Chapter 11

EID: Error Insertion Device

An error insertion device (EID) is used to study the behaviour of codec over digital transmission systems and equipments under error conditions. This requires a model for the transmission channel, and an error generation algorithm. In the most general case, burst or random bit error generators are needed. In other cases, such as when evaluating mobile and wireless systems, random and bursty frame erasures are of importance.

The EID module implements the four functionalities: random and bursty bit errors, and for random frame erasure functionalities. These four models are based on a linear congruential sequence random number generator, and the bit error insertion and random frame erasure are based on a two-state channel model.

The burst frame erasure function requires a more elaborated model. For the specific application of wireless systems, a model based on Markov sequences is used. This is known within ITU as the Bellcore model [76, 77]. This model was used in the ITU-T 8 kbit/s speech coder selection tests, and has been incorporated in the STL since.

Upon development needs of scalable and embedded-variable bitrate codecs, a more elaborate EID simulation software was added in STL2009 release. This tools allows more complex error insertion simulations: layered application mode where upper layers are dependent on the error of lower layers, and (optional) individual application mode where errors are inserted individual to each layer.

11.1 Description of Algorithm

11.1.1 Simple Channel Model

The bit error insertion algorithm of the EID is based on a channel model where (binary) data bitstreams are to be transmitted, and is based on the discrete *Gilbert Elliott channel* (GEC) model, described in [78].

This model (see figure 11.1) has two states, Good (G) and Bad or Burst (B). Associated with these two states, there are four parameters (probabilities): two relating to the probability of remaining in state G or B, and two relating to the probability of transition from the current state to the other state (i.e., the occurrence of a binary digit transition, or error).

The probabilities associated with the channel states are P and Q, P being the probability of transition from state G to B, and Q the probability of transition from B to G. Hence, the probability of remaining in the same state is $(1-P)$ and $(1-Q)$ for states G and B, respectively. For a given state, there are probabilities that a change in a bit occur, and this is P_G for state G, and P_B for state B.

Therefore, the channel may be either in the good state G, where the mean bit error probability P_G is very low ($P_G \approx 0$), or in the bad state B, where the mean bit error probability P_B is rather high ($P_B \approx 0.5$).

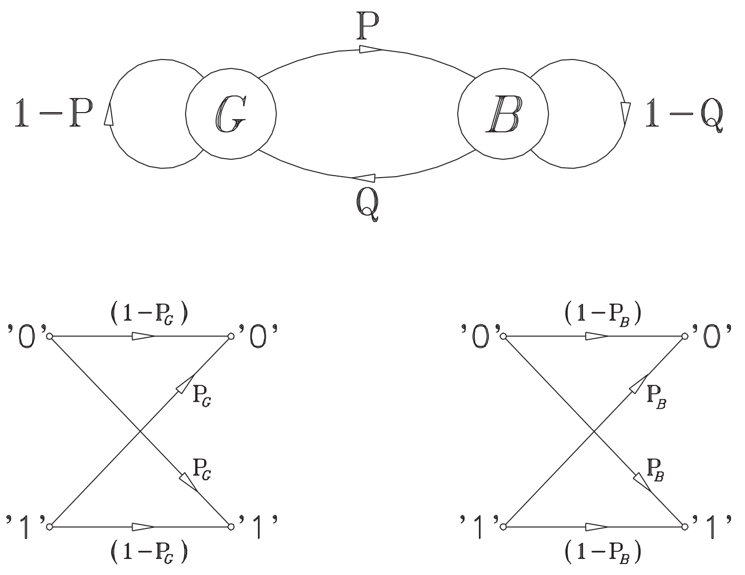

Figure 11.1: Gilbert Elliot Channel Model (GEC)

The mean bit error probability BER generated by this channel model is

$$BER = \frac{P}{1-\gamma} \cdot P_B + \frac{Q}{1-\gamma} \cdot P_G \tag{11.1}$$

where

$$\gamma = 1-(P+Q) \tag{11.2}$$

is a measure for the correlation of the bit errors, and consequently an indication of the burst or random characteristic of the channel. In this issue, $\gamma \approx 0$ implies a nearly random error channel, while $\gamma \approx 1$ implies a totally bursty channel. Please note that BER is reasonable only in the range $0 \le BER \le 0.5$.

For many applications, bit error sequences with a distinct mean bit error probability BER and a distinct bit error correlation γ are of interest. From equations (11.1) and (11.2) we get for the remaining parameters of the GEC for arbitrarily chosen values of $0 \le P_G < P_B \le 0.5$:

$$P = (1-\gamma) \cdot (1 - \frac{P_B - BER}{P_B - P_G}) \tag{11.3}$$

$$Q = (1-\gamma) \cdot \frac{P_B - BER}{P_B - P_G} \tag{11.4}$$

In the Error Insertion Device (EID) the special values $P_G = 0$ and $P_B = 0.5$ are chosen. This relates to the fact that in the good state no bit changes are expected, hence $P_G = 0$; now, for the bad state, the channel is supposed to be in a totally uncertain state, then $P_B = 0.5$. With this choice, equations (11.3) and (11.4) reduce to:

$$P = 2 \cdot (1 - \gamma) \cdot BER \tag{11.5}$$

$$Q = (1 - \gamma) \cdot (1 - 2 \cdot BER) \tag{11.6}$$

As an example, figure 11.2 shows the effect of γ on the auto-correlation of a bitstream, generated by the *GEC* (with $BER = 0.02$ in equations (11.5),(11.6)).

