NetXPTO - LinkPlanner

13 de Julho de 2017

Conteúdo

Capítulo 1

Introduction

Capítulo 2

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.

2.1 System

You can run the System

Capítulo 3

Development Cycle

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

Capítulo 4

Visualizer

visualizer

Capítulo 5

Case Studies

5.1 QPSK Transmitter

This system simulates a QPSK transmitter. A schematic representation of this system is shown in figure 5.1.

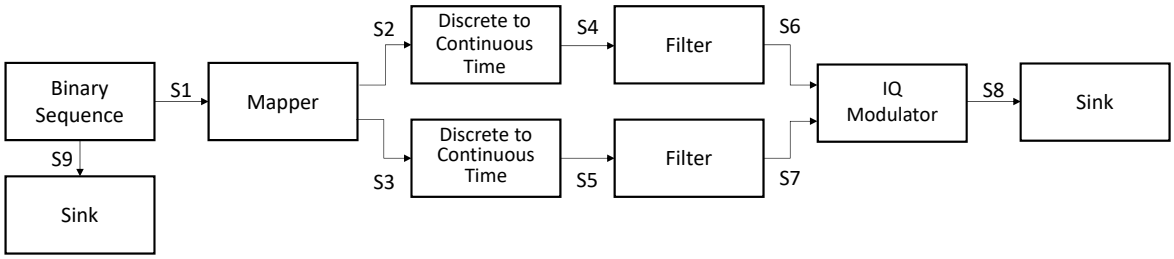

Figura 5.1: QPSK transmitter block diagram.

System Input Parameters

Parameter: sourceMode

Description: Specifies the operation mode of the binary source.

Accepted Values: PseudoRandom, Random, DeterministicAppendZeros, DeterministicCyclic.

Parameter: patternLength

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

Accepted Values: Integer between 1 and 32.

Parameter: bitStream

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

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

Input parameters	**Description**	**Accepted values**
bitStream	Specifies the bit stream generated by the source in the DeterministicCyclic and DeterministicAppendZeros mode.	"XXX..", where X is 0 or 1.
Number of bits generated	setNumberOfBits()	Any integer
Number of bits	setNumberOfBits()	Integer number greater than zero
Number of samples per symbol	setNumberOfSamplesPerSymbol()	Integer number of the type 2^n with n also integer
Roll of factor	setRollOfFactor()	$\in$ [0,1]
IQ amplitudes	Vector of coordinate points in the I-Q plane	**Example** for a 4-qam mapping: { { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }
Output optical power	setOutputOpticalPower()	Real number greater than zero
Save internal signals	setSaveInternalSignals()	True or False

Tabela 5.1: List of input parameters of the block MQAM transmitter

Functional description

This block generates an optical signal (output signal 1 in figure **??**). The binary signal generated in the internal block Binary Source (block B1 in figure **??**) can be used to perform a Bit Error Rate (BER) measurement and in that sense it works as an extra output signal (output signal 2 in figure **??**).

Input parameters

This block has a special set of functions that allow the user to change the basic configuration of the transmitter. The list of input parameters, functions used to change them and the values that each one can take are summarized in table 5.2.

Input parameters	**Function**	Type	**Accepted values**
Mode	setMode()	string	PseudoRandom Random DeterministicAppendZeros DeterministicCyclic
Number of bits generated	setNumberOfBits()	int	Any integer
Pattern length	setPatternLength()	int	Real number greater than zero
Number of bits	setNumberOfBits()	long	Integer number greater than zero
Number of samples per symbol	setNumberOfSamplesPerSymbol()	int	Integer number of the type 2^n with n also integer
Roll of factor	setRollOfFactor()	double	$\in$ [0,1]
IQ amplitudes	setIqAmplitudes()	Vector of coordinate points in the I-Q plane	**Example** for a 4-qam mapping: { { 1.0, 1.0 }, { -1.0, 1.0 }, { -1.0, -1.0 }, { 1.0, -1.0 } }
Output optical power	setOutputOpticalPower()	int	Real number greater than zero
Save internal signals	setSaveInternalSignals()	bool	True or False

