ADSL Toolbox for MATLAB

Anders Backstig

LiTH-ISY-EX-3168

December 12, 2001

Avdelning, Institution
Division, department

Department of Electrical Engineering
Kommunikationssystem

Datum
Date

2001-12-12

Språk Language	☐ Svenska/Swedish
	☒ Engelska/English
	☐ ____________

Rapporttyp Report: category	☐ Licentiatavhandling
	☒ Examensarbete
	☐ C-uppsats
	☐ D-uppsats
	☐ Övrig rapport
	☐ ____________

ISBN

ISRN

Serietitel och serienummer
Title of series, numbering

ISSN

LiTH-ISY-EX- 3168

URL för elektronisk version

Titel
Title

ADSL Toolbox for MATLAB

Författare
Author

Anders Backstig

Sammanfattning
Abstract

En ADSL verktygslåda för MATLAB har formgivits och programmerats. Den innehåller alla funktioner som behövs för en komplett överföring nedströms.

Verktygslådan består av en förevisningsmiljö som kan visa bittilldelning, signal/brus-förhållande samt de olika konstellationernas utseende. Produkten kan jämföra överföringshastigheten och bitfelssannolikheten för åtta olika kanalmodeller och två olika brusmodeller.

An Asymmetric Digital Subscriber Line (ADSL) toolbox for MATLAB has been designed and implemented. The toolbox contains all functions that are needed for a complete downstream transmission session.

The toolbox contains a demonstration view which can show the bit allocation, the signal to noise ratio and the different constellations in a graphical way. The application can compare the transmission rate and the bit error rate for eight different channel models and two different noise models.

Nyckelord
Keywords

ADSL, TEQ, FEQ, MATLAB, simulation, modem, communication

ADSL Toolbox for MATLAB

Examensarbete utfört i Kommunikationssystem

vid Tekniska Högskolan i Linköping

av

Anders Backstig

Reg nr: LiTH-ISY-EX-3168

Supervisor: Mikael Karlsson-Rudberg, Ericsson Microelectronics AB

Jonas Elbornsson, Linköping University

Examiner: Fredrik Gustafsson, Linköping University

Linköping, December 12, 2001.

Abstract

An Asymmetric Digital Subscriber Line (ADSL) toolbox for MATLAB has been designed and implemented. The toolbox contains all functions that are needed for a complete downstream training and transmission session.

The toolbox contains a demonstration view which can show the bit allocation, the signal to noise ratio and the different constellations in a graphical way. The application can compare the transmission rate and the bit error rate for eight different channel models and two different noise models.

Abbreviations

ADC	Analog Digital Converter
ADSL	Asymmetric Digital Subscriber Line
BER	Bit Error Rate
CAP	Carrierless Amplitude Phase
CRC	Cyclic Redundancy Check
CSA	Carrier Serving Area
DAC	Digital Analog Converter
DFT	Discrete Fourier Transform
DMT	Discrete Multitone
DSL	Digital Subscriber Line
FDM	Frequency Division Multiplexing
FEC	Forward Error Correction
FEQ	Frequency Domain Equalization
FEXT	Far-end crosstalk
FFT	Fast Fourier Transform
HDSL	High bit rate Digital Subscriber Line
IDFT	Inverse Discrete Fourier Transform
IDSL	ISDN Digital Subscriber Line
IFFT	Inverse Fast Fourier Transform
ISDN	Integrated Services Digital Network
ISI	Intersymbol Interference
LMS	Least Mean Square
MMSE	Minimum Mean Square Error
NEXT	Near-end crosstalk
POTS	Plain Old Telephone Services
QAM	Quadrature Amplitude Modulation
SDSL	Symmetrical Digital Subscriber Line
SNR	Signal Noise Ratio
TCM	Trellis Code Modulation
TEQ	Time Domain Equalization
VDSL	Very high speed Digital Subscriber Line

1 Introduction

When I was a little child we had no computer at home, and I had never tested Internet before I came to the university. Nowadays all children learn playing at the web when they are really young.

For a few years ago all data files where small and most people had no reason to transmit or receive large files at home. When the Internet get bigger and bigger with more things to use it is more important to have a fast connection.

During the last decades the modem rate has increased rapidly. The first transmission lines had a rate of a few bits/s but now they have kbits/s or Mbits/s. Because the thing that the user wants to transmit over the Internet, like music and movies, are very large it is important to increase the transmission rate.

The fastest connection is of course fiber but the big problem is that it requires a completely new infrastructure with high initial cost. A much cheaper and easier way is to use the common telephone lines that already exist in all houses. One of the techniques that uses the telephone line is Asymmetric Digital Subscriber Line (ADSL) which is the main topic of this thesis.

Ericsson Microelectronics AB produces chip for ADSL but have until now prioritized the analog part. To increase their knowledge of the digital parts of ADSL this thesis has been made.

1.1 Scope

The main task has been to write a toolbox in MATLAB which has all the blocks that appear in ADSL. First priority has been to write a complete toolbox and for this reason the toolbox has many not optimized algorithms instead of very few very good algorithms.

With the program it is possible to compare different transmissions for specific situations that appear in ADSL. Here it is possible to examine which rate that is possible for a known channel.

1.2 Outline

Section 2 contains a short background to ADSL. Section 3 describes the function for all different blocks. Section 4 presents the theory of filters and Section 5 the theory of noise. Section 6 illustrates the performance for the demo and the conclusions.

Appendix A explains all functions in the toolbox. Appendix B describes how the demo tool will be used in the best way.

2 Background

ADSL [1] is a high rate transmission technology. It is asymmetric which means that the transmission rates are different for the two directions. The slower direction, from the consumer is however faster for ADSL than for an analogue modem connection, which has a top speed on 56 kbit/s. It is even faster than Integrated Services Digital Network (ISDN) with a top speed on 128 kbit/s [2].

The idea with ADSL is to divide the bandwidth of the copper lines into a lot of frequencies. The different frequencies are called carriers. Each carrier transmits different parts of the data stream on the same time. This modulation technique is called Frequency Division Multiplexing (FDM). With ADSL it is possible to simultaneously transmit data and make a usual phone call. This is because the Plain Old Telephone Services (POTS) is transmitted over a reserved part of the frequency. In *Figure 1* a typical ADSL modem is shown.

FIGURE 1. This modem is one of the products from Ericsson in the ADSL area.

The maximum capacity for a ADSL transmission is 1 Mbps upstream and 8 Mbps downstream. These values require that the distance to the telephone station is less than 3.5 km. In average, the distance for swedish customers to the telephone station is 1.5 km [3], but a more interesting information is that approximately 75% [4] of the swedish telephone customers are located less than 3.5 km from the telephone station. Telia [5] is the biggest ADSL company in Sweden and today they deliver only a 0.5 Mbps downstream connection to their customers. The reason for this is that for this low speed they need not test the line performances, and they also have the possibility to sell faster connections more expensive. More information about different operators in Sweden can be found in [6].

2.1 History

ADSL was developed in 1987 by Bellcore Labs of Morristown, New Jersey. It was developed as a better version of ISDN with the purpose to use it for interactive television. ADSL has for a long time been known as the next great high-bandwidth technique because it can transmit as much as 8 Mbps of data over the already installed telephone cables.

In 1992 there were three different versions of ADSL [7]. Quadrature Amplitude Modulation (QAM) which is an old technique for transmitting digital signals on a copper wire was used by all the versions.

One of the methods used QAM in the normal way and the other two used it in modified forms. The other two was Carrierless Amplitude Phase (CAP) modulation which was developed by AT&T Paradyne and Discrete Multitone (DMT) which was developed by a group of Stanford University professors that ultimately created Amati Corp. The difference between CAP and DMT are described in [8].

To decide which of the methods that should be the standard a competition was held. Bellcore, the former research arm of the Bell companies and the place where ADSL has been developed arranged the contest that is known as the Bellcore Olympics. To the surprise of many, DMT won the competition and became the standard.

2.2 Other DSL family members

The most known other members of the DSL family is HDSL, IDSL, SDSL and VDSL [9][10].

HDSL (High bit rate Digital Subscriber Line) was developed as a faster variant of ISDN. HDSL is the oldest version of DSL. The rate is 1.544 Mbps over a regular copper wire when repeaters are not used. It requires two pairs of wires which is more expensive than other versions that use only one pair.

IDSL (ISDN DSL) can be used by customers who live up to 11,000 meter from the telephone station. That distance is not possible to reach with ADSL or SDSL. IDSL uses a symmetrical speed of 128 kbps

SDSL (Symmetrical Digital Subscriber Line) is a symmetrical service. Symmetrical means that the transmission rates are equal in both directions. Typical applications that use symmetrical transmissions are business operations. The speed is between 192 kbps and 1.1 Mbps.

VDSL (Very high speed Digital Subscriber Line) has the fastest transmission rate. For cables up to 300 meter, the transmission rate can be as high as 52 Mbps.

3 Block

In this chapter, each block in ADSL is explained [11][12][13][14][15][16][17][18][19]. A summary of ADSL is presented in *Figure 2*. In the first block the signals are put together in frames. These frames are transmitted over two different paths where the bit order is changed and coded.