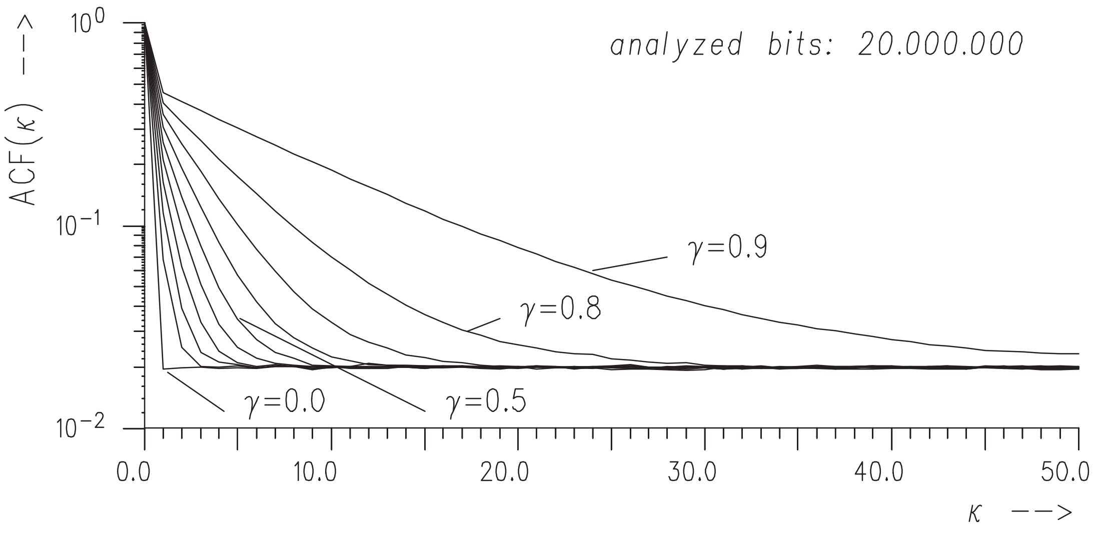

Figure 11.2: Bit Error Correlation for a Bit Error Rate of 2%

It can be seen that for $\gamma = 0$ the bit errors are statistically independent, because the autocorrelation sequence ACF(κ) has a peak (1.0) in $\kappa = 0$, and the remaining coefficients ACF(1),ACF(2),... oscillate around the selected bit error rate of 0.02 ($2 \cdot 10^{-2}$). For $\gamma = 0.5$, slightly bursty errors can be observed, when the initial terms of the correlation sequence build a transition region, and the remaining (higher) terms are around 0.02. Increasing γ towards unity, the correlation between the bit errors also increases, leading to totally bursty errors in the limit.

11.1.2 Bellcore Model

The following description has been based on [76]. The actual error sequence in a wireless environment will depend upon the carrier frequency, user speed, detection scheme, type of diversity employed, mean SNR, hand-off mechanism, etc. Though a model could be created using the above parameters, it would be impossible to apply because of the wide variance of the model parameters. It was found that a speech coder can be tested using error bursts generated by a much simpler model because the burst error performance of a speech coder can be characterized to a great extend by the way it reacts to short (5–20 ms), medium (30–60 ms) and long (over 80 ms) error bursts. If a coder performs well in the presence of a representative range of short, medium and long error bursts, it can be expected that the coder will behave well in an actual wireless communications environment, even though the actual radio channel generates error bursts with different statistics.

The Bellcore model, rather than modeling the wireless communications channel, models the occurrence of these short, medium, and long error bursts that would enable the characterization of the coder reaction to error bursts and to the error burst patterns it is expected it will encounter in practice.

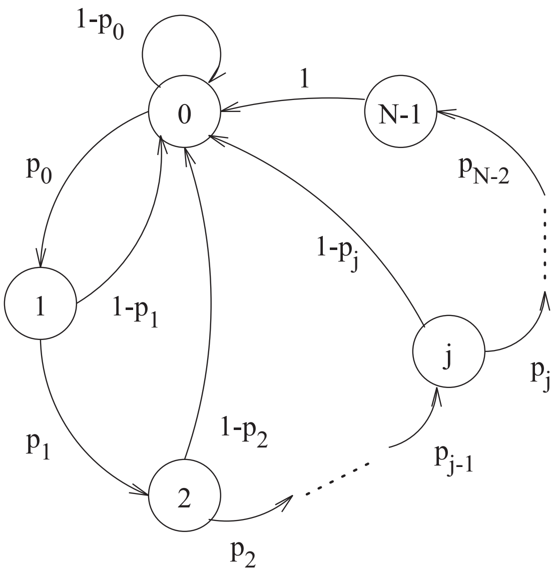

Figure 11.3: N-state Markov Model.

An N-state Markov model, as illustrated in figure 11.3, is used in the STL to generate frame erasure bursts. This model has to be adequate to test speech coders using short speech segments (6-8s). In this model, a transition from any state (0..N–1) to state 0 represents a frame received without errors, while a transition from state j–1 to state j indicates that j previous frames have been received in error. A transition from j back to 0 marks the end of an error burst of length j followed by a good frame.

The Markov model with N states for creating a bursty wireless communications channel is capable of erasing up to N–1 frames. The model generates both frames with correlated frame erasures and error-free frames. The value of N will depend on the frame duration of the speech coder under test and the maximum error burst length the coder expects to find in practice. The error statistics can be controlled by selecting the N–1 transition probabilities p_k, k=0..N–2. The probabilities will also determine the sequence of good and erased frames.

The steady state probabilities can be calculated by solving the state transition matrix or using numerical methods. If S_j denotes the steady state probability that the chain is in state j and p_j is the probability of transitioning from state j to $j+1$, the following relationships can be established:

$$S_{j+1} = p_j S_j \qquad\qquad 0 \le j \le N-2$$

$$S_0 = \sum_{j=0}^{N-1} S_j(1-p_j) \qquad\qquad (p_{N-1}=0)$$

The equations above can be solved since the probabilities should satisfy:

$$\sum_{j=0}^{N-1} p_j = 1$$

A frame erasure length of j can occur only if the chain first enters state j, and then transitions to state 0. The probability P_{fe} of this occur is

$$P_{fe} = S_j(1-p_j)$$

The probability of receiving a frame in error can be calculated as

$$P_e = \sum_{j=1}^{N-1} jP_{fe}(j)$$

and the probability of receiving an error-free frame is $1 - P_e$. It can be seen that the steady state probability of being in state 0, S_0, also gives the probability of receiving a frame without error, i.e.,

$$S_0 = 1 - \sum_{j=1}^{N-1} jP_{fe}(j)$$

The frame error distribution can be controlled by selecting the transition probabilities.