Tabela 5.2: List of input parameters of the block MQAM transmitter

Methods

MQamTransmitter(vector<Signal *> &inputSignal, vector<Signal *> &outputSignal); (**constructor**)

void set(int opt);

void setMode(BinarySourceMode m)

BinarySourceMode const getMode(void)

void setProbabilityOfZero(double pZero)

double const getProbabilityOfZero(void)

void setBitStream(string bStream)

string const getBitStream(void)

void setNumberOfBits(long int nOfBits)

long int const getNumberOfBits(void)

void setPatternLength(int pLength)

int const getPatternLength(void)

void setBitPeriod(double bPeriod)

double const getBitPeriod(void)

void setM(int mValue) int const getM(void)

void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues)

vector<t_iqValues> const getIqAmplitudes(void)

void setNumberOfSamplesPerSymbol(int n)

int const getNumberOfSamplesPerSymbol(void)

void setRollOffFactor(double rOffFactor)

double const getRollOffFactor(void)

void setSeeBeginningOfImpulseResponse(bool sBeginningOfImpulseResponse)

double const getSeeBeginningOfImpulseResponse(void)

void setOutputOpticalPower(t_real outOpticalPower)

t_real const getOutputOpticalPower(void)

void setOutputOpticalPower_dBm(t_real outOpticalPower_dBm)

t_real const getOutputOpticalPower_dBm(void)

Output Signals

Number: 1 optical and 1 binary (optional)

Type: Optical signal

Example

Sugestions for future improvement

Add to the system another block similar to this one in order to generate two optical signals with perpendicular polarizations. This would allow to combine the two optical signals and generate an optical signal with any type of polarization.

Capítulo 6

Library

6.1 Add

Input Parameters

This block takes no parameters.

Functional Description

This block accepts two signals and outputs one signal built from a sum of the two inputs. The input and output signals must be of the same type.

Input Signals

Number: 2

Type: Real, Complex or Complex_XY signal (ContinuousTimeContinuousAmplitude)

Output Signals

Number: 1

Type: Real, Complex or Complex_XY signal (ContinuousTimeContinuousAmplitude)

6.2 Binary source

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

This blocks doesn't accept any input signal. It produces any number of output signals.

Input Parameters

Parameter: mode{PseudoRandom}
(Random, PseudoRandom, DeterministicCyclic, DeterministicAppendZeros)

Parameter: probabilityOfZero{0.5}
(real $\in$ [0,1])

Parameter: patternLength{7}
(integer $\in$ [1,32])

Parameter: bitStream{"0100011101010101"}
(string of 0's and 1's)

Parameter: numberOfBits{-1}
(long int)

Parameter: bitPeriod{1.0/100e9}
(double)

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)

Examples

Random Mode

PseudoRandom Mode As an example consider a pseudorandom sequence with *patternLength*=3 which contains a total of 7 ($2^3 - 1$) bits. In this sequence it is possible to find every combination of 0's and 1's that compose a 3 bit long subsequence with the exception of 000. For this example the possible subsequences are 010, 110, 101, 100, 111, 001 and 100 (they appear in figure 6.1 numbered in this order). Some of these require wrap.

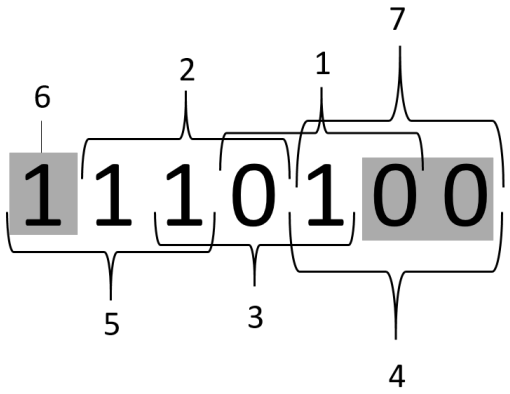

Figura 6.1: Example of a pseudorandom sequence with a pattern length equal to 3.

DeterministicCyclic Mode As an example take the *bit stream* '0100011101010101'. The generated binary signal is displayed in.

