MQAM transmitter

November 18, 2016

This block generates ~~one or two optical signals~~. A schematic representation of this block is shown in figure 1.

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 1.

Methods

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

void set(int opt);

void setMode(BinarySourceMode m) ~~{ B1.setMode(m); }~~;

BinarySourceMode const getMode(void) ~~{ return B1.getMode(); }~~;

void setProbabilityOfZero(double pZero) ~~{ B1.setProbabilityOfZero(pZero); }~~;

double const getProbabilityOfZero(void) ~~{ return B1.getProbabilityOfZero(); }~~;

void setBitStream(string bStream) { B1.setBitStream(bStream); };

string const getBitStream(void) { return B1.getBitStream(); };

void setNumberOfBits(long int nOfBits) { B1.setNumberOfBits(nOfBits); }

long int const getNumberOfBits(void) { return B1.getNumberOfBits(); }

void setPatternLength(int pLength) { B1.setPatternLength(pLength); }

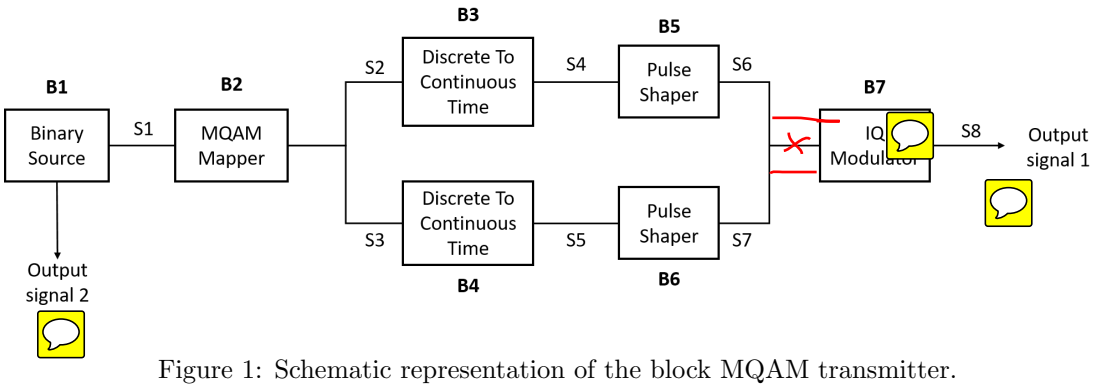

Figure 1: Schematic representation of the block MQAM transmitter.

Input parameters	**Function**	**Accepted values**
Mode	setMode()	PseudoRandom Random DeterministicAppendZeros DeterministicCyclic
Bit period	setBitPeriod()	double
Pattern length	setPatternLength()	int
Number of bits	setNumberOfBits()	long
Number of samples per symbol	setNumberOfSamplesPerSymbol()	int
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
Save internal signals	setSaveInternalSignals()	True/False

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

int const getPatternLength(void) { return B1.getPatternLength(); }

void setBitPeriod(double bPeriod) {B1.setBitPeriod(bPeriod);}; double const getBitPeriod(void) { return B1.getBitPeriod(); }

void setM(int mValue){B2.m = mValue;}; int const getM(void) { return B2.m; };

void setNumberOfSamplesPerSymbol(int n){ B3.setNumberOfSamplesPerSymbol(n); B4.setNumberOfSamplesPe
};

int const getNumberOfSamplesPerSymbol(void){ return B3.getNumberOfSamplesPerSymbol();
};

void setRollOffFactor(double rOffFactor){ B5.setRollOffFactor(rOffFactor); B6.setRollOffFactor(rOffFactor);
};

double const getRollOffFactor(void){ return B5.getRollOffFactor(); };

void setSeeBeginningOfImpulseResponse(bool sBeginningOfImpulseResponse){ B5.setSeeBeginningOfImpulseRes
B6.setSeeBeginningOfImpulseResponse(sBeginningOfImpulseResponse); };

double const getSeeBeginningOfImpulseResponse(void){ return B5.getSeeBeginningOfImpulseResponse();
};

Functional description

This block generates an optical signal (output signal 1 in figure 1). The binary signal generated in the internal block Binary Source (block B1 in figure 1) 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 1).

Output Signals

Number: 1 ~~or 2~~ optical and 1 binary

Type: Optical signal

Example

Sugestions for future improvement