11.1.3 Error insertion for layered bitstreams

A layered scalable codec provides a multi-layer bitstream that can be modified by application or network entities:

- The core layer of the codec provides a minimum quality.
- Upper layers enable improving quality by increasing bitrate up to a maximum value.

One main feature of a scalable codec lies in the layering flexibility. The layers can be transported over different channels or over the same channel but with different priorities. Further the bitrate can be adjusted between minimal and maximal values by any network element in the communication chain.

To simulate the layering functionality a bitstream error application tool (eid-ev) was added for STL2009 release. This tool applies layer errors at desired levels. For each frame of an input bitstream, this tool performs the following operations:

1. Read and validate the input frame, (each input frame size needs to represent a valid layer boundary)
2. Read the frame error pattern files, (one error pattern file is read for each layer)
3. Apply errors by copying non-distorted layers to the output frame, and setting the bits of distorted layers to the softbit value zero.
4. Truncate output frame if consecutive higher layers are hit by errors.
5. Set the correct synchronization header of the output frame
6. Set the correct length of the output frame.
7. Provide statistics of error application across layers.

11.2 Implementation

The EID algorithm is written in C-source code can be found in the module `eid.c`, with prototypes in `eid.h`. This version evolved from previous C implementations developed by PKI[1], and was used in the Host Laboratory Sessions of ETSI's contest for the second generation of the GSM Digital Mobile Radio Systems, and in the Selection Phase of the ITU-T 8 kbit/s speech coder.

The random-number generator is based on a linear congruential technique, as described in [79]. The rule here is:

$$a_n = (69069 * a_{n-1} + 1) \bmod 2^{32},$$

which is converted (mapped) to a float number between 0 and 1.

Since the random number generator and the channel need their internal state to be saved, two state variable data structure types were defined for the EID module. The structure type called `SCD_EID` is applicable to the burst and random bit erasure functions, as well as to the random frame erasure function. The fields of this structure are:

seed	Seed for random number generator.
nstates	Number of states of the channel model (presently 2).
current_state	Index of current channel state.
ber	Pointer to array containing thresholds according to the bit error rate in each state.
usrber	User defined bit error rate.
usrgamma	User defined correlation factor.
matrix	Pointer to matrix containing thresholds according to the probabilities for changing from one state to another one.

For burst frame erasures only, a different state variable structure type called `BURST_EID` has been defined, whose fields are:

[1]Phillips Communications Industry.

seedptr	Memory for random number generator.
internal	Array with probabilities for each state of the Markov process.
index	Channel's current state.

The values of the fields shall not be altered and are not needed by user.

The random number generator always starts from the same point, if the user does not specify different initial seeds. In order to avoid this, the EID state variables should be saved at the end of the processing of a speech sequence, e.g. to a file by the user. This saving is not implemented by the EID module because this involves I/O to the computer file systems, and this would violate one of the UGST guidelines. Nevertheless, an example of this procedure is described in the demonstration programs that accompany this release of the EID. Therefore, users should keep in mind that, unless they save (e.g. to a file) the EID state at the end of the processing, identical error patterns will be produced, when the processing is re-started.

The EID routines for random bit errors are `BER_generator` and `BER_insertion`; for random frame erasures, `FER_generator_random` and `FER_module`; for burst frame erasures `FER_generator_burst`; and `open_eid`, `open_burst_eid` and `close_eid` for initialization (allocation) and release of EID state structures `SCD_EID` and `BURST_EID`. Their description can be found next. Besides these, there are other routines which are local (private) to the EID module, and therefore are not described.

11.2.1 Bitstream format

The EID module operates on *softbits* basis. Softbits are defined as a multi-level representation of the binary ("hard") bits '1' and '0' which are associated to probabilities of being in error. The softbit definition adopted in the ITU-T STL uses 16-bit words as representation of the hardbits `'1'` and `'0'`, where a hardbit `'1'` is represented by the softbit `0x0081` and a hardbit `'0'` is represented by the softbit `0x007F`. This means that 8 significant bits in a softbit is available for other utilities. When soft-decision is not used, the hard bit information can be derived directly from Bit 7 of the 16-bit softbit word. It should also be noted that values `0x0081` and `0x007f` are equally spaced from `0x0000` (in other words, `0x0000` is exactly the middle of the two-complement range for `0x81` and `0x7F`), such that a softbit representation `0x0000` represents *total uncertainty* of the true bit value.

Error patterns produced and used by the EID module use this softbit definition. Input and output data (i.e. signals which are affected by bit errors or frame erasures) also use softbits, but additionally have a so-called synchronization header.

The *synchronization header* is defined as two consecutive 16-bit words, the first one always being a synchronization (sync) word in the range `0x6B21` to `0x6B2F`, followed by the bitstream length word, a two-complement number indicating the number of softbits in the frame. The sync word `0x6B20` is reserved to indicate that a frame erasure happened. For example, for the RPE-LTP algorithm, which uses 260 bits per frame, the soft bitstream would have the format indicated in figure 11.4. It can be seen that each RPE-LTP frame will have 262 16-bit words, being one for the sync word (`0x6B21` in the example), one for the frame length word (whose value here is 260), followed by 260 softbit words (here corresponding to '1', '0', ..., '0', '0'). This combination of the synchronization header and a softbit "payload" is called the bitstream signal representation and is used in ITU-T Recommendation G.192 [80] to represent encoded signals between speech encoders, error-insertion devices, transmission channel models and speech decoders.