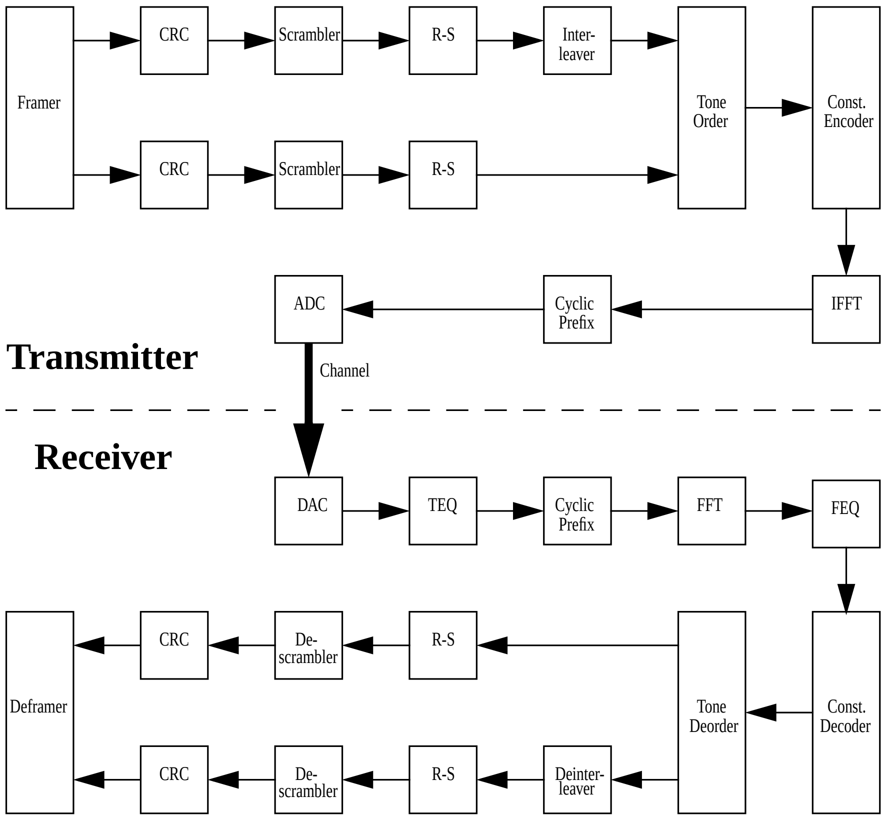


FIGURE 2. The ADSL block structure. All major parts such as framing, encoding, distributing in different frequencies, filters and decoding are described in their respective block.

The next main block is where the bits are divided to different frequencies. The modulation technique that is used in ADSL is called Discrete Multitone (DMT). The method divides the copperline in smaller frequency bands that are used independently.

ADSL can transmit data in both directions at the same time and on the same frequencies. To handle the overlapping frequencies an echo canceller is used to cancel the signal that is transmitted in the wrong direction. It is also possible to transmit the data on different frequencies. Because most of the consumer products are very cost sensitive, and it is cheaper to use a non-overlapping method, that is the most usual. For a downstream transmission 256 different carriers are used but for a upstream transmission only 32 carriers are used, see *Figure 3*.

The 256 carriers downstream and the 32 carriers upstream use the frequency interval from 0 Hz up to 1.104 MHz and 138 kHz, respectively. The number of carriers that are actually used are lower. One reason for that is that some frequencies can not be used. For example, the plain old telephone system (POTS) uses the lowest frequency interval.

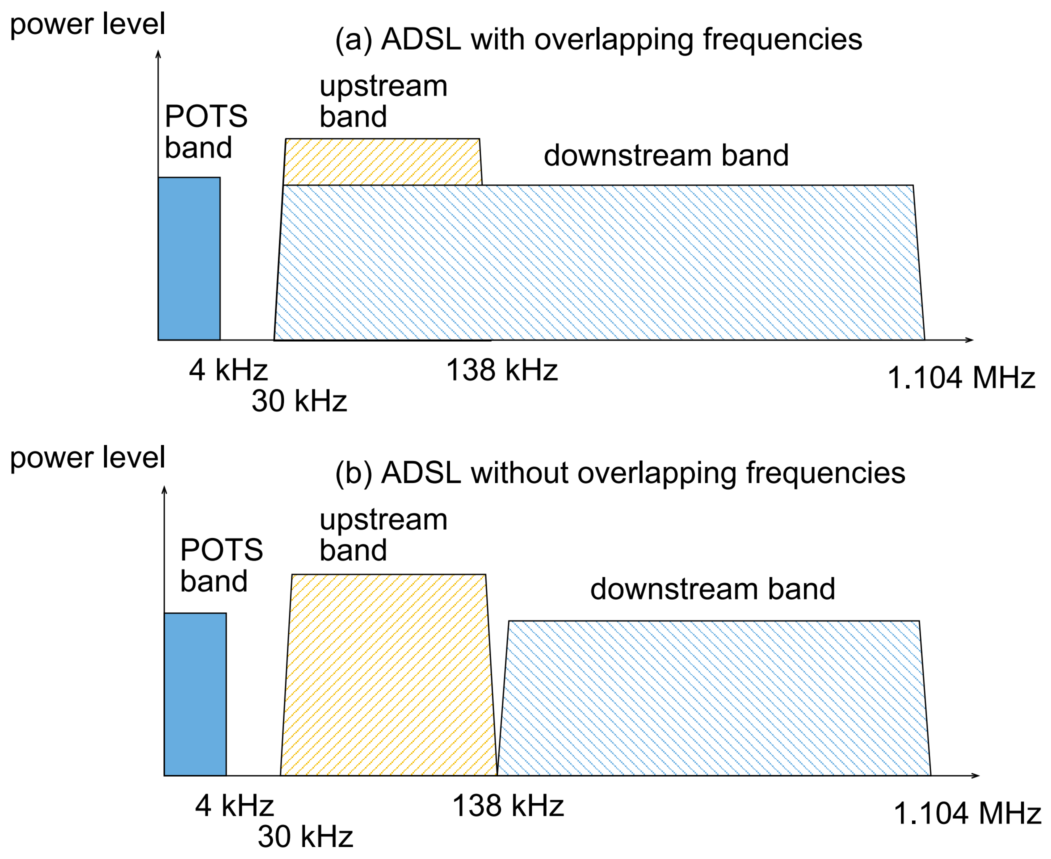


FIGURE 3. ADSL frequency plan.

3.1 Framer

The first block that appears in an ADSL circuit is the framing block. In this block a lot of different incoming data channels are put together. The different incoming data channels can be used one and one or many together. The framer puts the bits in frames and the frames in superframes. A superframe consists of 68 frames and one synchronization frame, see *Figure 4*. The length of a superframe is 17 ms. Because of the flexibility in ADSL the superframe length will vary from approximately 67 bytes to approximately 14,875 bytes.

Each frame of data corresponds to one ADSL symbol. The frame is divided in a fast path and an interleaved path. In the framer the cyclic redundancy check (CRC) is appended. CRC is a control of that the transmission is ok. Extra bits for control information and other overhead are also transmitted and controlled from this part of the system. After this block all blocks are working on one or more symbols.

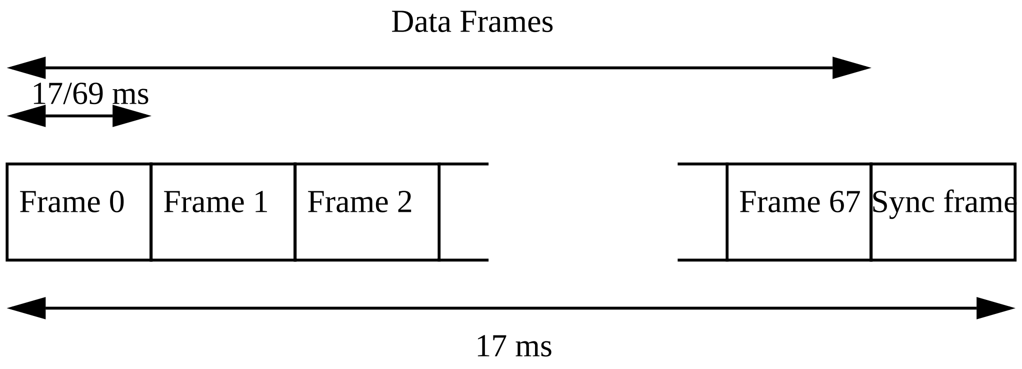


FIGURE 4. The structure of an ADSL superframe. Each superframe consists of 68 data frames and one sync frame.

The data from the framer can go through two different paths. Both paths pass through scramble and forward error correction but the interleaved path also passes through an interleaver.

Which way the superframe will go depends on which logical channel the data come from. Different applications transmit their data in different logical channels. In this case applications such as communication between two persons can be transmitted on the fast path. Other applications such as movies that are not sensitive for an initial delay can be transmitted the interleaved way.

The framer is not so important for the function of the ADSL algorithms because it always produces some symbols that are not used for any information before they are unpacked in the deframer in the receiver. This function is not implemented.

