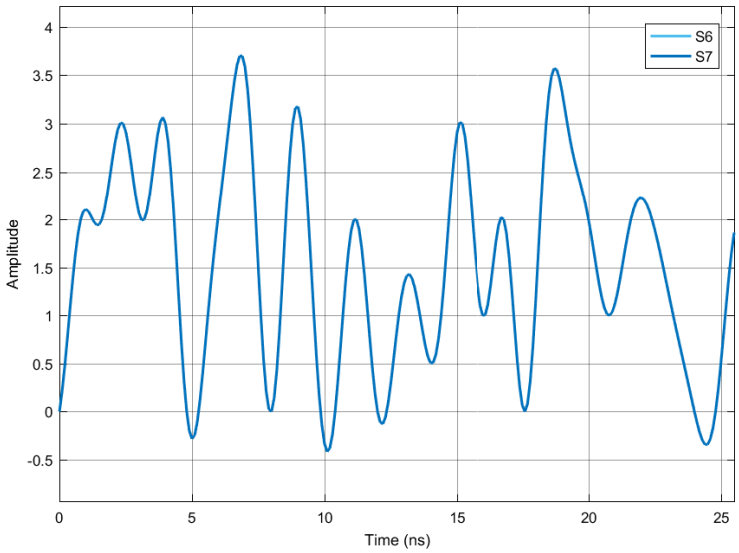

Figure 9.31: Raised cosine pulse shaping results comparison

Case 2 : Root raised cosine

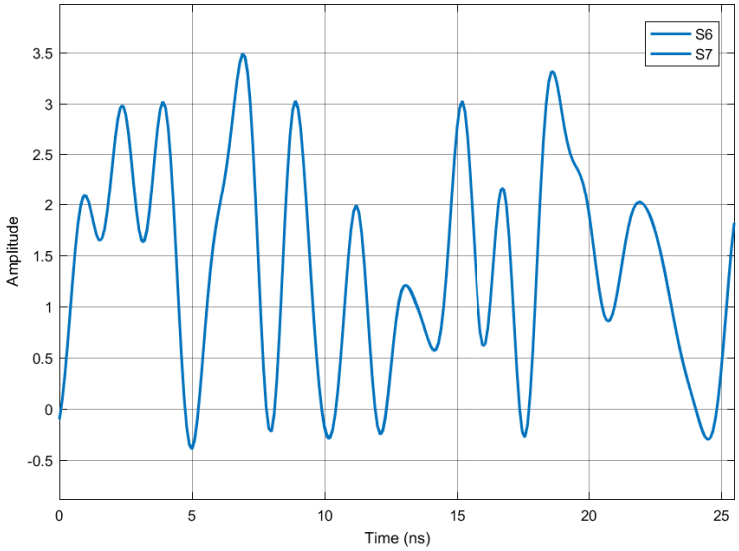

Figure 9.32: Root raised cosine pulse shaping result

Case 3 : Gaussian

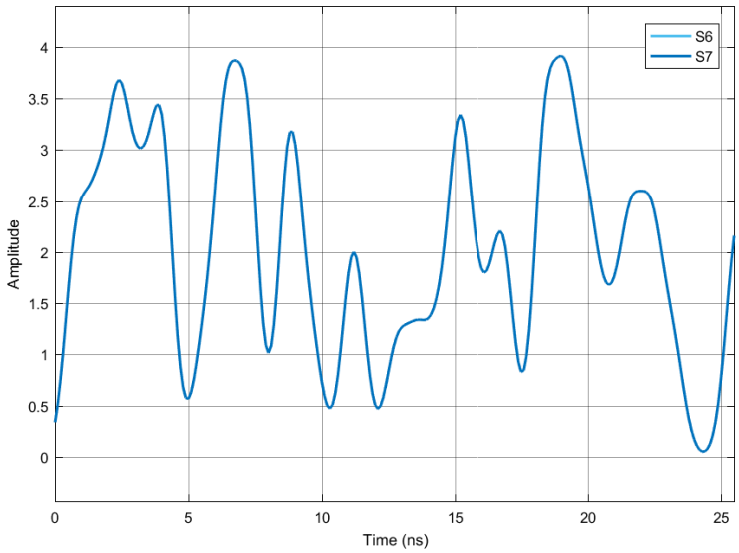

Figure 9.33: Gaussian pulse shaping results comparison

APPENDICES

A. Raised cosine pulse shaper

The raised cosine pulse shaping filter has a transfer function given by,

$$H_{RC}(f) = \begin{cases} 1 & \text{for } |f| \leq \dfrac{1-\beta}{2T_s} \\ \dfrac{1}{2}\left[1 + cos\left(\dfrac{\pi Ts}{\beta}\left[|f| - \dfrac{1-\beta}{2T_s}\right]\right)\right] & \text{for } \dfrac{1-\beta}{2T_s} < |f| \leq \dfrac{1+\beta}{2T_s} \\ 0 & \text{otherwise} \end{cases} \tag{9.7}$$