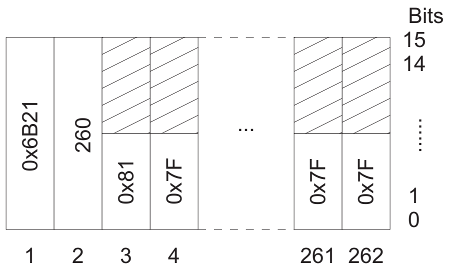

Figure 11.4: Soft bitstream format for the 13 kbit/s RPE-LTP algorithm, where 260 bits are transmitted per 20 ms transmission frame.

11.2.2 `open_eid`

Syntax:

```
#include "eid.h"
SCD_EID *open_eid (double ber, double gamma);
```

Prototype: eid.h

Description:

Allocate memory for EID struct, set up the transmission matrix according to the selected bit error rate, and initialize the seed for the random number generator. If the symbol `PORT_TEST` is defined at compilation time, then the seed will always be initialized to the same value; otherwise, the seed is initialized with the system time (in seconds). The former is used to test portability of the EID module, since identical patterns will be generated[2].

Variables:

ber	....	User desired bit error rate;
gamma	....	User desired burst factor;

Return value:

Returns a pointer to struct `SCD_EID`; if the initialization failed, returns a null pointer.

11.2.3 `open_burst_eid`

Syntax:

[2]Another way to force the EID to produce identical bit error patterns is to save the EID state variable (of type `SCD_EID`) e.g. to a file and, in the next call to the routine, initialize the state variable with the saved value.

```
#include "eid.h"
BURST_EID *open_burst_eid (long index);
```

Prototype: eid.h

Description:

Allocate memory for a state variable structure of type `BURST_EID` and setup the transmission matrix according the burst frame erasure rate (BFER) selected by *index*, and initialize the seed for the random number generator. If the symbol `PORT_TEST` is defined at compilation time, then the seed will always be initialized to the same value; otherwise, the seed is initialized with the system time, in seconds (see note in the description of `open_eid()`).

Variables:

index	Indicates one of the three BFER defined for the Bellcore model. If *index* is equal to 0, the 1% BFER is selected; if 1, BFER of 3% is used; or, if *index* equals 2, a 5% BFER is used;

Return value:

This function returns a pointer to a structure of type `BURST_EID`. If the initialization failed, it returns a null pointer.

11.2.4 `reset_burst_eid`

Syntax:

```
#include "eid.h"
void reset_burst_eid (BURST_EID *burst_eid);
```

Prototype: eid.h

Description:

Reset a `BURST_EID` structure previously initialized by a call to `open_burst_eid()`. By default, only counters are reset; if the symbol `RESET_SEED_AS_WELL` is defined at compilation time, the seed is also reset. However, this is not recommended.

Variables:

burst_eid	`BURST_EID` structure to be reset.

Return value:

None.

11.2.5 `close_eid`

Syntax:

```
#include "eid.h"
void close_eid (SCD_EID *EID);
```

Prototype: eid.h

Description:

Release the memory previously allocated by `open_eid()` for the specified EID structure.

Variables:

EID	.... EID state variables' structure to be released.

Return value:

None.

11.2.6 `BER_generator`

Syntax:

```
#include "eid.h"
double BER_generator (SCD_EID *EID, long lseg, short *EPbuff);
```

Prototype: eid.h

Description:

Generates a softbit error pattern according to the selected channel model present in *EID*. The introduction of the bit errors in the bitstream is done by the function `BER_insertion`. It should be noted that softbit error pattern buffers do not contain synchronization headers.

Variables:

EID	.... Structure with channel model.
lseg	.... Length of current frame.
EPbuff	.... Bit error pattern buffer with softbits.

Return value:

The bit error rate in the current frame is returned as a `double`.

11.2.7 `FER_generator_random`

Syntax:

```
#include "eid.h"
double FER_generator_random (SCD_EID *EID);
```

Prototype: eid.h

Description:

Decides whether a random frame erasure should happen for the current frame according to the state of the GEC model in the channel memory pointed by *EID*.

Variables:

EID	.... Structure with channel model.

Return value:

Returns a `double` value: 0 if the current frame should not be erased ("good frame") and 1 if the frame should be erased ("bad frame").

11.2.8 FER_generator_burst

Syntax:

```
#include "eid.h"
double FER_generator_burst (BURST_EID *EID);
```

Prototype: eid.h

Description:

Decides whether a burst frame erasure should happen for the current frame according to the state of the Bellcore model in the channel memory pointed by *EID*. It should be noted that in the long run, the overall burst frame erasure rate (BFER) may not be consistent with the BFER specified by the user (1%, 3%, or 5%). This is an inherent deficiency of the implemented model and the calling program is responsible for computing the overall BFER and monitoring whether this overall BFER is close enough to the desired BFER.

Variables:

EID	.... Structure with Bellcore model parameters.

Return value:

This function returns a `double` value: 0 if the current frame should not be erased ("good frame") and 1 if the frame should be erased ("bad frame").

11.2.9 BER_insertion

Syntax:

```
#include "eid.h"
void BER_insertion (long lseg, short *xbuff, short *ybuff, short
                    *error_pattern);
```

Prototype: eid.h

Description:

Disturbs an input bitstream *xbuff* according to the error pattern provided in *error_pattern*, saving the disturbed bitstream in the output buffer *ybuff*. The input and output bitstream are compliant to the bitstream format described before, i.e. are comprised of a synchronization header (sync word followed by a frame length word) and softbits representing the encoded bitstream. The sync and frame length words are always located in the offsets 0 and 1 of the array, respectively. The error pattern contains only softbits. The following summarizes the bit error insertion rules:

a) input signal (after synchronization header):

- hard bit `'0'` represented as `0x007F`;
- hard bit `'1'` represented as `0x0081`.

b) error pattern:

- the probability for undisturbed transmission has values in the range `0x0001..-0x007F`, being `0x0001` the lowest probability.