DeterministicAppendZeros Mode Take as an example the *bit stream* '0100011101010101'. The generated binary signal is displayed in 6.2.

Sugestions for future improvement

Implement an input signal that can work as trigger.

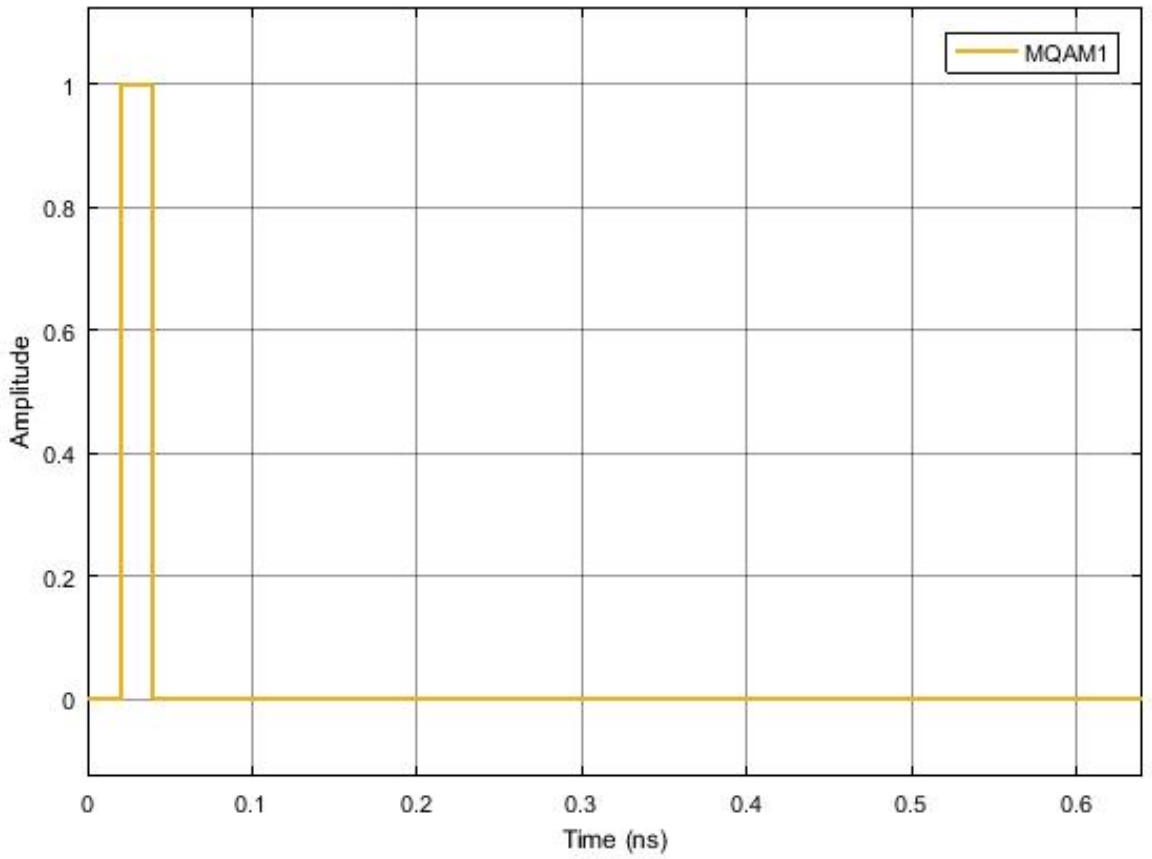

Figura 6.2: Binary signal generated by the block operating in the *Deterministic Append Zeros* mode with a binary sequence 01000...

6.3 Decoder

This block accepts a complex electrical signal and outputs a sequence of binary values (0's and 1's). Each point of the input signal corresponds to a pair of bits.

Input Parameters

Parameter: `t_integer` m{ 4 }

Parameter: vector<`t_complex`> iqAmplitudes{ { 1.0, 1.0 },{ -1.0, 1.0 },{ -1.0, -1.0 },{ 1.0, -1.0 } };

Methods

Decoder()

Decoder(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig)

void initialize(void)

bool runBlock(void)

void setM(int mValue)

void getM()

void setIqAmplitudes(vector<`t_iqValues`> iqAmplitudesValues)

vector<`t_iqValues`>getIqAmplitudes()

Functional description

This block makes the correspondence between a complex electrical signal and pair of binary values using a predetermined constellation.