The parameter, β is the roll-off factor of the raised cosine filter. The impulse response of the raised cosine filter is given by,

$$h_{RC}(t) = \frac{sin(\pi t/T_s)}{\pi t/Ts}\frac{cos(\pi\beta t/Ts)}{1 - 4\beta^2 t^2/T_s^2} \tag{9.8}$$

B. Root raised cosine pulse shaper

The raised cosine pulse shaping filter has a transfer function given by,

$$H_{RC}(f) = \begin{cases} 1 & \text{for } |f| \leq \dfrac{1-\beta}{2T_s} \\ \sqrt{\dfrac{1}{2}\left[1 + cos\left(\dfrac{\pi Ts}{\beta}\left[|f| - \dfrac{1-\beta}{2T_s}\right]\right)\right]} & \text{for } \dfrac{1-\beta}{2T_s} < |f| \leq \dfrac{1+\beta}{2T_s} \\ 0 & \text{otherwise} \end{cases} \tag{9.9}$$

The parameter, β is the roll-off factor of the raised cosine filter. The impulse response of the root raised cosine filter is given by,

$$h_{RRC}(t) = \begin{cases} \dfrac{1}{T_s}\left(1 + \beta(\dfrac{4}{\pi} - 1)\right) & \text{for } t = 0 \\ \dfrac{\beta}{T_s\sqrt{2}}\left[\left(1 + \dfrac{2}{\pi}\right)sin\left(\dfrac{\pi}{4\beta}\right) + \left(1 - \dfrac{2}{\pi}\right)cos\left(\dfrac{\pi}{4\beta}\right)\right] & \text{for } t = \dfrac{T_s}{4\beta} \\ \dfrac{1}{Ts}\dfrac{sin\left[\pi\dfrac{t}{T_s}(1-\beta)\right] + 4\beta\dfrac{t}{Ts}cos\left[\pi\dfrac{t}{T_s}(1+\beta)\right]}{\pi\dfrac{t}{T_s}\left[1 - \left(4\beta\dfrac{t}{T_s}\right)^2\right]} & \text{otherwise} \end{cases} \tag{9.10}$$

C. Gaussian pulse shaper

The Gaussian pulse shaping filter has a transfer function given by,

$$H_G(f) = exp(-\alpha^2 f^2) \tag{9.11}$$

The parameter α is related to B, the 3-dB bandwidth of the Gaussian shaping filter is given by,

$$\alpha = \frac{\sqrt{ln2}}{\sqrt{2}B} = \frac{0.5887}{B} \tag{9.12}$$

From the equation 9.12, as α increases, the spectral occupancy of the Gaussian filter decreases. The impulse response of the Gaussian filter can be given by,

$$h_G(t) = \frac{\sqrt{\pi}}{\alpha} exp(-\frac{\pi^2}{\alpha^2} t^2) \tag{9.13}$$

From the equation 9.12, we can also write that,

$$\alpha = \frac{0.5887}{BT_s} T_s \tag{9.14}$$

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

Header File	:	hilbert_filter_*.h
Source File	:	hilbert_filter_*.cpp
Version	:	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.

$$s_a(t) = s(t) + i\hat{s}(t) \tag{9.15}$$

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 $S_a(t)$ directly, on the other hand, second method will generate the $\hat{s}(t)$ signal which is multiplied with i and added to the $s(t)$ to generate the analytical signal $S_a(t)$.

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)

$$S_a(f) = \begin{cases} 2S(f) & \text{for } f > 0 \\ S(f) & \text{for } f = 0 \\ 0 & \text{for } f < 0 \end{cases} \tag{9.16}$$

which is inverse transformed to obtain an analytical signal $S_a(t)$.