- the probability for disturbed transmission has values in the range `0x00FF..-0x0081`, being `0x00FF` the lowest probability.

c) output signal computation (does not affect the synchronization header, which is copied unmodified from the input buffer to the output buffer):

   For input '`1`' (`0x0081`):
   - if the error pattern is in the range `0x00FF..0x0081` (255..129), then the output will be `0x0001..0x007F` (1..127), respectively;
   - if the error pattern is in the range `0x0001..0x007F` (1..127), then the output will be `0x00FF..0x0081` (255..129), respectively.

   For input '`0`' (`0x007F`):
   - if the error pattern is in the range `0x00FF..0x0081` (255..129), then the output will be `0x00FF..0x0081` (255..129), respectively;
   - if the error pattern is in the range `0x0001..0x007F` (1..127), then the output will be `0x0001..0x007F` (1..127), respectively.

Variables:

lseg	Length of current frame (including synchronization header).
xbuff	Buffer with input bitstream of length *lseg*.
ybuff	Buffer with output bitstream of length *lseg*.
error_pattern	Buffer with error pattern (without synchronization header), of length *lseg–2*.

Return value:

None.

11.2.10 FER_module

Syntax:

```
#include "eid.h"
double FER_module (SCD_EID *EID, long lseg, short *xbuff, short
                    *ybuff);
```

Prototype: eid.h

Description:

Implementation of the frame erasure function based on the GEC model allowing a variable degree of burstiness (as specified by parameter `gamma` in the state variable structure of type `SCD_EID` pointed by *EID*). This function actually erases the current frame (as described below), as opposed to function `FER_generator_random()`, which only indicates whether the current frame should be erased.

- computes the "frame erasure pattern";
- erases all bits in one frame according the current state of the pattern generator.

The input (undisturbed) and output (disturbed) buffers have samples conforming to the bitstream representation description in Annex B of G.192. The input and output bitstream are compliant to the bitstream format described before, i.e. are comprised of a synchronization header (sync word followed by a frame length word) and softbits representing the encoded bitstream. The sync and frame length words are always located in the offsets 0 and 1 of the array, respectively. Should the frame be erased (depending on the frame erasure pattern), all softbits are set to `0x0000`, which corresponds to a total uncertainty about the true bit values.

In addition, the lower 4 bits of the sync word in the synchronization header are set to 0. This makes it easier for the succeeding software to detect an erased frame. The frame length word is copied unmodified to the output buffer.

Variables:

EID	Pointer to a state variable structure structure of type `SCD_EID`.
lseg	Length of current frame (including synchronisation header).
xbuff	Pointer to input bitstream. The synchronisation word (*xbuff[0]*) is processed, the frame length word (*xbuff[1]*) is not changed.
ybuff	Buffer with output bitstream.

Return value:

This function returns a `double` value: 1 if the current frame has been erased, and 0 otherwise.

11.3 Tests and portability

Portability may be checked by running the same speech file on a proven platform and on a test platform, for the whole range of input parameters. Results should be identical when the compilation is done with the symbol `PORT_TEST` properly defined and the channel states are set to a same value.

For the eid-ev program, please note that the 16 bit oriented G.192 files require correct byte swapping of inputs and outputs on big/little-endian machines. It checks for inconsistent sync headers and exits the program with a warning if incorrectly swapped synchronism headers are detected.

This routine had portability tested for VAX/VMS with VAX-C, MS-DOS with Turbo C v2.0, Sun-OS with Sun-C, and HPUX with gcc.

11.4 Examples

11.4.1 Description of demonstration programs

Number of programs are provided as demonstration programs for the EID module, eid-demo.c (version 3.2), eid8k.c (version 3.2), eid-xor.c (version 1.0), gen-patt.c (version 1.4), ep-stats.c (version 2.0), eid-int.c (version 1.0), eid-ev.c (version 1.0), gen_rate_profile.c (version 1.2).

Program `eiddemo.c` uses input and output file in the form of a serial bit stream conforming to the bitstream signal representation, as defined in Annex B of ITU-T G.192. This program will disturb the input bitstream with errors using the Gilbert Elliot Channel model for random or burst bit error insertion and for random frame erasures. The Bellcore model, which is used for burst frame erasures, is supported as a command line option, but not as default. It should be noted that this program uses function `FER_module()`, not function `FER_generator_random()`, for random frame erasures.

Program `eid8k.c` was developed during the standardization process of the ITU-T G.729 8 kbit/s speech codec for the task of producing bit error masks which would be used in the host laboratory hardware-implemented EID. For this program, input files are not generated, but only bit error pattern files. Consistent with the definition in the STL, error patterns do not have synchronization headers (sync word and frame length word), but only softbits representing disturbations of the channel. GEC and the Bellcore model are supported in this program. The output file format is, as was necessary for the G.729 work, different from a serial bitstream as defined in the STL because the softbits are saved as `char` (8-bit words) rather than as `short` (16-bit words). Conversion of this format to the STL 16-bit bitstream format can be accomplished using the unsupported program `ch2sh.c`.

It should be noted that both programs save in files the current state of the EID models under use and also try to read these state files at startup time (if not found, the programs create new ones, which are updated when the programs terminate).

Program `eid-xor.c` is an error-insertion program that simply XORs bits in a bitstream file (in one out of three formats: G.192, byte-oriented G.192, and compact) with error patterns (bit errors or frame erasures in one out of three formats) and saves the disturbed bitstream in a file. The error patterns need not have been produced by any of the EID models implemented in the STL, they only have to be in one of the three input formats. Since error patterns are either bit error EPs or frame erasure EPs, simultaneous bit errors and frame erasures are not allowed by `eid-xor.c`.

The program `gen-patt.c` is used to generate error patterns (EPs) using the EID models implemented in the STL (Gilbert and Bellcore models). The EPs will be either frame erasures or bit errors EPs, since the models in the STL do not support mixed frame erasure/bit error mode.