3.2 Cyclic redundancy check

Cyclic redundancy check (CRC) is a short sequence that is evaluated for each superframe. Two CRC results are transmitted. One from each of the fast and the interleaved path. In ADSL the CRC-8 with the polynomial $G(x) = 1 + x^2 + x^3 + x^4 + x^8$ is used. That means that eight bits are appended to each superframe. These bits are in the receiver used to decide if the transmission is correct. CRC can only detect errors and it can accordingly not correct errors.

The evaluation is made in that way that the incoming symbol is divided with the polynomial G(x). The rest that then appears is the result from the CRC operation. To control the result, the received symbol is evaluated in exactly the same way as the transmitted one. These two answers are compared and if they are equal the transmission is said to be ok.

Because it is only eight bits in the answer it is of course possible to have the same answer also for other incoming sequences. The probability for this is only $\frac{1}{2^8} \approx 0.39\,\%$, so the CRC code detects 99.6% of all possible errors. CRC is a fast method that is very easy to compute. It gives a delay for only a few clock cycles and it is well known how it can be implemented efficiently in hardware.

3.3 Scrambler

The scrambler in ADSL is applied to a serial data stream. Frames and synchronization are not important for this function to scramble and also not for descramble. The function that is shown in *Figure 5* has the formula $y = x[n] \oplus x[n-18] \oplus x[n-23]$.

The idea with scrambling is to mix up the bits to avoid very long bursts of ones or zeros. A better mix gives a better performance. Another reason to have scrambling is that it ensures that the spectrum from the transmitted bits are more like noise.

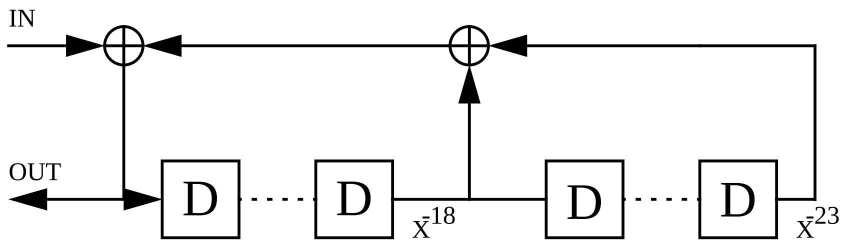


FIGURE 5. The scrambler mixes up the bits to avoid very long bursts of ones or zeros.

3.4 Reed-Solomon coding

The idea with this block is to append redundancy to the transmitted bits that can be used to detect and correct bit errors that can occur during the transmission.

After the data have been scrambled it is passed through the forward error correction (FEC) block. This block uses the Reed-Solomon algorithm to append R parity bytes to each DMT symbol. R is an even number between 0 and 16 that is set during the initialization. A single code word is sent for each data frame for the fast path, but for the interleaved path the code word can be a mix from S symbols, where S is 1, 2, 4, 8 or 16. Even S is set during the initialization process. The number of parity bytes is approximately 10 times fewer than the number of message bytes, M. An example with S=4 can be seen in *Figure 6*.

The Reed-Solomon code is working in the galois field arithmetic, GF(256). That means that it is a number system with 256 different numbers in which all the common math rules are defined, but of course in a different way than in the decimal system. The Reed-Solomon code produces code words with a known number of bytes. The number of bytes depends on different things that are decided in the initialization. For example, with 224 data bytes and 16 check bytes it can correct eight errors, that is one half of the number of check bytes.

The easiest way to understand this would be if a symbol only contained one code word. For a low rate it would give short and also ineffective code words and with a high rate it would be too long code words. Because of this problem it is a design parameter that decide how many symbols that will be in each code word. It is called S and have the values 1, 2, 4 and so on. The solution for S is often made so each code word has approximately 200 bytes. In GF(256) the codeword must have less than 255 bytes.

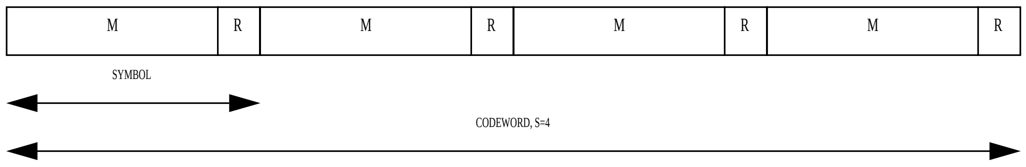


FIGURE 6. Each Reed-Solomon code word depends on message bits, M and parity bits, R.

3.5 Interleaver

The idea with the interleaver is to spread bursts of errors that can appear to many different code words. The interleaver has two important parameters, the interleave depth, D and the number of bytes per code word, N. D is always a power of two and is decided during the initialization.

An example with N=3 and D=4 can be seen in *Figure 7*. Each of the N bytes (B_1, B_2, B_3) is delayed by an amount that varies linearly with the byte index. More precisely, byte B_i is delayed by $(D-1) \cdot (i-1)$ bytes in the interleaver. With these rules the output bytes always occupy distinct slots when N is odd. When N is even, a dummy byte will be added at the beginning of the interleaver. The dummy byte is later removed from the output of the deinterleaver.

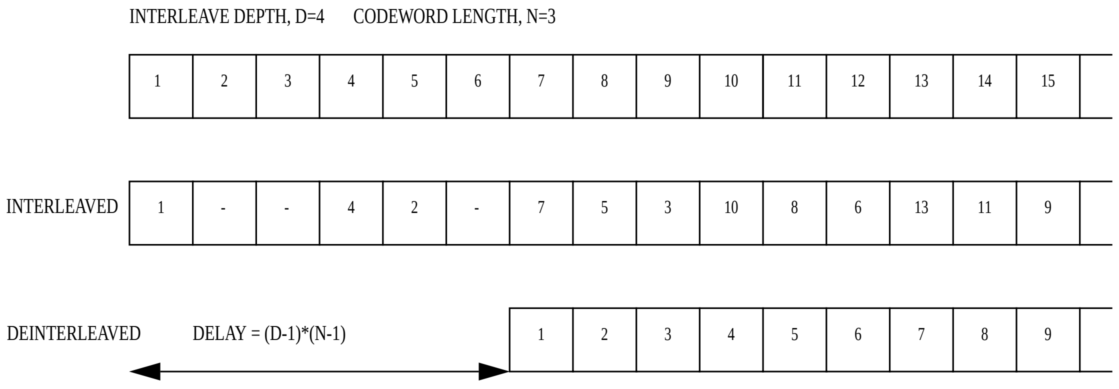


FIGURE 7. The interleaver changes the order of the bits. This spreads bursts of errors to many different code words.

3.6 Tone Order

The tone order assigns bits to different carriers. The most important thing to know is how many bits that can be assigned for each symbol. That information is stored in the bit-table. The bit-table is calculated in the initialization process and for each symbol the tone order gets so many bits from the previous block.

The bit allocation function that produces the bit-table can be explained in an easy way. During a test transmission period some training symbols are transmitted. The receiver modem calculates the signal to noise ratio (SNR). From the SNR it is possible to calculate the bit-table and that value is transmitted back to the transmitter before the training time has expired. A SNR-value and the corresponding bit-table is presented in *Figure 8*.

SNR is the ratio between signal and noise and this is the main parameter to decide how much data that can be transmitted over the channel. SNR is calculated through the formula in Equation 1, where x is the transmitted and y the received signal.

$$SNR = 20 \cdot \log\left(\left|\frac{x}{(x-y)}\right|\right) \qquad \text{(EQ 1)}$$

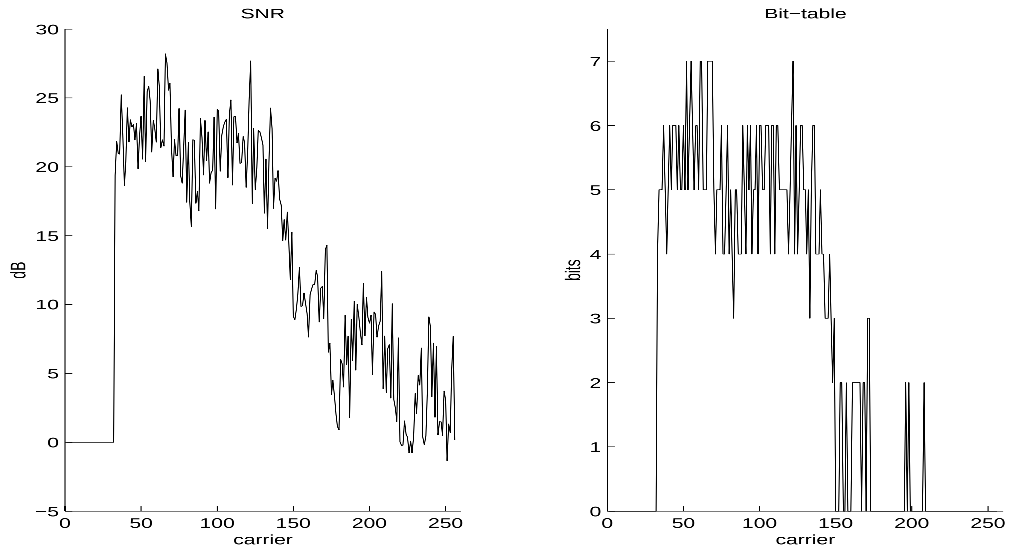


FIGURE 8. The left plot shows SNR and the right plot a bit allocation for this SNR. The bit table is in this case correlated to the SNR with a simple bit allocation algorithm.