Method 2 :

The discrete time Hilbert transformed signal $\hat{s}(t)$ corresponding to $s(t)$ is defined in the frequency domain as [2]

$$\hat{S}(f) = \begin{cases} i\ S(f) & \text{for } f > 0 \\ 0 & \text{for } f = 0 \\ -i\ S(f) & \text{for } f < 0 \end{cases} \tag{9.17}$$

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

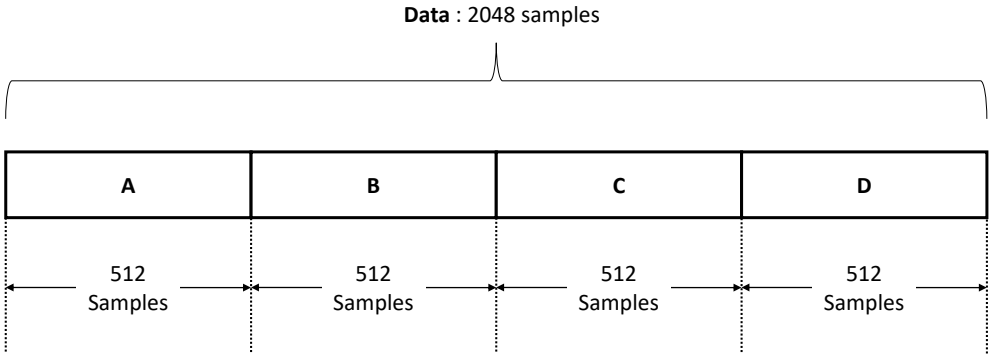

Figure 9.34: Logical flow

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 x $3 = 1536$ 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"