Program `ep-stats.c` examines an error pattern file (either bit error EPs or frame erasure EPs) and displays the actual BER/FER found in the EP and the distribution of number of consecutive errored bits or erased frames.

Program `eid-int.c` interpolates a frame erasure EP such that each synchronism word found in the EP is repeated a user-specified number of times. This is useful to align the frame erasures for codecs that have frame sizes that are an integer sub-multiple of each other (e.g. 10ms codecs and 20 ms codecs). In the latter example, the master EP will be the 20ms one, and the one generated by eid-int would be used for the 10ms codec.

Program `eid-ev.c` performs an error insertion for layered G.192 files. This program can be used to apply errors to individual layers in layered bitstreams such as G.729.1 or G.718.

Program `gen_rate_profile.c` generates random rate/layer switching file.

As a final note, it should be reinforced that the definition of the symbol `PORT_TEST` at compilation time **will** affect the operation of the programs as explained before. If this symbol is defined, functions `open_eid()` and `open_burst_eid()` will always start from the same seed. Therefore, the output of the programs will be the same, unless EID state files are available. When that symbol is not defined at compilation time, the programs will use the run-time library function `time()` to get the seed used in functions `open_eid()` and `open_burst_eid()`.

11.4.2 Using bit-error insertion routine

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "ugstdemo.h"
#include "eid.h"

#define OVERHEAD 2
#define LSEG 2048L                     /* Frame length is FIXED! */
#define SYNCword 0x6B21

void main(argc, argv)
  int             argc;
  char           *argv[];
{
  SCD_EID        *ber_st;                /* pointer to EID-structure */
  char            ifile[128], ofile[128];/* input/output file names */
  FILE           *ifilptr, *ofilptr;     /* input/output file pointer */
  static int      EOF_detected = 0;      /* Flag to mark END OF FILE */
  double          ber;                   /* bit error rate factor */
  double          gamma;                 /* burst factor */
  static double   dstbits = 0;           /* distorted bits count */
  static double   prcbits = 0;           /* processed bits count */
  short           err_pat[LSEG];         /* error pattern-buffer */
  short           inp[LSEG+OVERHEAD], out[LSEG+OVERHEAD]; /* bit-buffers */


  GET_PAR_S(1, "_File with input bitstream: ............... ", ifile);
  GET_PAR_S(2, "_File for disturbed bitstream: ............ ", ofile);
  GET_PAR_D(3, "_Bit error rate     (0.0 ... 0.50): ........ ", ber);
  GET_PAR_D(4, "_Burst factor       (0.0 ... 0.99): ........ ", gamma);

  /* Open input and output files */
  ifilptr = fopen(ifile, RB);
  ofilptr = fopen(ofile, WB);

  /* Allocate EID buffer for bit errors */
  ber_st = open_eid(ber, gamma);
  if (ber_st == (SCD_EID *) 0)
    QUIT(" Could not create EID for bit errors!\n", 1);

  /* Now process serial soft bitstream input file */
  while (fread(inp, sizeof(short), LSEG+OVERHEAD, ifilptr) == LSEG+OVERHEAD)
  {
    if (inp[0] == SYNCword && EOF_detected == 0)
    {
      /* Generate Error Pattern */
      dstbits += BER_generator(ber_st, LSEG, err_pat);

      /* Modify input bitstream according the stored error pattern */
      BER_insertion(LSEG+OVERHEAD, inp, out, err_pat);
      prcbits += (double) LSEG; /* count number of processed bits */

      /* Write disturbed bits to serial soft bitstream output file */
      fwrite(out, sizeof(short), LSEG+OVERHEAD, ofilptr);
    }
    else
      EOF_detected = 1;          /* the next SYNC-word is missed */
  }

  if (EOF_detected == 1)
    printf("   --- end of file detected (no SYNCword match) ---\n");
  printf("\n");

  /* Print message with measured bit error rate */
  if (prcbits > 0)
    printf("Measured BER: %f  (%ld of %ld bits distorted)\n",
           dstbits / prcbits, (long) dstbits, (long) prcbits);
}
```

11.4.3 Using frame erasure routine

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "ugstdemo.h"
#include "eid.h"

#define QUIT(m,code) {fprintf(stderr,m); exit((int)code);}
#define LSEG 2048L                        /* Frame length is FIXED! */
#define OVERHEAD 2
#define SYNCword 0x6B21

void main(argc, argv)
  int            argc;
  char          *argv[];
{
  SCD_EID       *FEReid;                  /* pointer to EID-structure */
  char           ifile[128], ofile[128];/* input/output file names */
  FILE          *ifilptr, *ofilptr;     /* input/output file pointer */
  static int     EOF_detected = 0;      /* Flag to mark END OF FILE */
  double         fer;                   /* frame erasure factor */
  double         gamma;                 /* burst factor */
  static double  ersfrms = 0;           /* total distorted frames */
  static double  prcfrms = 0;           /* number of processed frames */
  short          inp[LSEG+OVERHEAD], out[LSEG+OVERHEAD]; /* bit-buffers */


  GET_PAR_S(1, "_File with input bitstream: ................ ", ifile);
  GET_PAR_S(2, "_File for disturbed bitstream: ............ ", ofile);
  GET_PAR_D(3, "_Frame erasure rate (0.0 ... 0.50): ........ ", fer);
  GET_PAR_D(4, "_Burst factor       (0.0 ... 0.99): ........ ", gamma);

  /* Open input and output files */
  ifilptr = fopen(ifile, RB);
  ofilptr = fopen(ofile, WB);

  /* Allocate EID buffer for bit errors frame erasure */
  FEReid = open_eid(fer, gamma);
  if (FEReid == (SCD_EID *) 0)
    QUIT(" Could not create EID for frame erasure module\n", 1);

  /* Now process serial soft bitstream input file */
  while (fread(inp, sizeof(short), LSEG+OVERHEAD, ifilptr) == LSEG + OVERHEAD)
  {
    if (inp[0] == SYNCword && EOF_detected == 0)
    {
      /* Generate frame erasure */
      ersfrms += FER_module(FEReid, LSEG+OVERHEAD, inp, out);
      prcfrms++;                /* count number of processed frames */

      /* Write (erased) frames to serial soft bitstream output file */
      fwrite(out, sizeof(short), LSEG+OVERHEAD, ofilptr);
    }
    else
      EOF_detected = 1;          /* the next SYNC-word is missed */
  }

  if (EOF_detected == 1)
    printf("   --- end of file detected (no SYNCword match) ---\n");
  printf("\n");

  /* Print message with measured bit error rate */
  if (prcfrms > 0)
    printf("measured FER: %f  (%ld of %ld frames erased)\n",
           ersfrms / prcfrms, (long) ersfrms, (long) prcfrms);
}
```