To do so it computes the distance in the complex plane between each value of the input signal and each value of the *iqAmplitudes* vector selecting only the shortest one. It then converts the point in the IQ plane to a pair of bits making the correspondence between the input signal and a pair of bits.

Input Signals

Number: 1

Type: Electrical complex (TimeContinuousAmplitudeContinuousReal)

Output Signals

Number: 1

Type: Binary

Examples

As an example take an input signal with positive real and imaginary parts. It would correspond to the first point of the *iqAmplitudes* vector and therefore it would be associated to the pair of bits 00.

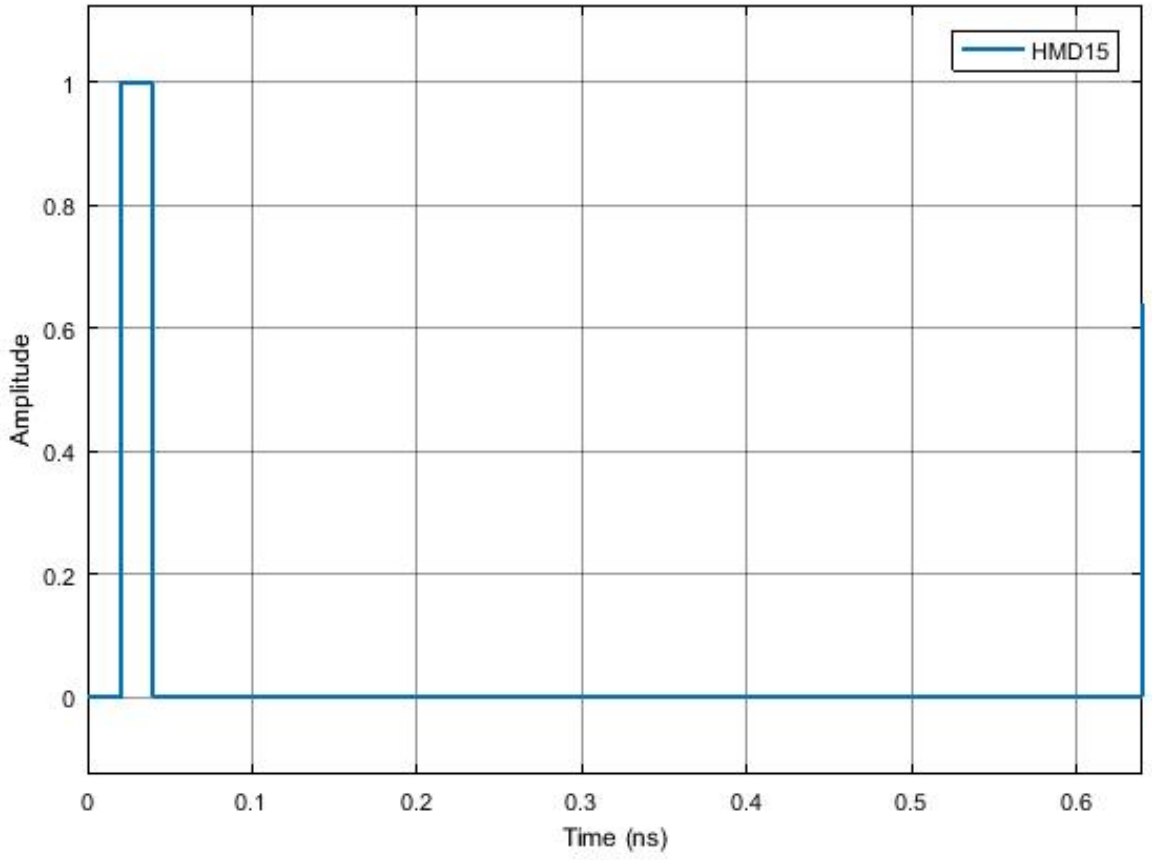

Figura 6.3: Example of the output signal of the decoder for a binary sequence 01. As expected it reproduces the initial bit stream

Sugestions for future improvement

6.4 Clock

This block doesn't accept any input signal. It outputs one signal that corresponds to a sequence of Dirac's delta functions with a user defined *period*.