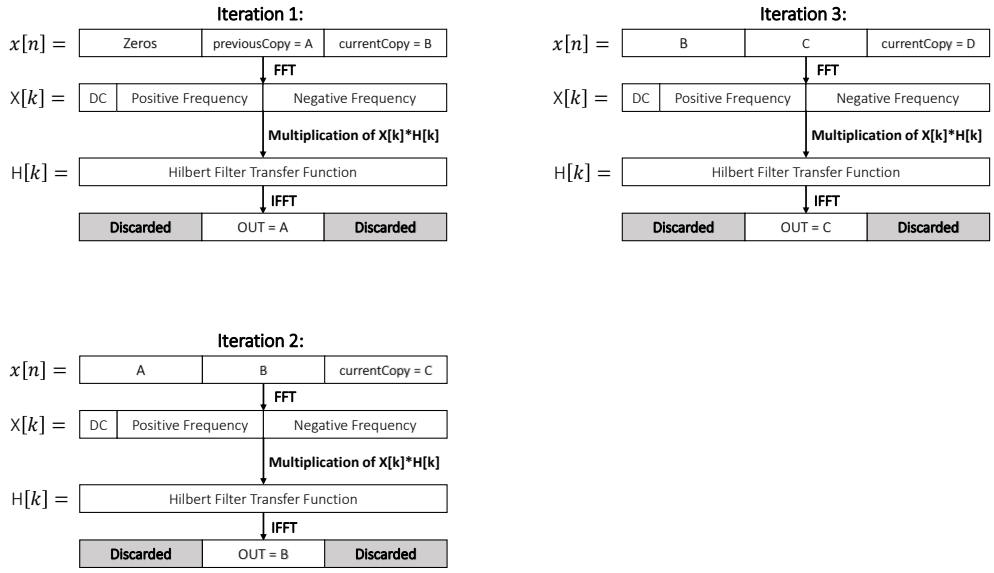

Figure 9.35: Logical flow of real-time Hilbert transform

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

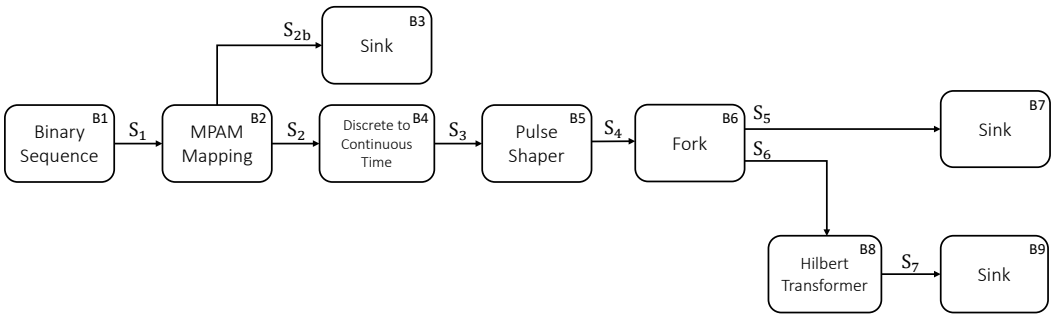

Figure 9.36: Test setup for the real time Hilbert transform

Real-time Hilbert transform : Results

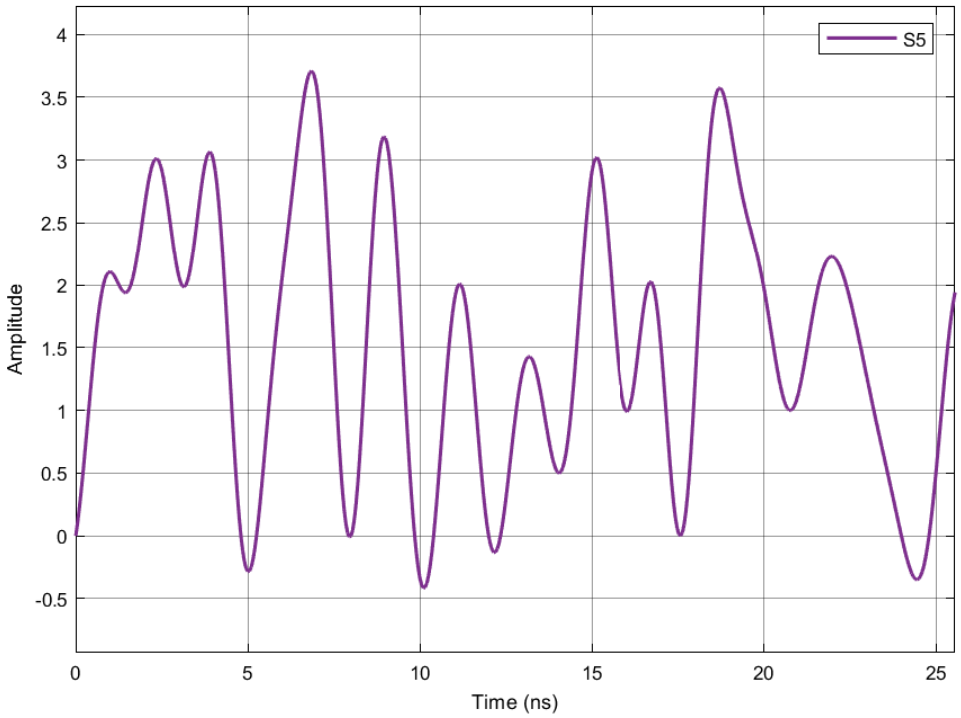

Figure 9.37: $S5$ signal

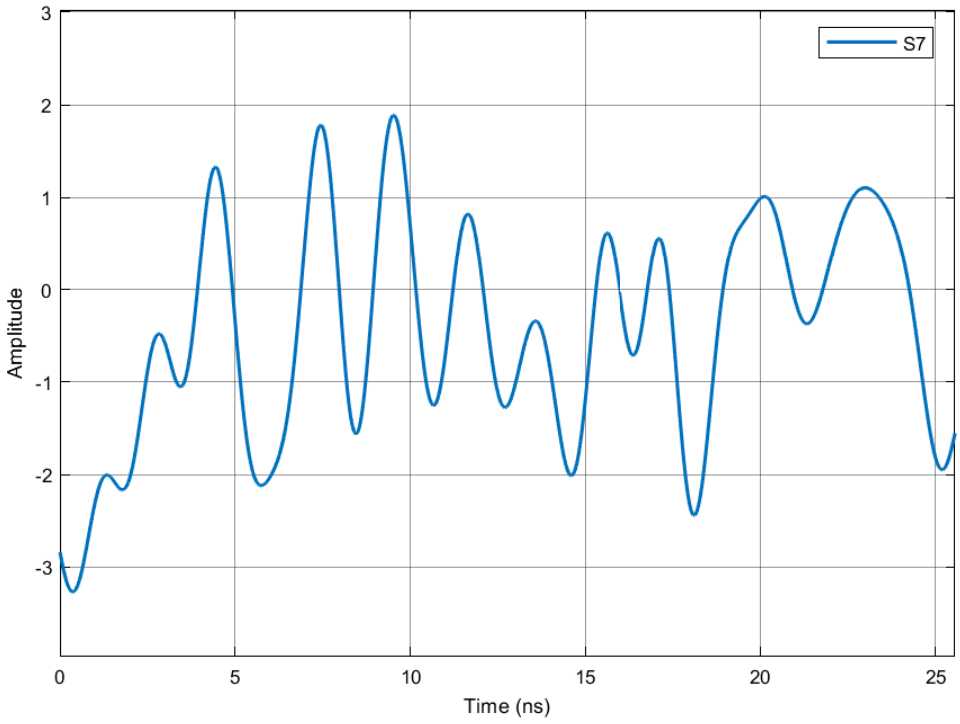

Figure 9.38: $S7$ signal

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. 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

github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md

10.0.1 Integrated Development Environment

The recommended IDE is the Visual Studio 2017. To install the Visual Studio 2017 first instal the Microsoft Visual Installer and after proceed to the Visual Studio 2017 installation.