11.4.4 Using layered bitstream error routine (eid-ev)

The demonstration program, `eid-ev.c` demonstrates the use of this module to apply errors in a scalable layered bit stream to simulate a flexible transport channel.

G.192 bitstreams (with sync header), G.192 byte-oriented bitstreams (with sync header) can be processed with this tool. (The byte oriented G.192 input stream is however limited to a maximum frame size of 255). The supported frame error patterns formats for the layers are the G.192 16-bit softbit format (without synchronism header) and the byte-oriented version of the G.192 format (also without synchronism header).

After processing the `eid-ev` program supplies statistics for the layer erasing operations performed for each layer. The statistics reported are the errors applied in the `eid-ev` error application ('erasing rate') as well as the total erasure rate ('total erasure rate').

The total erasure rate includes both input erasures and erasures performed in the `eid-ev` layer error application. When calculating the `eid-ev` 'erasing rate', applying an layer error to an already erased input layers is not counted as an layer erasure. The statistics can be suppressed by using the -q (quiet option).

To enable the simulation of cascaded network elements, the output of one eid-ev operation can be used as input to a subsequent eid-ev operation.

Error Application Modes

By default the tool operates in the layered error application mode, in this case errors that hit one layer will also lead to the erasure of higher layers. Optionally, the tool can operate individual error application mode, for this case it is assumed that layers are transported over individual uncorrelated channels. Thus for the individual error application mode an error in one layer does not result in an error in any other layer.

The layered error application mode is simulated by simply truncating the frame according to the error in the lowest layer for that frame. If the core (lowest layer) is hit by a layer error the frame is marked as a G192_FER frame, otherwise it is marked as a valid G192_SYNC frame with a shortened frame length.

The individual error application mode is simulated by setting the softbits for the individual layers with errors to the value zero. Subsequently the frame is then truncated if the highest layers have consecutive layer errors. If the frame has remaining layer errors (layers with all zero softbits) after attempting truncation, the frame is tagged as a G192_FER frame. If the frame has no remaining layer errors after truncation it is re-tagged as a valid G192_SYNC frame.

Note that a scalable decoder that wants to utilize the valid bits that may remain in the layers of bitstream after the individual error application operation will need to scan G192_FER tagged frames with a non-zero frame length for the layers with valid bits by discarding the layers with all zero bits.

Program options for EID-EV

Usage:
`eid-ev [options] in_bs e0 [e1, ..., eN] out_bs`
Where:

`in_bs`	input encoded speech bitstream file
`eX`	error pattern bitstream file (one for each layer)
`out_bs`	disturbed encoded speech bitstream file

options:

`-bs mode`	Mode for bitstream (g192 or byte)
`-ep mode`	Mode for error pattern file (g192 or byte)
`-ind`	Individual layer error application (individual intermediate layers may be erased)
`-layers`	Set layering setup in absolute bits default is "-layers 160,240,320,480,640"
`-q`	Quiet operation, skip statistics
`-h`	Displays this message
`-help`	Displays a complete instructive help message

Examples

For example, one frame of the input bitstream will comprise 640 bits for a 32 kbit/s codec operating a frame size of 20 ms, the core rate is 24 kbit/s and the highest layer starts at 24kbits/s (480 bits). To apply correlated layered errors to the two layers of the bitstream the following command line may be issued:

```
eid-ev -layers 480,640 inp f.01.g192 f.30.g192 outp.lay
```

where `f.01.g192` and `f.30.g192` are frame error pattern files with 1 % FER and 30 % FER respectively. And where the `inp` is a valid G192 bitstream file (containing frame sizes 0, 480, 640 only) and `outp.lay` is the resulting G192 bitstream file. If individual layer error application is desired the following command may be issued.

```
eid-ev -ind -layers 480,640 inp fer.01.g192 fer.30.g192 outp.ind
```

where `outp.ind` is the resulting G192 bitstream file after individual layer error application.

Default layering

The default layering is "-layers 160,240,320,480,640", with 20 ms frames this corresponds to the accumulated layer bitrates of 8, 12, 16, 24 and 32 kbit/s. This default layering requires five error pattern files, e.g.

```
eid-ev inp f.00.g192 f.02.g192 f.06.g192 f.10.g192 f.20.g192 outp
```

Note that a 0 kbit/s input frames are allowed, there are two possible 0-kbps frames:

- NoData frame (zero length and a G192_SYNC tag) (see Figure 11.6).
- A totally erased frame (zero length and a G192_FER tag), (see Figure 11.9).

