The General Data Library

Version 1.07

Programmed by : Ammar Falih A. Hasan
2006-2007

This document contains detailed information about the classes implemented by the “GenData” C++ library.

The library consists of a header file “GenData.h” and a library file “GenData.lib”. To use the library inside your program, include the header to the program and add the path of the library to the list of paths for the compiler in use.

New Features:

- A major problem was solved regarding memory leaks using a program called Visual Leak Detector (VLD) for Visual C++. The problem was in the syntax of the function delete.
- The Use class was improved to include the similed effect on the interleavers.

Globals :

- "bool ADD_CORRECTION" : A constant Boolean indicating weather to add correction to the decoding algorithm, This directly influences the use of either Log-MAP or the Max-Log-MAP algorithms. Default = false indicating the use of Max-Log-MAP.
- "void add(bool a,bool b)" : Adds two Booleans logically.
- "enum Interleaver" : An enumeration that defines all the basic interleavers supported by this version of the library.
  The Interleavers supported are :
  - SAME : means the data left uninterleaved.
  - REVERSE : the data are permuted reversely(last bit first, first bit last and so on).
  - RANDOM : the data are permuted randomly.
  - ROWCOLUMN : data are set in rows and are retrieved by columns.
  - CIRCLEDIST: distributes the data into a circle where the data compete to be as far as possible from each other.

Class Convert :

This class is used to translate an integer number to its binary form expressed as an array of Boolean bits.

Public member variables:

- "bool* array" : array is a pointer to the binary digits expressed in Boolean started from right as the first bit and ending on the left as the last bit.
- "int state" : state is the integer form of the data.

Constructors:

- "Convert(int memorySize)" : constructs the object with the given array size, the array size determines the number of Boolean bits to be used.

Access functions:

- "Convert& operator=(const Convert &copy)" : default assignment function.
- "int getSize() const" : returns the number of bits(i.e. array size).

Helper functions:

- "bool* inttobool()" : returns array after calculating its translation from state.
- "bool* inttobool(int newState)" : assigns state and then returns its array bit translation.
- "int booltoint()" : returns state after calculating its translation from array.
- "int booltoint(bool *copy)" : assigns the values of copy to array and then returns its state translation.
- "void shiftRight()" : rotates the bits one step to the right and translates state.

Class SqMatrix :

A helper class that is used to build matrix related interleavers.

Public member variables:

- "int *inter" : a pointer to the matrix of the interleaver.

Constructors:

- "SqMatrix(int size)" : constructs a square matrix where row by column multiplication would result into the given size.

Access Functions:

- "int getRows()" : returns number of rows.
- "int getColumns()": returns number of columns.
- "int getSize()" : returns the given size.
- "void print()" : a diagnostic function used to print the matrix in the console terminal.

Helper functions:

- "void setRowColumn()" : builds a row column interleaver matrix from the internal square matrix.

Class InterMat :

Class InterMat is used to build the array of the interleaver, which is constructed by giving the desired interleaving technique option.

Constructors:

- "InterMat(Interleaver l, int blockSize, int firstVal=1)" : Constructs the array of interleaving matrix given the option of the desired interleaver and desired block size. The firstVal argument is used by the interleavers for special distributions options.
- "InterMat(Interleaver l, int N, int p, bool similed)" : Constructs a simile interleaver concatenating the given interleaver type. Make sure that N(the interleaver size) is divisible by p(the delay unit of the encoder). The delay unit p is given by $p=2^{v}-1$, where v is the No. of delay elements of the encoder. The similed option is just to indicate the similed operation, recommended to always be true.

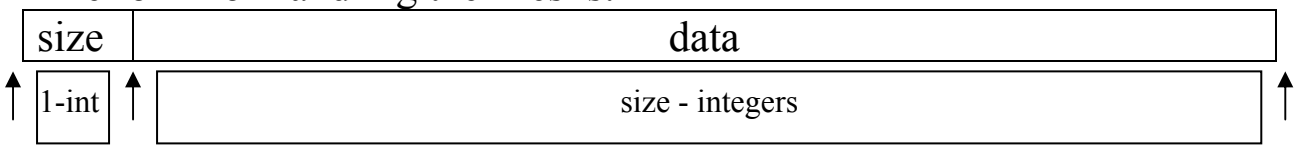
Access functions:

- "int* getMat() const" : returns the pointer to the matrix of interleaving.