Visual Studio Community 2017 - version 15.7.6

10.0.2 Compiler Switches

Disable Language Extensions	No
Conformance mode	No
C++ Language Standard	ISO C++14 Standard (std:c++ 14)

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

12.1 Starting with Git

Git is a free and open source distributed version control system [1]. Git creates and maintains a database that records all changes that occur in a folder. The Git database is named a repository. It also allows to merge repositories that shared a common state in the past. These can be local repositories, i.e. stored in the same machine, or can be remote repositores, i.e. stored anywhere.

To create this database for a specific folder the Git application must be installed on the computer. The Git application and directions for its installation in all majors platforms can be obtained in the Git website (http://git-scm.com). You can access to the Git commands through the console or through a GUI interface. Here, we assume that you are going to use the console.

After the installation, to create the Git initial database open the console program and go to a folder and execute the following command:

git init

The Git database is created and stored in the folder *.git* in the root of your folder. The *.git* folder is your repository.

The Git commands allow you to manipulate this database, i.e. this repository.

12.2 Data Model

To understand Git is fundamental to understand the 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;
- tags - text files that store information about commits.

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. The SHA1 is just an algorithm

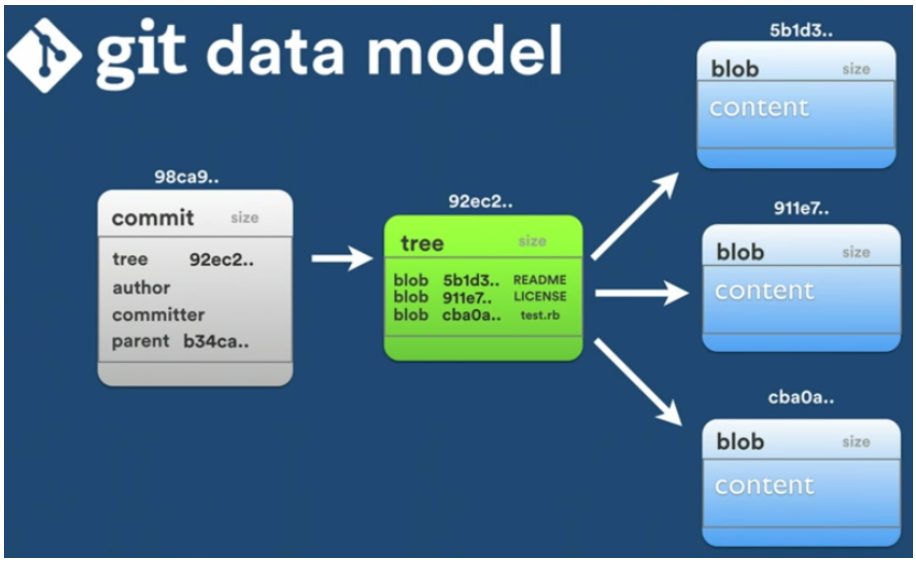

Figure 12.1: Git data model.

that accepts some binary information and generates 20 bytes which are ideally unique for that information. The probability of collisions is extremely low, i.e. the probability that different information generates the same hash value is extremely low. 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 remanning thirty eight characters of the SHA1 hash value. The Git stores all committed versions of a file. The Git maintains a contend-addressable file systems, i.e. a file system in which the files can be accessed based on its contend. Lets look more carefully in each stored object.

A **commit** object is identified by a SHA1 hash value, and has the following information: a pointer for a tree (the root), a pointer for the previous commit, the author, 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. A tree object identifies a folder and its contend. 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 object type, a space character, the file size in bytes and the \NUL caracter, for instance "blob 13\NUL", the tailor is just the \n caracter. The compressed blob (header+file contend+tailer) is stored as a binary file.

There are two types of **tags**, lightweight and annotated tags. Lightweight tags are only a ref to a commit, see section below. Annotated tags are objects stored as text files, which as has information about the commit, to each the tag point, the tagger (name and e-mail), tag date and a tag message.

12.2.1 Objects Folder

Git stores the database and the associated information in a set of folders and files inside the the folder *.git* in the root of your repository.

The folder *.git/objects* stores information about all objects (commits, trees, blobs and annotated tags). 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 space.

12.3 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, used by the computer. Therefore refs are pointers to objects. Refs are implementes 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. Tags and branches are example of refs. Tags are static references, i.e. tags are never updated, and branches are dynamic references, i.e. branches are always automatically updated.

12.3.1 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.

12.3.2 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 atualize so that it always points for the most recent commit of that branch.

