

Homodyne receiver

April 13, 2017

1 Homodyne receiver

This block of code simulates the reception and demodulation of an optical signal (which is the input signal of the system) outputting a binary signal. A simplified schematic representation of this block is shown in figure 1.

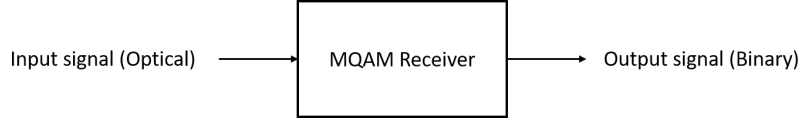


Figure 1: Basic configuration of the MQAM receiver

Functional description

This block accepts one optical input signal and outputs one binary signal that corresponds to the M-QAM demodulation of the input signal. It is a complex block (as it can be seen from figure 2) of code made up of several simpler blocks whose description can be found in the *lib* repository.

In can also be seen from figure 2 that there's an extra internal (generated inside the homodyne receiver block) input signal generated by the *Clock*. This block is used to provide the sampling frequency to the *Sampler*.

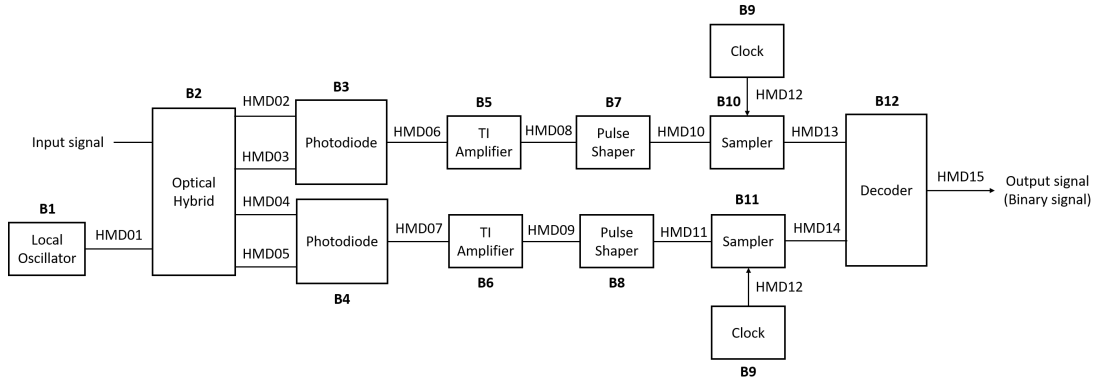


Figure 2: Schematic representation of the block homodyne receiver.

Input parameters

This block has some input parameters that can be manipulated by the user in order to change the basic configuration of the receiver. Each parameter has associated a function that allows for its change. In the following table (table 1) the input parameters and corresponding functions are summarized.

Input parameters	Function	Type	Accepted values
IQ amplitudes	setIqAmplitudes	Vector of coordinate points in the I-Q plane	Example for a 4-qam mapping: $\{ \{ 1.0, 1.0 \}, \{ -1.0, 1.0 \}, \{ -1.0, -1.0 \}, \{ 1.0, -1.0 \} \}$
Local oscillator power (in dBm)	setLocalOscillatorOpticalPower_dBm	double(t_real)	Any double greater than zero
Local oscillator phase	setLocalOscillatorPhase	double(t_real)	Any double greater than zero
Responsivity of the photodiodes	setResponsivity	double(t_real)	$\in [0,1]$
Amplification (of the TI amplifier)	setAmplification	double(t_real)	Positive real number
Noise amplitude (introduced by the TI amplifier)	setNoiseAmplitude	double(t_real)	Real number greater than zero
Samples to skip	setSamplesToSkip	int(t_integer)	
Save internal signals	setSaveInternalSignals	bool	True or False
Sampling period	setSamplingPeriod	double	Given by <i>symbolPeriod/samplesPerSymbol</i>

Table 1: List of input parameters of the block MQAM receiver

Methods

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

```

void setIqAmplitudes(vector<t_iqValues> iqAmplitudesValues)
vector<t_iqValues> const getIqAmplitudes(void)
void setLocalOscillatorSamplingPeriod(double sPeriod)
void setLocalOscillatorOpticalPower(double opticalPower)
void setLocalOscillatorOpticalPower_dBm(double opticalPower_dBm)
void setLocalOscillatorPhase(double lOscillatorPhase)
void setLocalOscillatorOpticalWavelength(double lOscillatorWavelength)
void setSamplingPeriod(double sPeriod)
void setResponsivity(t_real Responsivity)
void setAmplification(t_real Amplification)
void setNoiseAmplitude(t_real NoiseAmplitude)
void setImpulseResponseTimeLength(int impResponseTimeLength)
void setFilterType(PulseShaperFilter fType)
void setRollOffFactor(double rOffFactor)
void setClockPeriod(double per)
void setSamplesToSkip(int sToSkip)

```

Input Signals

Number: 1

Type: Optical signal

Output Signals

Number: 1

Type: Binary signal

Example

Sugestions for future improvement

2 Local Oscillator

This block simulates a local oscillator which can have shot noise or not. It produces one output complex signal and it doesn't accept input signals.

Input Parameters

- opticalPower{ 1e-3 }
- wavelength{ 1550e-9 }
- frequency{ SPEED_OF_LIGHT / wavelength }
- phase{ 0 }
- samplingPeriod{ 0.0 }
- shotNoise{ false }

Methods

LocalOscillator()

```
LocalOscillator(vector<Signal *> &InputSig, vector<Signal *> &OutputSig) :Block(InputSig, OutputSig){};
```

```
void initialize(void);
```

```
bool runBlock(void);
```

```
void setSamplingPeriod(double sPeriod);
```

```
void setOpticalPower(double oPower);
```

```
void setOpticalPower_dBm(double oPower_dBm);
```

```
void setWavelength(double wlength);
```

```
void setPhase(double lOscillatorPhase);
```

```
void setShotNoise(bool sNoise);
```

Functional description

This block generates a complex signal with a specified phase given by the input parameter *phase*.

It can have shot noise or not which corresponds to setting the *shotNoise* parameter to True or False, respectively. If there isn't shot noise the the output of this block is given by $0.5 * \sqrt{\text{OpticalPower}} * \text{ComplexSignal}$. If there's shot noise then a random gaussian distributed noise component is added to the *OpticalPower*.

Input Signals

Number: 0

Output Signals

Number: 1

Type: Optical signal

Examples

Suggestions for future improvement