The bit-table can be calculated with several different methods but all give more bits to the carriers with the highest SNR. A signal with the double power has 3 dB higher value. An increase with one bit doubles the number of possible constellations. The distance between two successive constellation points then approximately is half the length. A good approximation is to give 3 dB for each new bit. To increase the performance it is also common to have a margin down to zero. In the toolbox, an algorithm that calculate the bit-table, b as: $b = \left\lceil \frac{snr-6}{3} \right\rceil$, there 6 is the margin in dB, are used.

The tone order function has two incoming paths, one fast and one interleaved that are multiplexed together to an incoming bit stream, see *Figure 9*. When the bits are assigned the bits from these two paths append in different areas.

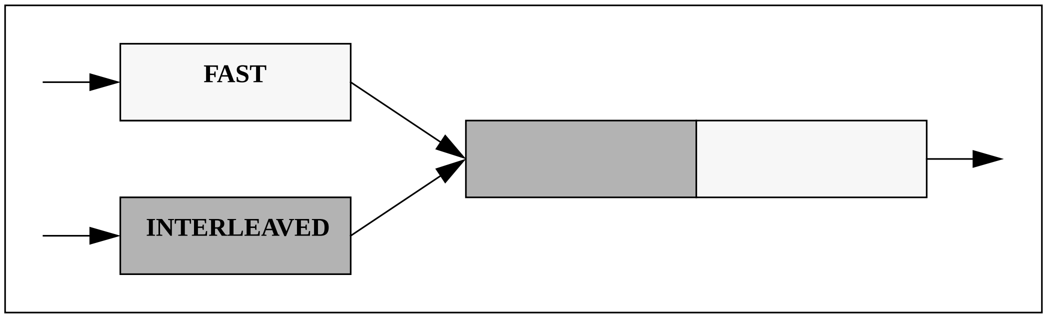


FIGURE 9. The fast and interleaved bitstreams are merged together and are thereafter transmitted to the tone order.

The algorithm for tone ordering starts with a bit stream with the *N* bits that will be transmitted. *N* is the number of bits that are possible to transmit. In these *N* bits the fast bits come first and then the interleaved bits. First allocate bits to the tones that can transmit the smallest number of bits and after that in increasing order. The result is that the fast bits are transmitted over the carriers with the fewest bits, see *Figure 10*.

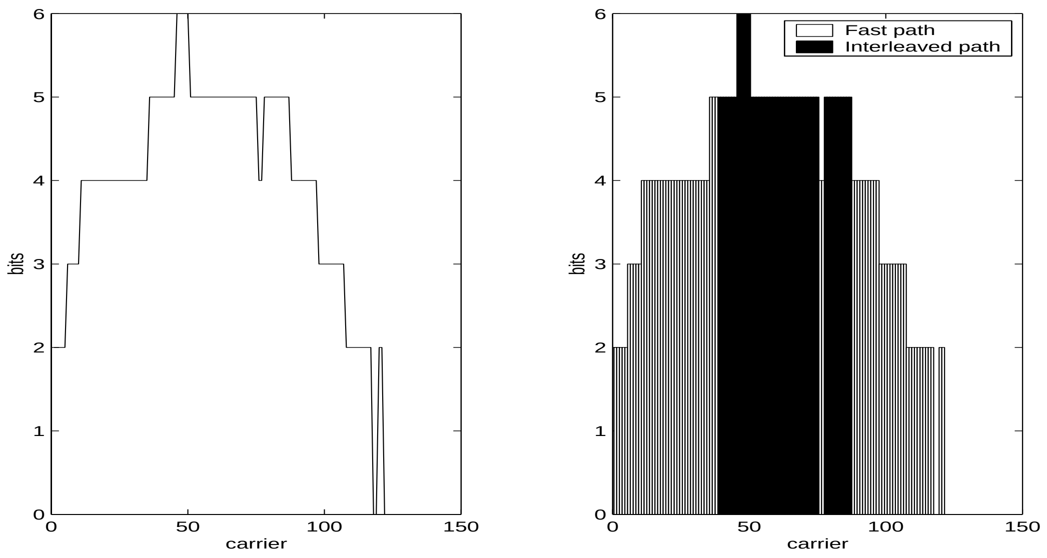


FIGURE 10. The bit allocation algorithm puts the fast bits on the channels with fewest bits.

The tone-ordering algorithm was designed to minimize errors. An impulse that comes during one symbol will cause the same noises in all carriers. Then the carriers that transmit a lot of bits are more likely to make a decoding error. In general, an impulse will affect the interleaved path much more than the fast path.

The bit errors from the interleaved part will be split up and because they probably come from different code words, they are truly possible to correct.

3.7 Constellation encoding

The constellation encoding is a process that is independent for each tone. The number of points for each tone depends on the number of bits that are assigned to each tone. The smallest tones with two bits can give four different constellation points and the biggest with 15 bits can give 32768 different points.

b bits give 2^b constellation points. If b is an even number it is just to take the constellations point from b-2 and change every point to four new points. If b is odd it is more difficult but the idea is to never have the constellation points further away than necessary. In *Figure 11* the two smallest even constellations are presented and in *Figure 12* the two smallest odd constellations are presented. The signal level is normalized to the value 0.5.

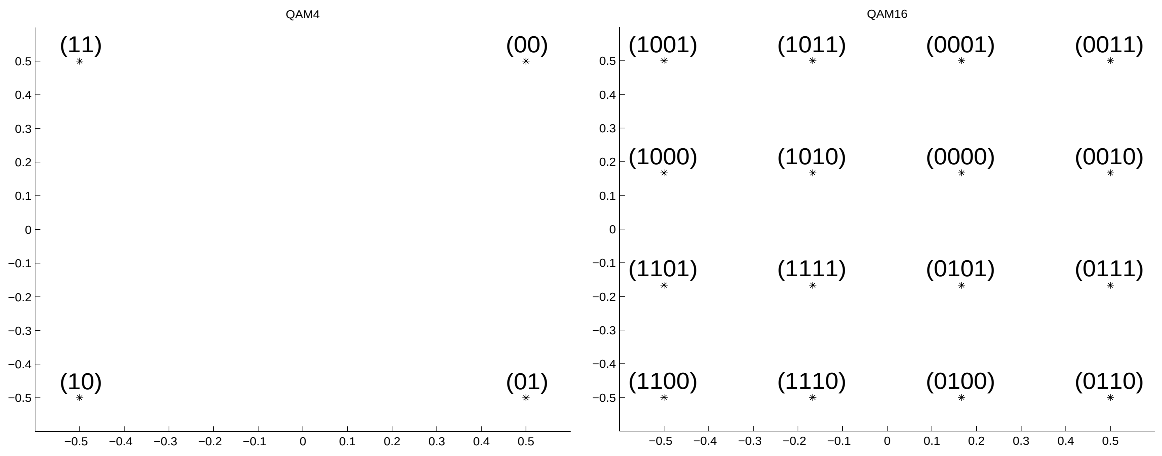


FIGURE 11. The constellation points for even number of bits. QAM4 (left) and QAM16 (right).

The constellation encoder delivers a complex value for each tone. These values are later transformed in a Fast Fourier Transform (FFT).

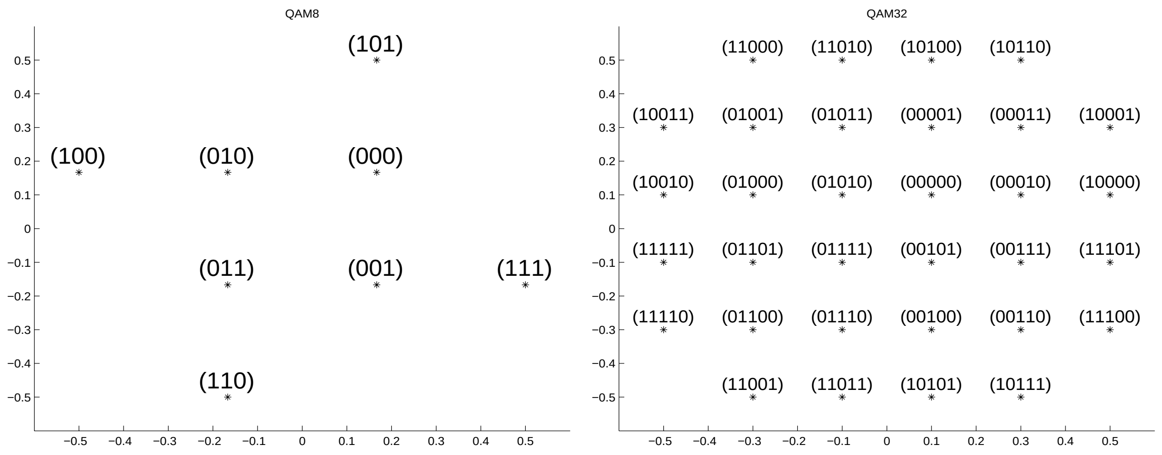


FIGURE 12. The constellation points for odd number of bits. QAM8 (left) and QAM32 (right).