EID-EV G.192 Input frame examples

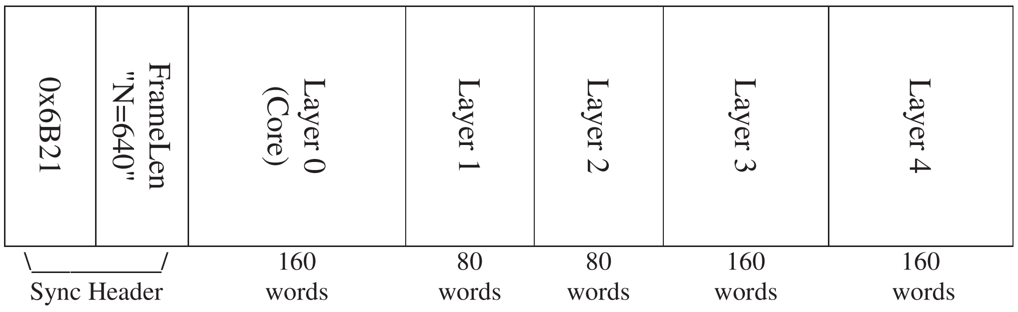

Figure 11.5: Example G.192 input frame with layering "-layers 160,240,320,480,640". This frame has a size of 640 bits and a G192_SYNC(0x6B21) header tag.

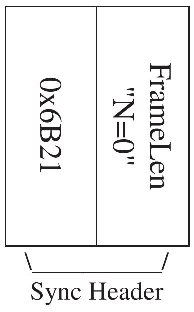

Figure 11.6: Example G.192 NoData frame, sync tag is G192_SYNC and frame length is zero. NoData frames may be used to simulate DTX (Discontinuous Transmission) operation.

EID-EV G.192 Output frame examples

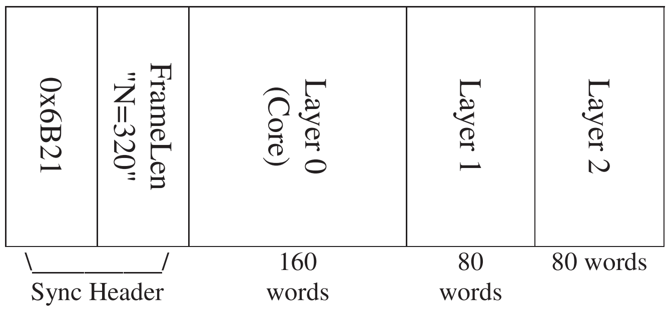

Figure 11.7: Example G.192 output frame when layer 3 is hit by a layer error in the layered error application mode.

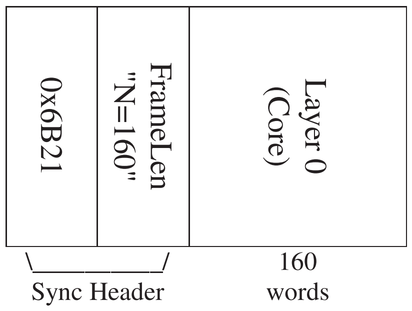

Figure 11.8: Example G.192 output frame when layer 1, layer 3 and layer 4 are hit by layer errors in the layered error application mode.

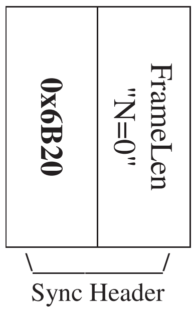

Figure 11.9: Example G.192 output frame when layers 0, 3 and 4 are hit by layer errors in the layered error application mode.

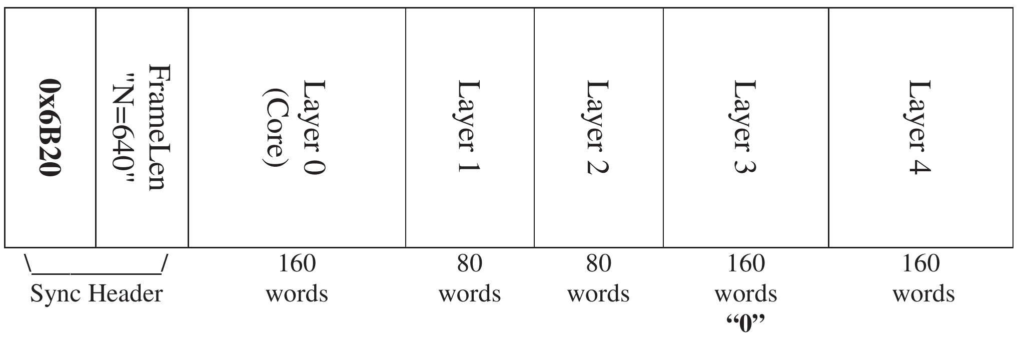

Figure 11.10: Example G.192 output frame when layer 3 is hit by layer errors in the individual error application mode. Note that the Sync header is changed to G192_FER(0x6B20) to indicate the presence of remaining layers with errors.

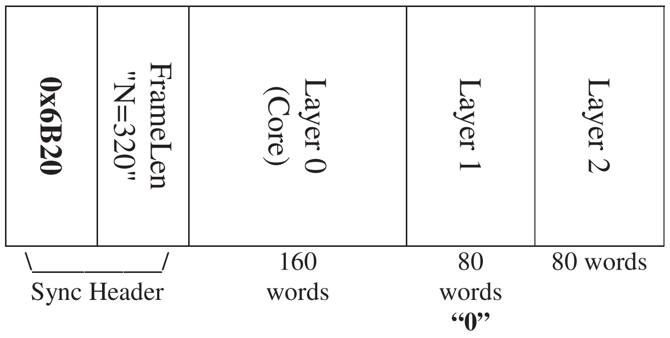

Figure 11.11: Example G.192 output frame when layers 1, 3 and 4 are hit by layer errors in the individual error application mode. Note that the frame is truncated and that the Sync header is changed to G192_FER(0x6B20) to indicate the presence of layers with remaining errors.

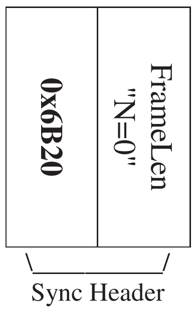

Figure 11.12: Example G.192 output frame when layers 0, 3 and 4 are hit by layer errors in the individual error application mode.