12.3.3 Heads

Heads is a pointer for the branch where we are. If we are in a commit that is not pointed by a branch we are in a detached HEAD situation.

12.4 Git Logical Areas

Git uses several spaces.

- Working tree - is your directories where you are working;
- Staging area or index - temporary area used to specify which files are going to be committed in the next commit;
- History - recorded commits;

All information related with the staging area and the history is stored in the *.git* folder.

12.5 Merge

Merge is a fundamental concept to git. It is the way you consolidate your work.

12.5.1 Fast-Forward Merge

It is used when there is a direct path between the two branches, the older branch is just updated.

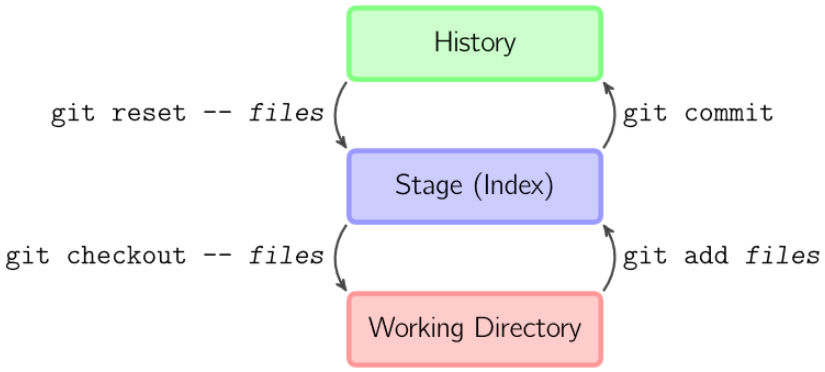

Figure 12.2: Git logical areas. Figure adapted from [2].

12.5.2 Three-Way Merge

It is used when there is no direct path between the two branches. In this case a common commit is found considering the two branches and the merge is performed using a recursive strategy. In this process conflicts can occur. The recursive strategy is applied to each modified file and line by line. If the line was not modified in both branches the line is not modified in the merged file. If the line was modified only in one branch the line is going to be modified in the merged file. If the line is modified in both branches Git cannot make a decision and a conflict occur. So, conflicts occur when branches that change the same file in the same line are being merged.

12.6 Remotes

A remote is a repository in another location. The remote location can be the http://github.com or the location of any other Git server.

12.6.1 GitHub

GitHub is a Git server that stores public repositories. A GitHub user will have a user name and password and inside his or her account creates public repositories. This repositories can be transferred to a local machine using the command *git clone <repository url>*. The local and remote repository will be linked and the local repository can be updated using the *git fetch* or *git pull* command. The remote repository can be updated using the *git push* command. When the remote repository is cloned an alias, named origin, is created that points to that remote repository. Another way to create a GitHub repository is by forking another existente repository. Forked repositories are linked and they can be syncronized using pull requests.

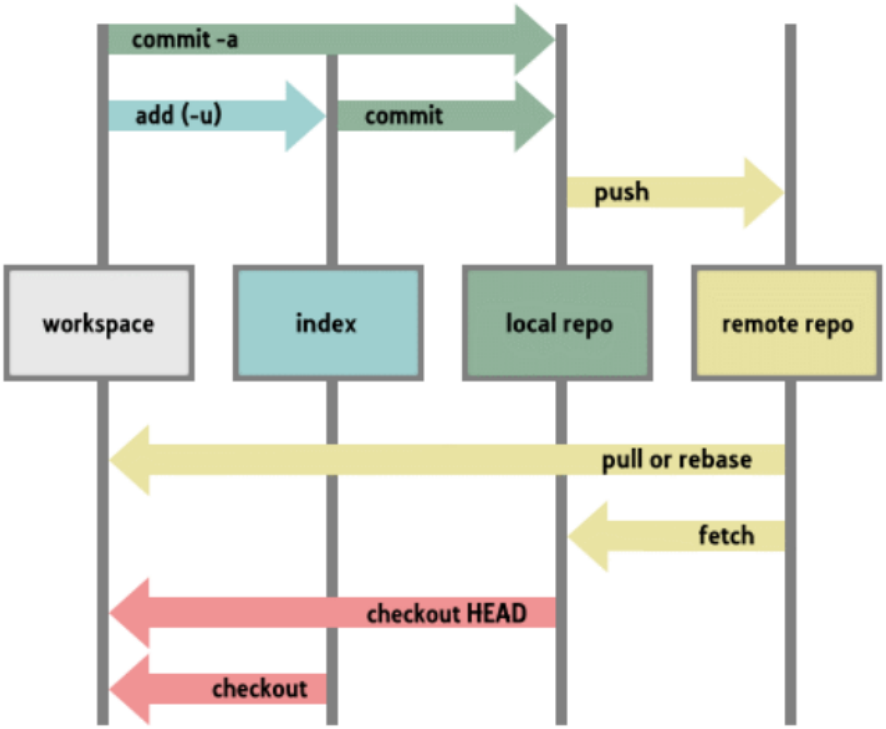

Figure 12.3: Git remotes.