3.8 Viterbi

Trellis code modulation (TCM) also known as Viterbi coding is one of the coding methods that is used to increase the redundancy in an ADSL system. Contradictory to other coding procedures this method is not required to be used in the modem. It is up to the system to decide if this method is used or not.

The function takes only a few bits when it is calculating the code, see *Figure 13*. It uses the least significant bits and from these it produces the code. The reason for this is that this coding is used to prevent errors that appear when one constellation point is detected as one of its neighbors. That is the same as if one of the least significant bits are corrupted. If the receiver would detect a constellation point that is further away this coding has no effect, but these errors are not so common.

TCM is produced in the same process as the modulation of the constellation points. The coding that is used is called Wei code. More about the trellis coding can be found in [20] and [21]. The Viterbi decoding is not implemented in the toolbox and for this reason the encoding is not in use.

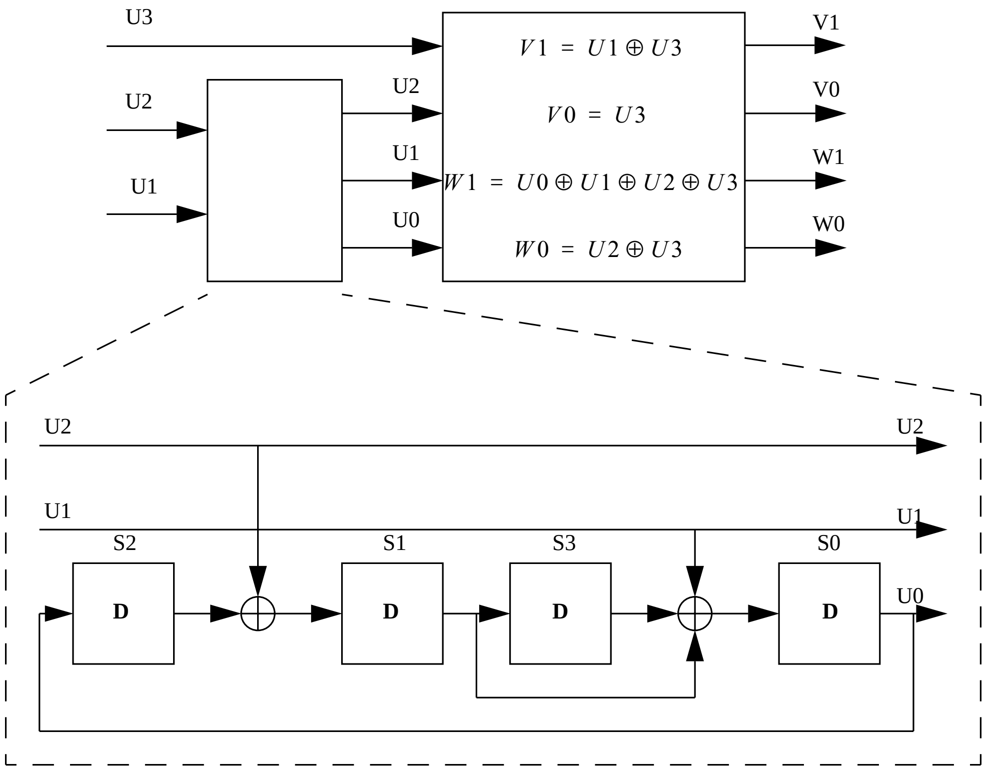


FIGURE 13. The Viterbi encoder used in ADSL produces a code from the three lowest bits.

3.9 FFT

The QAM-constellations are mapped on carriers using the inverse discrete fourier transform (IDFT). This transform and the symmetry in the constellation make the result real-valued. The transformation to real values makes it possible to convert the digital signal to an analog signal in a later step. In the receiver a discrete fourier transform (DFT) is used for demodulating the signal. More about FFT can be read in [22].

3.10 Cyclic prefix

A cyclic prefix is appended to the fourier transformed symbol. The idea with this block is to make the equalization procedure easier and make it possible to destroy most of the components from the ISI. This operation that for downstream transmission increases the number of bits from 512 to 544, also decreases the speed of the transmission. After the equalization the cyclic prefix is removed. The operation is explained in *Figure 14*.

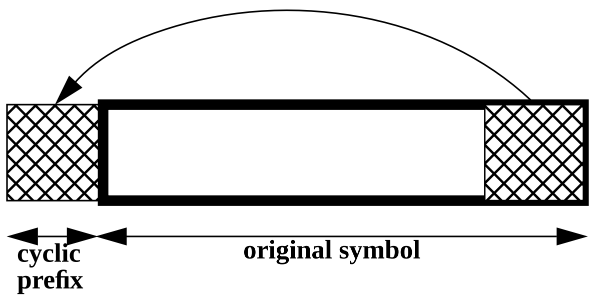


FIGURE 14. A cyclic prefix is a redundant information used to mitigate the TEQ.

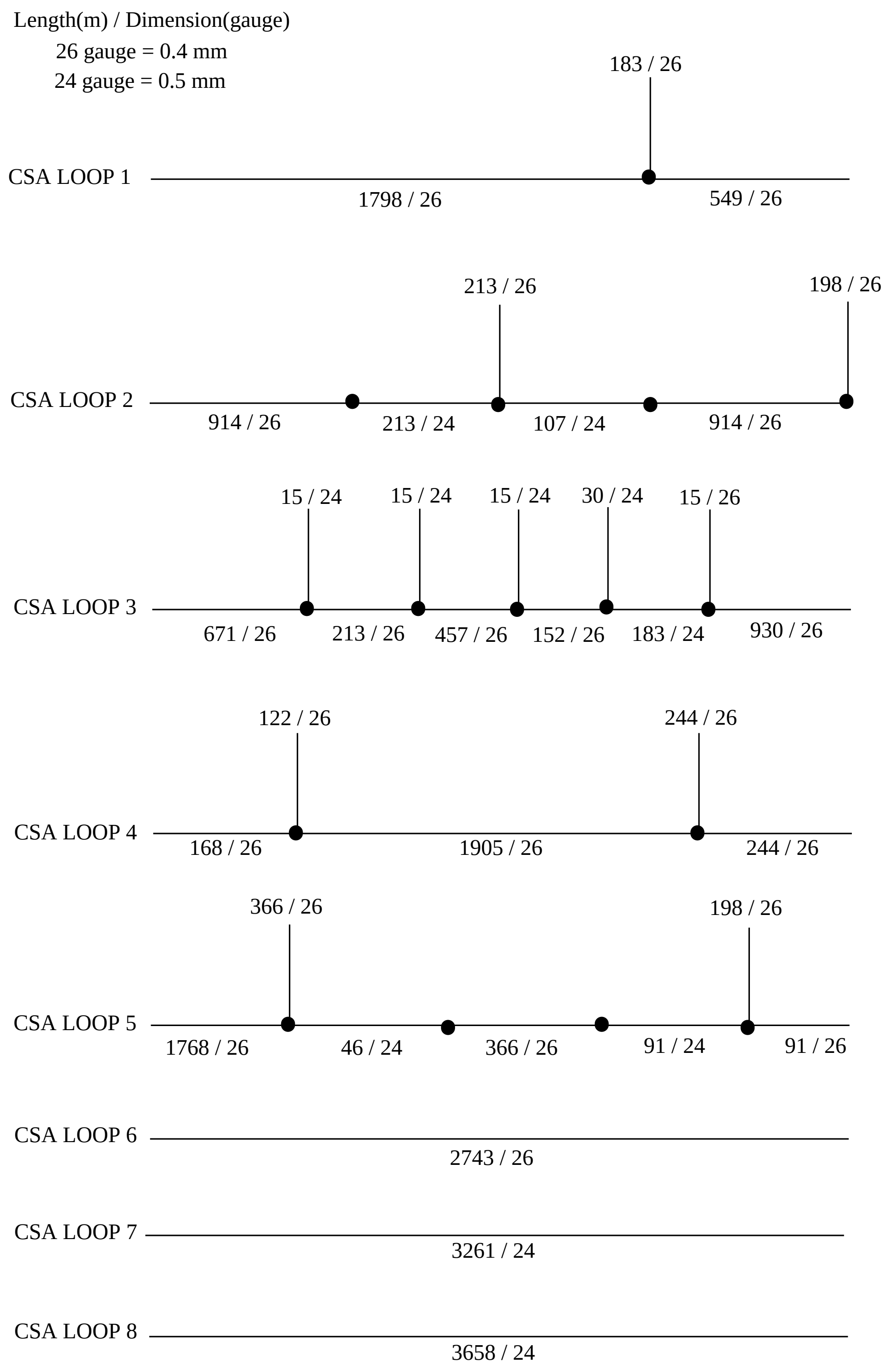
3.11 DAC

A digital to analog converter (DAC) converts the signal from digital values to the analog world. An *N*-bit DAC converts the digital signal to 2^N different analog values. If the signal is too high it will be clipped but that problem is much bigger for the reverse operation.