Helper Functions:

- "void isConsistent()" : a diagnostic function that returns true if the built interleaver having the distribution No.s distributed from 1 to matrix size.

File Handling:
The form for handling the files is:



- "bool saveMat(const char* fileName)" : saves the matrix to a file with the specified name and returns true on success.
- "bool loadMat(const char* filename)" :loads the matrix from a file with the name specified and returns true on success.

Class StateTran :

Holds the transition table of states according to a given encoder. The encoder used is a 4-bit* memory RSC with transfer function(by default) in octal (g1,g2)=(23,33) or in binary (g1,g2)=(10011,11011) this is the optimum encoder which minimizes the bit error probability by maximizing the effective free distance. It is possible as well to supply user defined encoder.

Constructors:

- "StateTran(int s=0)" : constructs the default transition table of states and initializes the state to the given input or to the zero state by default.
- "StateTran(bool g1[], bool g2[])" : constructs a zero initialized transition table according to the given g1 and g2 arrays. Be careful to supply input arrays of size 5, also be aware that element g1[0] must always be true to complete the feedback eq.

Access functions:

- "void reset(int newState=0)" : assigns the given state to the object or reset it to the zero state by default.
- "int getState()" : returns the current state of the object.

Helper functions:

- "bool getParity(int input)" : returns the bit result of adding the forward transfer function on the given state.
- "int getNextState(int input)" : returns the next state according to the given input.
- "int getPreviousState(int input)" : returns the previous state if the given input is assumed.
- "double tailer(int state)" : returns a value to be used by a data stream to lead to the zero state if the given state is assumed.

---

* **Note that it was proven that the 4-bit memory encoder could easily be converted to 3 or 2 bit memory encoders simply by neglecting the last delay element from the encoder's equations.**

Class DataStr :

Used to instantiate a stream of data of a specific size, the data are represented as an array of type double values. Associated with the stream are group of functions that are collectively used to manipulate the data with the desired tasks.

Constructors:

- "DataStr(int s)" : Creates the stream with the specified size. The data within the stream are not initialized.
- "DataStr(const DataStr& dataStream)" : Default assignment constructor.
- "DataStr(DataStr* dataStream)" : Creates a data stream copying the contents of the given dataStream.

Access functions:

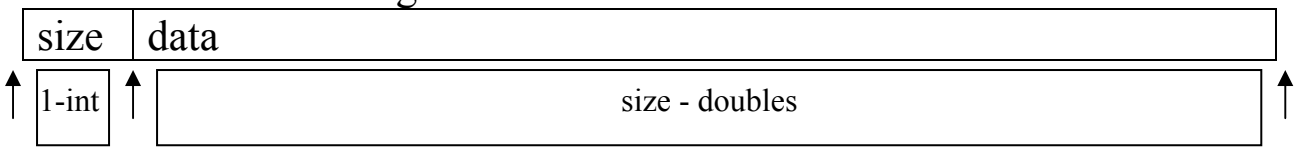
- "DataStr& operator = (const DataStr& dataStream)" : Default assignment function.
- "void setBits()" : Assigns random bits values to the stream.
- "void setBits(double choice)" : Assigns the given double to all the bits.
- "void setBits(double* copy)" : Assigns the bits the contents of the given copy array.
- "double* getBits()" : returns a pointer to the bits.
- "double* getCBits() const" : a constant version of the above function.
- "int getSize() const" : Returns the size of the stream.

Helper functions:

- "DataStr permute(const InterMat *mat, bool option)" : returns a permuted version of the given data stream according to the given interleaver matrix. option indicates interleave or deinterleave : true=interleave, false=deinterleave.
- "DataStr encode(StateTran *encoder)" : returns parity data stream from the given data according to the given encoder.

- "DataStr hard()" : returns hard limited version of the data. (+1.0,-1.0).
- "void tail(StateTran *encoder)" : tails the last bits of this data to lead to the zero state according to the given encoder in use and zero state initialization is assumed.
- "int compare(DataStr *original)" : compares the data with the given data stream and returns number of errors.
- "void halfPuncture(bool position)" : punctures the data for 1/2 rate simulation, this is used to omit the information received at the decoder as a result of puncturing that occurred at the encoder. The value of position determines the starting point of puncturing where: position=false-> start puncturing from first No., position=true-> start from second No.