12.7 Commands

12.7.1 Porcelain Commands

Porcelain commends are high-level commands.

git add
git add, adds a a new or modified file (or files) to the staging area.

git branch
git branch, lists local branches.
git branch -r, lists remote branches.
git branch -a, lists all branches.
git branch -u <remote>/<branch>, links the local branch in which you are in with a remote branch.
git branch –set-upstream-to=<remote>/<branch>, longer version of the previous command.
git branch -u <remote>/<branch> <local_branch>, links the local branch with a remote branch.
git branch –set-upstream-to=<remote>/<branch> <local_branch>, longer version of the previous command.

git cat-file

git cat-file -t <hash>, shows the type of the object identified by the hash value.
git cat-file -p <hash>, shows the contend of the file associated with the object identified by the hash value.

git checkout
git checkout -b <new_branch_name>, creates a new branch in the same position as the current branch and move to it.
git checkout -b <new_branch_name> <branch_name>, create a new branch in the position of <branch_name> and move to it.

git clean
git clean -f -d, removes from the working directory all untracked directories (d) and files (f).

git clone
git clone <url>, downloads the contend of the <url> repository, for instance http://www.github.com/netxpto/linkplanner.git, and creates a local repo.

git config
git config –global user.name "netxpto", sets the user name globally.
git config –global user.email "netxpto.gmail.com", sets the user e-mail globally.
git config –global user.emmail "netxpto.gmail.com", sets the user e-mail globally.
git config –global alias.<alias name> <commands>, creates a global alias for the commands.

git diff
git diff, shows the changes between the working space and the staging area.
git diff –name-only, shows the changes between the working space and the staging area, in the specified files.
git diff –cached, shows the changes between the staging area and the current branch history. Note that –cached or –staged are synonymous.
git diff –cached –name-only, shows the changes between the staging area and the current branch history, in the specified files.

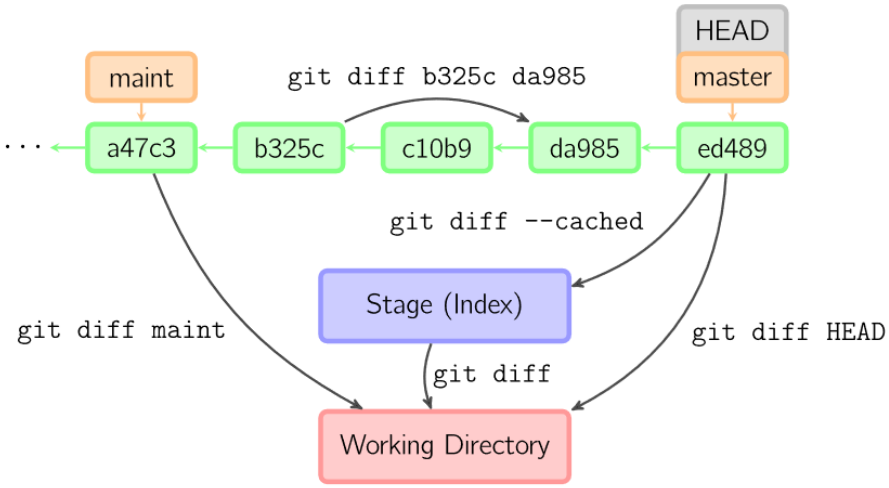

Figure 12.4: Git diff command. Figure adapted from [2].

git fetch
git fetch –all, downloads all history from all remote repositories.
git fetch <repository>, downloads all history from the remote repository.

git init
git init, inicializes a git repository. It creates the *.git* folder and all its subfolders and files.

git log
git log, shows a list of commits from reverse time order.
git log –graph –decorate –oneline <commit1>..<commit1>, shows the history of the repository in a colourful way.
git log –graph, shows a graphical representation of the repository commits history.
git log –stat, shows the name of the files that were changed in each commit
git log –follow <file>, lists version history for a file, including rename.

git ls-files

git ls-remote

git merge
git merge <branch>, combines the specified branchâ€™s history into the current branch.

git pull

git pull, downloads remote branch history and incorporates changes.

git push
git push, uploads local branch commits to remote repository branch.