3.12 Channel

Between the two modems in ADSL is the channel which in this case is the telephone line. It must be pretty short because the performance decreases with the length to the telephone station.

A telephone line can have a lot of different forms. It can be one very long cable or it can be a cable with a lot of crossings that disturb the signal. In a simulation some standard loops, called csaloops are used. In the toolbox, eight different channel models called csaloop1 - csaloop8 are used. They can be viewed in *Figure 15*. They are presented in more detail in [12]. Different models for testing can be found in [23]. Because everything in the system is not ideal, noise will be added. Read more about noise in Section 5.



The lengths are converted from feet to meter

FIGURE 15. The eight different channel models called CSA Loops that are used in the demo.

3.13 ADC

An analog to digital converter (ADC) converts the signal from a time- and amplitude continuous signal to a time- and amplitude discrete signal. An *N*-bits ADC converts the analog signal to 2^N different digital values. If the signal amplitude is too high it will be clipped. In *Figure 16* a 3 bits converter is explained.

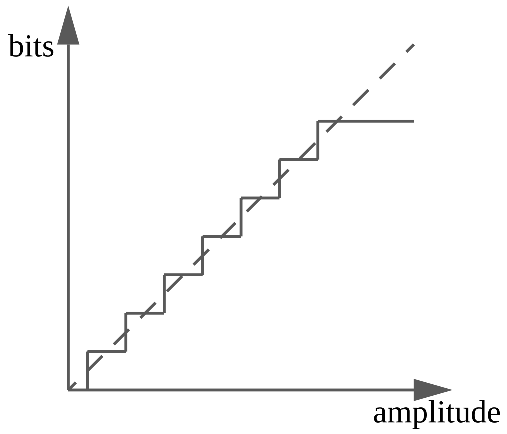


FIGURE 16. An analog to digital converter converts the signal from a time- and amplitude continuous signal to a time- and amplitude discrete signal.

3.14 Time Domain Equalizer (TEQ)

The main idea with the TEQ filter is to mitigate the intersymbol interference (ISI) that appears between two different symbols. ISI appears because two symbols overlap each other. The main idea with a TEQ filter is to push the ISI to a small range. If the range is shorter than the cyclic prefix all the problem can be removed. The theory of TEQ is more explained in Section 4.1.

In the toolbox the minimum-ISI method that optimizes for shortest possible length of the ISI is used. That method is more explained in Section 4.1.3.

3.15 Frequency Domain Equalizer (FEQ)

The Frequency Domain Equalizer is a vector with complex values that is multiplied row wise with the subchannels, if the cyclic prefix is sufficiently large and the TEQ removes the ISI completely. That also means that if the prefix length is longer than the length of the channel impulse response, then the FEQ is only one single complex coefficient for each subchannel, $C_{FEQ} = A \cdot e^{j \cdot \varphi}$. In the calculations it is assumed that the condition is fulfilled. The constellation points before and after a FEQ filter can be seen in *Figure 17*.

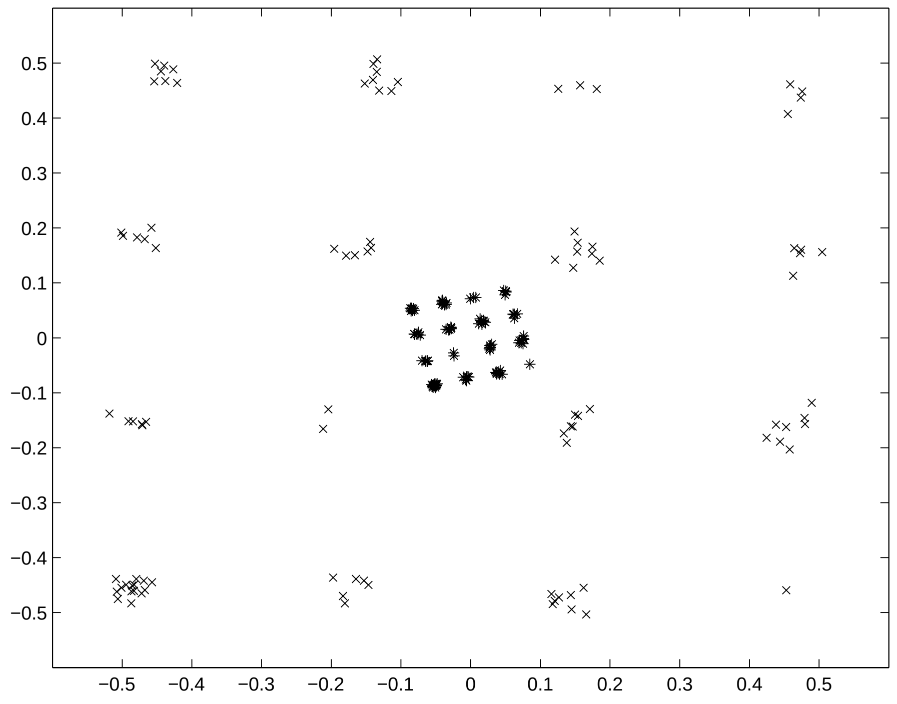


FIGURE 17. Constellation points before(*) and after(x) the FEQ filter.

4 Filter theory

In an ADSL modem two different types of equalizers are applied. These are Time Domain Equalizer (TEQ) and Frequency Domain Equalizer (FEQ).

4.1 Time Domain Equalizer (TEQ)

In Discrete Multitone (DMT) transceivers, a cyclic prefix is inserted between the transmitted symbols. The linear convolution between the channel impulse response and the transmitted symbols becomes circular corresponding to term-by-term product in the frequency domain. It is possible to avoid the intersymbol interference (ISI) completely if the cyclic prefix is longer than the channel impulse response, otherwise it causes a leekage.

4.1.1 Intersymbol Interference (ISI)

Intersymbol Interference (ISI) is a big problem in a communication system. ISI appears between two symbols if they are not equal. In this case the first symbol will stop in a place that is not the same as the second symbol start. That means that we get a jump which will cause the ISI. This phenomenon is explained in *Figure 18*.

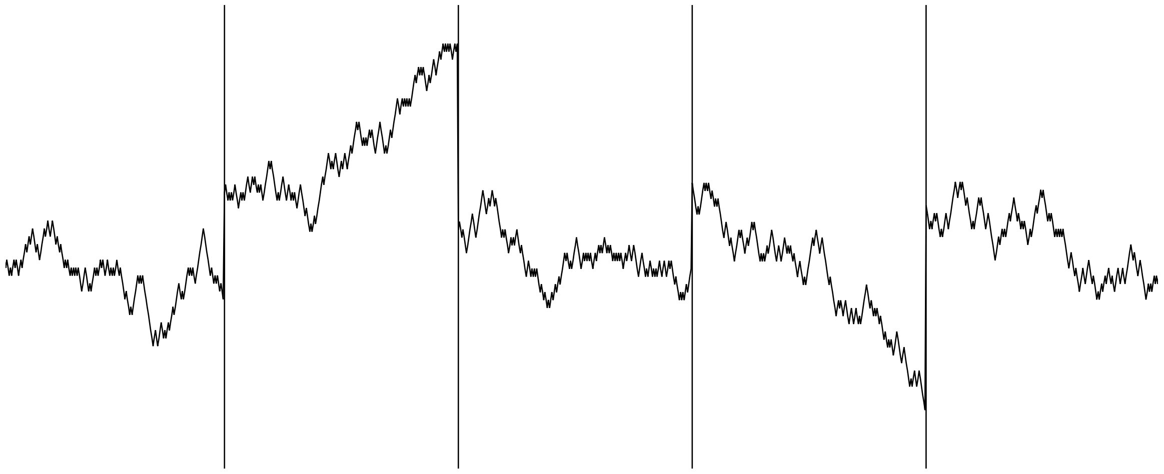


FIGURE 18. Example of different transmitted symbols with various start- and stop amplitude.

In *Figure 19* the channel response that causes the ISI is explained. Here the discontinuity is spread over almost the whole new symbol. This is because the length of the channel impulse response is too long. If we in this case do no operation to make the response better, the result will be really bad and the bit rate would be severely decreased.

What we will do is in some way to move the problem from a long interval to a very short interval. With a TEQ operation that will be done. An example of a good filtering will give the result in *Figure 20*.

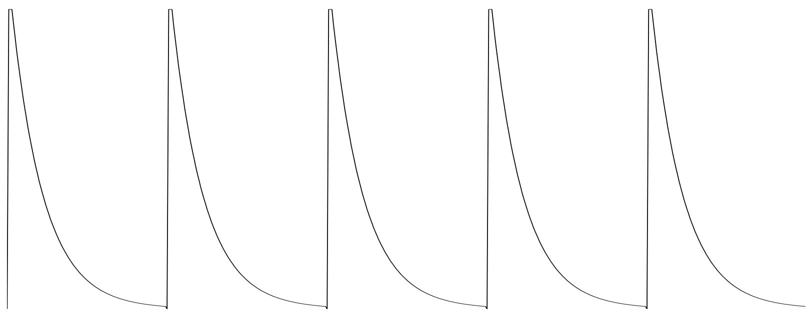


FIGURE 19. Example of channel impulse.