File Handling:
The form for handling the files is:



- "bool saveData(const char* fileName)" : saves the data stream to a file with the name specified, and returns true on success.
- "bool loadData(const char* fileName)" : loads the data from a file with the name specified, and returns true on success.

Class BCJR :

A class used to build a decoding component object. The decoder is built using the optimized BCJR algorithm. The ADD_CORRECTION global variable directly affects this component. If ADD_CORRECTION is false the component will use MAX_LOG_MAP algorithm, otherwise a correction term will be added to the data making the component act as LOG_MAP algorithm.

Constructors:

- "BCJR(DataStr *message, DataStr *parity, double Lchannel, StateTran *encoder)" : Constructs and initializes a BCJR component with the given message, parity and encoder in use. The reliability of the channel value Lc given to the constructor is calculated from the equation : $Lc = 4SNR\_channel$ where SNR_channel is the signal-to-noise ratio of the channel*.

Access functions:

- "void setLeIn(DataStr *extrinsic)" : Sets the input extrinsic information to the decoder. Make sure to call this function before processing the decoder.
- "DataStr getLeOut()" : processes the overall data and calculates the output extrinsic information from the component and returns the result.
- "DataStr getFinal()" : adds(input message+input extrinsic+output extrinsic) as the final decoding result. Make sure to call both setLeIn and getLeOut before calling this function.

* **SNR_channel=pSNR, p=code rate, SNR=Eb/No, Eb=1, No=Noise floor of the channel.**

Class Channel :

Channel class is an implementation of the GenData library classes that simulates a channel having a collection of data streams and features noise addition along with a decoding process function that implements turbo decoding algorithm of individual components.

Constructors:

- "Channel(DataStr *msg, DataStr *par1, DataStr *par2, double SNR_channel, bool punctured=false)" : constructs three DataStr objects and initializes them using the given inputs. The given SNR_channel represents the SNR of the output of the encoder until it reaches the decoder side and the punctured bool is for describing the punctured status of the sent parities: false=not punctured, true = punctured.
- "Channel(DataStr *msg, double SNR_channel)" : constructs single DataStr object and initialize it using the given input. Using this constructor disables the coding specialized functions (i.e. decode).

Access functions:

- "DataStr getDat(int choice)" : returns a data stream object that represents a copy of one of the three internal DataStr objects. The given choice is the DataStr object to be returned : 0=message,1=par1 and 2=par2. for the single input case, the single data is always returned.
- "bool isPunctured()": returns the state of puncturing.

Helper functions:

- "void applyNoise()" : applies the noise of the channel to the data.
- "DataStr decode(int iterations, InterMat *interleaver, StateTran *encoder)" : decodes the three DataStr objects with a turbo decoding algorithm according to the given interleaver and decoder in use. The iterations argument Specify the number of iterations that the decoder will iterate.

Class Use :

Use class is an implementation of the GenData library, that is capable of simulating coded and uncoded experiments. The class supports saving the data by opening a file stream before processing. The file stream is closed by the destructor or manually by calling closeStream function.

Constructors:

- "Use(int packetSize)": initializes the uncoded experiment. Processing function will operate on the packet data without coding considerations.
- "Use(int interleaverSize, bool puncturedState, Interleaver intType=RANDOM, bool SIMILED=false, int p=15)": initializes the coded experiment. Processing function will simulate a turbo coding/decoding system initialized to a random interleaver with the given size. Puncturing mechanism is supported by the specified puncturedState: false=no puncturing is assumed, true=puncuring is done on the data. intType argument specifies the interleaver to be used, RANDOM is assumed by default, the new similed interleaver is supported along with the p parameter*.

Helper functions:

- "double processAndSave(double SNR_dB, int noOfPackets)": process and returns the Pe value at the given SNR in dB for the given No. of packets.

File Handling:
The form for handling the files is:

Pe	SNR_dB	Pe	SNR_dB	Pe	SNR_dB	Pe	SNR_dB	Pe	SNR_dB
↑	↑								
double	double								

* The p parameter is calculated from $\rho = 2^{v} - 1$ where $v =$ No. of memory elements in the encoder, always check that $N\%\rho = 0$ where N= size of the interleaver.

The size of the file is not saved, the implementing programs should either read all the file or use EOF capabilities as supported by their environments.

- "bool openStream(const char* filename)": creates a new file with the specified name. call this function before processing in order to save processed data. A warning will be printed if not called.
- "bool closeStream(const char* filename)": closes the file stream. This function is called by the destructor if the file is opened.