Input Parameters

Parameter: period{ 0.0 };

Parameter: samplingPeriod{ 0.0 };

Methods

Clock()

Clock(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig)

void initialize(void)

bool runBlock(void)

void setClockPeriod(double per)

void setSamplingPeriod(double sPeriod)

Functional description

Input Signals

Number: 0

Output Signals

Number: 1

Type: Sequence of Dirac's delta functions. (TimeContinuousAmplitudeContinuousReal)

Examples

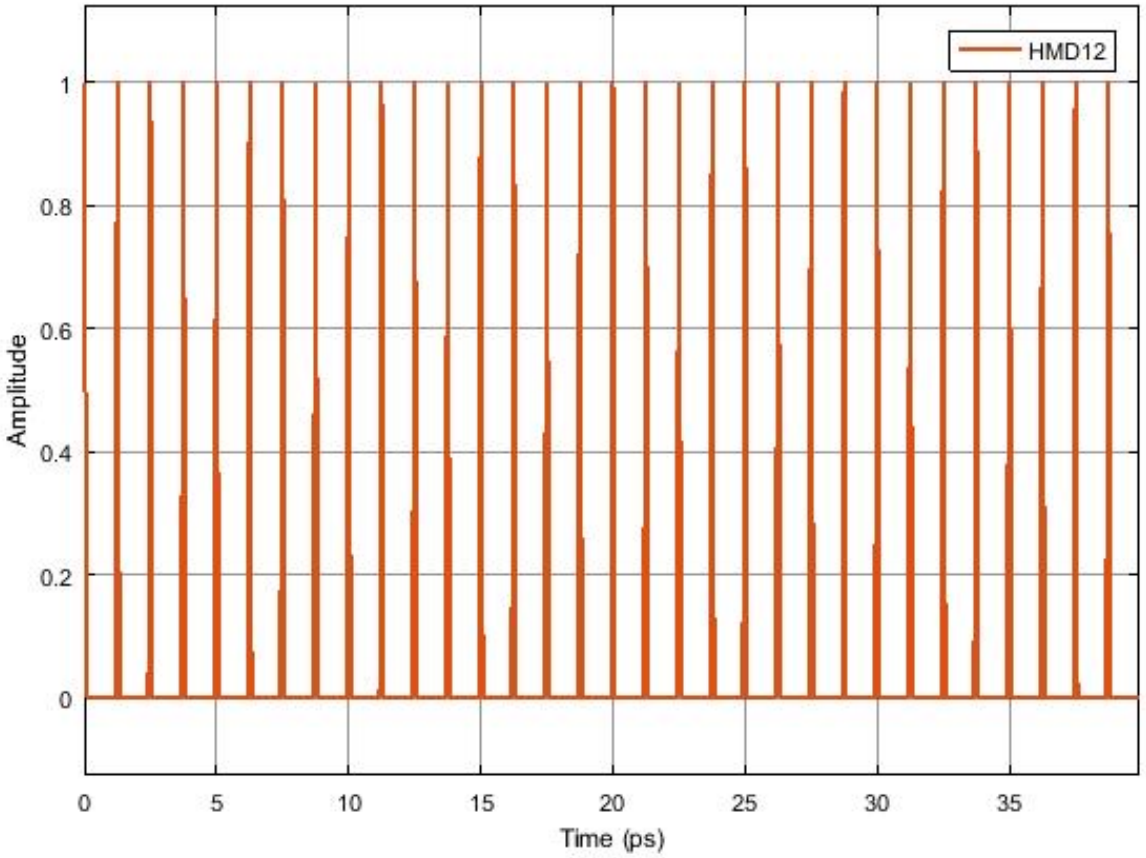

Figura 6.4: Example of the output signal of the clock

Sugestions for future improvement