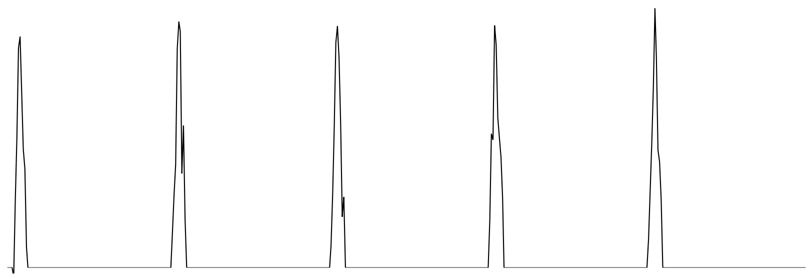


FIGURE 20. Example of channel response after TEQ.

After the TEQ filter almost everything of the problem is transported to the first samples. With a cyclic prefix which contains redundant transmitted symbols it is possible to remove all the intersymbol interference in this ideal example.

4.1.2 Different TEQ methods

There are many different TEQ methods. The big difference between the methods is what they prefer to optimize. The methods can be categorized into two major approaches: methods that minimize the mean squared error and methods that minimize the energy outside of the shortened channel response. All the methods in this chapter are described in [24].

The Minimum Mean Square Error (MMSE) method that minimizes the mean square error is an adaptive method that often is used in practical systems. The Maximum Shortening Signal to Noise Ratio (MSSNR) minimize one component of the channel impulse response, the component that causes ISI. Neither the MMSE nor the MSSNR methods attempt to maximize channel capacity directly. Another method that uses the idea with minimizing the energy in some bands is the minimum intersymbol interference method that is more described in Section 4.1.3

A MATLAB toolbox [25] where all these different TEQ filters are implemented is included in the toolbox. The method that is recommended to avoid intersymbol interference is the Minimum-ISI method that is described below.

4.1.3 Minimum-ISI

The idea with this method is to force the ISI to change its location from a big part of the symbol to only the part with the cyclic prefix. The method that is called Minimum-ISI will minimize the ISI outside the area of the cyclic prefix.

The only effect that TEQ has on the channel capacity is the way it spreads ISI power over different frequencies. This experience is used in the minimum-ISI method. If the total sum of the ISI power is minimized it would be better but this is not the optimal way. The ISI from the frequencies with a dominant noise has no effect on SNR so it can be ignored.

The channel impulse response h and the equalizer w are mixed. Then it is one part that causes ISI, h * w and one part that does not. Also only the part that extends beyond the cyclic prefix causes ISI. See *Figure 21* and *Figure 22*.

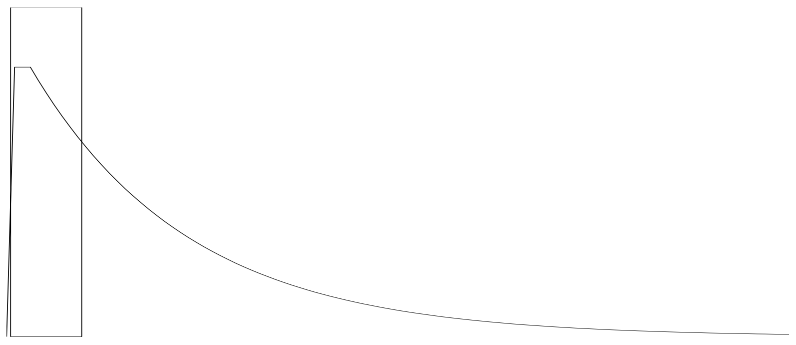


FIGURE 21. The part that does not cause ISI (in the box) and the part that cause ISI (outside the box).

The Optimization Problem Goal [26]:

- Find w that minimizes a weighted sum of the ISI power gains in each subchannel.

$$min\left\|\sqrt{diag(S)} \cdot Q \cdot D \cdot H \cdot w\right\|^2$$

- Prevent w from also minimizing the desired part of the $h \oplus w$.

$$\|G \cdot H \cdot w\| = 1, G = I - D$$

w: The equalizer. This is the little vector that is the answer.

H: The convolution matrix, such that $H \cdot w = h \oplus w$.

D: The windowing matrix. This is a diagonal matrix that isolates the part of $h \oplus w$ causing ISI.

Q: The FFT matrix. Takes FFT of $D \cdot H \cdot w$.

S: The weighting matrix diag(S).

More about the fastest way to solve this equation, the theory about the minimum-ISI and other TEQ functions can be found in [24].

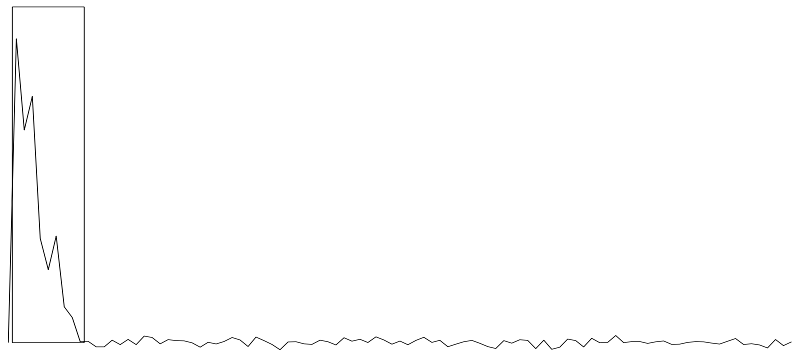


FIGURE 22. Example of a response after a good TEQ filter.

4.2 Frequency Domain Equalizer (FEQ)

The Frequency Domain Equalizer uses an adaptive algorithm. One of the simplest algorithms is the Least Mean Square (LMS) algorithm which I have used in my toolbox.

4.2.1 Least Mean Square (LMS)

Least Mean Square Algorithm (LMS) [27] [28] is a simple adaptive algorithm that is commonly used in practice, see *Figure 23*. The algorithm does not need off-line gradient estimations or data repetitions. For this reason and because it is simple and easy to compute, the LMS algorithm is often used. The algorithm is a very good choice for many adaptive signal processing applications.

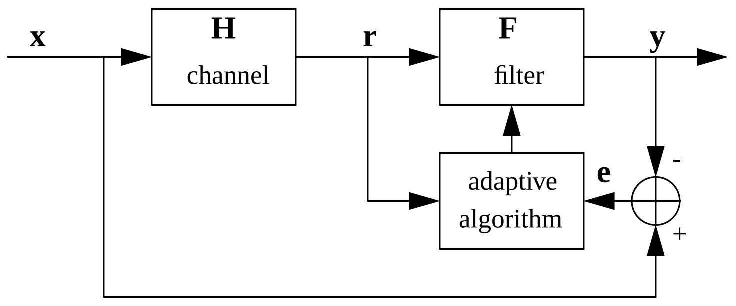


FIGURE 23. The adaptive LMS algorithm that is used in FEQ is a very simple algorithm.

In each iteration of the algorithm the error is calculated as the difference between the transmitted signal and the signal after the equalization process, see Equation 2.

The received signal after the transmission, r is calculated in Equation 3. This signal, the error signal and the old filter are all applied to an adaptive algorithm, Equation 4. This algorithm has different values for different applications depending on how fast the function must adapt and how much variation that is acceptable.

At last the signal is filtered through the equalizer in Equation 5 and everything would take a new round, possibly a little closer to the correct result.

$$e = x - y \tag{EQ 2}$$

$$r = x \cdot H \tag{EQ 3}$$

$$F_N = f(F_{N-1}, e, r) \tag{EQ 4}$$

$$y = r \cdot F \tag{EQ 5}$$

5 Noise

Noise is a combination of signals from intelligent sources, such as other communication systems and non-intelligent sources such as electrical machines. If the noise occurs in the communication frequency band it could affect the quality of the communication. Noise is more described in [12] and [29].

5.1 Sorts of noise

The received signal power is determined by the strength of the transmitted signal, the channel attenuation and the noise that is applied during the channel. The noise can be divided in three major parts: crosstalk noise, impulse noise and background noise.

5.1.1 Crosstalk noise

Crosstalk occurs when signal power leaks between different twisted pair cables. The crosstalk can be divided in two different types, near-end crosstalk and far-end crosstalk which can be seen in *Figure 24*.

Near-end crosstalk (NEXT) [30] appears when two or more cables are near to each other. It is not necessary that the two lines are touching physically on the contrary it is enough that they are close enough. The signal from one of the lines then disturbs the signal on the other line. This is a problem that often appears for system that use the same frequency for connections in both directions.