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 remote
git remote, shows a list of existing remotes.
git remote -v, shows the full location of existing remotes.
git remote add <remote name> <remote repository url>, adds a remote.
git remote remove <remote name>, removes a remote.

git reset
git reset –soft <commit_hash_value>, moves to the commit identified by <commit_hash_value> but leaves in the staging area all modified files.
git reset –hard <commit_hash_value>, moves to the commit identified by <commit_hash_value> and cleans all modified tracked files.
git reset <commit_hash_value>, this is a mix reset (is the default reset), puts all modified files in the working area.
git reset <file>, unstages the file, but preserve its contents.

git reflog
git reflog, shows all commands from the last 90 days. Git only perform garbaged collection after 30 days.

git rm
git rm <file>, deletes the file from the working directory and stages the deletion.
git rm –cached <file>, removes the file from version control but preserves the file locally.

git show
git show, shows what is new in the last commit.

git stash
git stash, temporarily stores all modified tracked files.
git stash –list, shows what is in the stash.
git stash pop, restores the most recently stashed files.

git stash drop, discards the most recently stashed changeset.

git status
git status, lists all new or modified files to be committed.

12.7.2 Pluming Commands

Pluming commands are low-level commands.

git cat-files
git cat-files -p <sha1>, shows the contend of a file in a pretty (-p) readable format.
git cat-files -t <sha1>, shows the type of a object, i.e. blob, tree or commit.

git count-object
git count-object -H, counts all object and shows the result in a (-H) human readable form.

git gc
git gc, garbage collector, eliminates all objects that has no reference associated with.
git gc –prune=all

git hash-object
git hash-object <file>, calculates the SHA1 hash value of a file plus a header.
git hash-object -w <file>, calculates the SHA1 hash value of a file plus a header and write it in the *.git/objects* folder.

git merge-base
git merge-base <branch1> <brach2>, finds the base commit for the three-way merge between <branch1> and <branch2>.

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 rev-parse
git rev-parse <ref>, return the hash value of <ref>.

git rev-parse <short_hash_value>, return the full hash value associated with <short_hash_value>.

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

12.8 Navigation Helpers

<*ref*>^, one commit before <ref>.
<*ref*>^^, two commits before <ref>.
<*ref*>~*5*, five commits before <ref>.
<*ref1*>..<*ref2*>, between commit <ref1> and <ref2>.
<*branch*>^*tree*, identifies the tree pointed by the commit pointed by <branch>.
<*commit*>:<*file_name*>, identifies the version of a file in a given commit.

12.9 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

12.10 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.

12.11 Applications

12.11.1 Meld

12.11.2 GitKraken

12.12 Error Messages

12.12.1 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.

12.13 Git with Overleaf

You can use git with overleaf. For that you have to create a project on overleaf. Associate with that overleaf project it is also create a git repository. The address of that git repository is almost the same as the overleaf project that you can obtain going to the overleaf Menu/Share. Let's assume that the overleaf project address is:

https://www.overleaf.com/12925162jkwbhrdkwrfm

In this case the repository address is https://git.overleaf.com/12925162jkwbhrdkwrfm
The only change was the replacement of **www** by **git**.
Now you can just do
git clone https://git.overleaf.com/12925162jkwbhrdkwrfm
and clone your repository.
You can also do
git push https://git.overleaf.com/12925162jkwbhrdkwrfm

Bibliography

[1] Scott Chacon and Ben Straub. *Pro Git, 2nd Edtion*. Apress, 2014.

[2] Feb. 4, 2019. URL: https://marklodato.github.io/visual-git-guide.

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\ghdl\bin**.
   Example: **C:\repos\LinkPlanner\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\ghdl\bin**.
   Example: **C:\repos\LinkPlanner\ghdl\bin**.
10. Press **Ok** and you're done.

13.2 Using GHDL To Simulate VHDL Programs

This guide is meant to explain how to execute the testbench for module CPE BPS.
This simulation will take two .sgn files as input and produce two .sgn files.

13.2.1 Simulation Input

Make sure that files S19.sgn and S20.sgn exist in directory **\LinkPlanner\sdf\dsp_laser_phase_compensation\signals**. The content of these files will be used as input of the CPE BPS module.

13.2.2 Executing Testbench

Execute the batch file **simulation.bat**, located in the directory **\LinkPlanner\sdf\dsp_laser_phase_compensation\VHDL\Simulator** of this repository.

13.2.3 Simulation Output

The simulation will produce two files: **sim_out_1.sgn** and **sim_out_2.sgn** which will contain the output of the CPE BPS module.