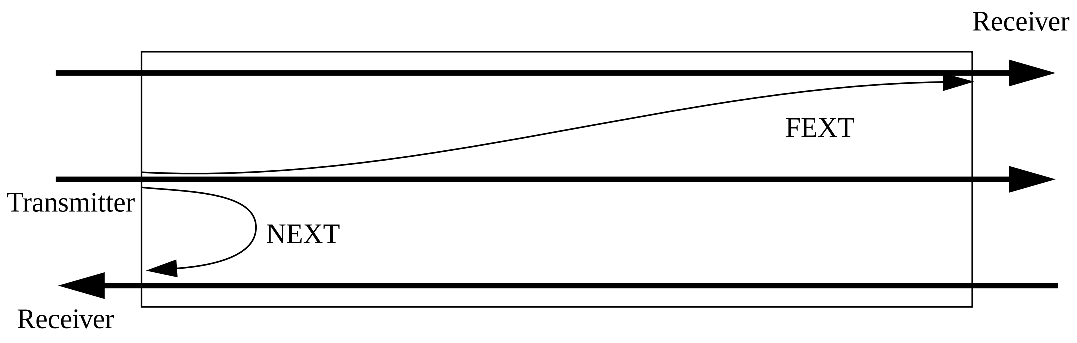


FIGURE 24. The principles of crosstalk noise. NEXT influences the nearest receiver while FEXT influences the receiver far away.

The value of NEXT can be approximately calculated from Equation 6 where S(f) is the spectrum for the transmitted signal and N is the number of other used channels. This formula is valid for a cable with 50 lines.

$$S_{NEXT}(f) = 10^{-13} \cdot \left(\frac{N}{49}\right)^{0.6} \cdot f^{1.5} \cdot S(f) \qquad \text{(EQ 6)}$$

Far-end crosstalk (FEXT) is the noise detected by the receiver located at the far end of the cable from the transmitter that is the noise source. For most systems FEXT is much smaller than NEXT but when the systems use different frequencies for up- and down-stream transmission the NEXT is minimized and the FEXT can then be a bigger problem.

The value of FEXT can be approximately calculated with the help of Equation 7 where d is the channel length in feet and H(f) is the channels transmission function:

$$S_{FEXT}(f) = 8 \cdot 10^{-20} \cdot d \cdot f^2 \cdot |H(f)| \cdot S(f) \tag{EQ 7}$$

5.1.2 Impulse noise

Temporary electromagnetic impulses in the near of the telephone line causes non stationary crosstalk. Example of this sort of noises are when the motor to the refrigerator turns on and off because the door was opening, the power to an elevator because the telephone lines are often run through the elevator shafts in apartment buildings. Another example is the ring signal from another telephone that is sharing the same cable.

5.1.3 Background noise

Background noise is the same as radio noise which comes from a lot of different signals. In this category we have wireless transmission signals on phone lines, particularly AM radio broadcasts and amateur operator transmissions.

5.2 Noise model

A noise model is a complicated method that has not been prioritized to develop in the thesis. Instead a model that is developed by other persons in Ericsson Microelectronics [31] is used. The models are not verified or proved in any way in this thesis. It is only presumed that the models are good enough.

The first thing to do was to convert the noise model that was scaled in dBm/Hz to the scale that is used in the toolbox method which is dB. The formulas that are used are given in Equation 8 to Equation 11.

$$Y = 10 \cdot \log(P_{signal}) \tag{EQ 8}$$

$$P_{signal} = 10^{\frac{X}{10}} \cdot 1.104 \cdot 10^{6} \cdot \frac{1}{256} \tag{EQ 9}$$

$$X(dBm) + 36.3 = Y\left(\frac{dBm}{Hz}\right) \tag{EQ 10}$$

$$X(dBm) + 3 = Z(dB) \tag{EQ 11}$$

After these formulas have been applied, the resulting model is in *Figure 25*.

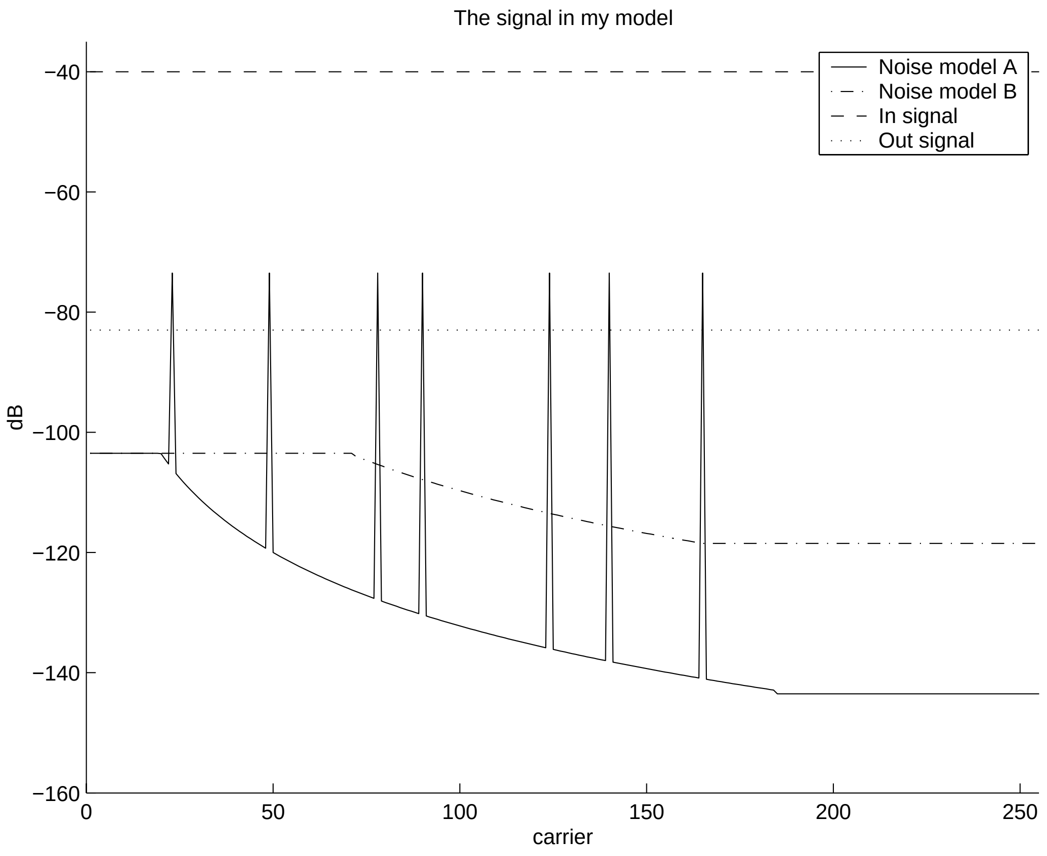


FIGURE 25. Model of noise and signal range. Each carrier correspond to a frequency between 0 and 1.104 MHz.

The power of a transmitted symbol is -36.5 dBm/Hz [1]. Two different noise models are used. Model A has values from -100 dBm/Hz to -140 dBm/Hz with narrow peaks at -70 dBm/Hz. Model B has values between -100 dBm/Hz and -115 dBm/Hz. The transmitted signal in the model is -40 dB and the signal after the channel is -83 dB.

The noise starts on -100 dBm/Hz which is 63.5 dBm/Hz under the signal power. In my model that will be 63.5 dB under the in signal (-40 dB). The noise startpoint is accordingly -103.5 dB.

The difference between the received signal and the noise model is the interesting information in the figure. Another thing to look at is noise model A, which has some peaks that would stop every transmission in some carriers because the noise in these peaks are larger than the signal.

6 Performance

In this chapter the data from the model are presented and explained.

The demo has eight different channels and two different noise models. It has also one mode without any noise at all. From these 24 different cases the bit rate and the bit error rate have been calculated.

The bit error rate (BER) is often chosen to 10^{-7} but in this model that is difficult because a lot of bits would be transmitted and that takes time. Another reason is that the BER would not be so small because no coding operations are used in the model. The coding operations (Reed-Solomon and Trellis) decrease the BER a lot but they are not implemented in this version of the program.

The results from my model are presented in Table 1. The simulations that give these values are optimized for the highest possible rate under the condition that the bit error rate would be lower than approximately 1%.

The results in the model are far from the optimal result. The most important reason for that is the TEQ. TEQ functions are a very big issue that have not been prioritized in this thesis. The TEQ functions that are used in the toolbox were found on Internet and that was the best model for TEQ that was possible to get.

TABLE 1. Result from a simulation with optimized filters.

Channel	Noise	Rate (Mbits)	BER
1	-	2.2486	0.0082
1	a	2.2080	0.0110
1	b	2.0903	0.0106
2	-	1.9239	0.0066
2	a	1.8874	0.0060
2	b	1.8386	0.0070
3	-	1.3151	0.0061
3	a	1.3313	0.0058
3	b	1.2176	0.0049
4	-	2.7154	0.0058
4	a	2.6220	0.0085
4	b	2.0822	0.0060
5	-	3.4297	0.0085
5	a	3.2349	0.0067
5	b	2.6626	0.0080
6	-	0.8929	0.0058
6	a	0.8483	0.0065
6	b	0.8199	0.0077
7	-	0.8158	0.0077
7	a	0.7915	0.0087
7	b	0.7955	0.0061
8	-	0.6494	0.0022
8	a	0.6048	0.0052
8	b	0.6413	0.0037

7 References

[1] ITU - Telecommunication Standardization Sector, "G.992.1: Asymmetrical Digital Subscriber Line (ADSL) Transceiver", 1998.

[2] Cisco, "ADSL Explained", [Cited Nov 2001], http://www.minx.co.uk/Services/ADSL_Services/ADSL_Explained/adsl_explained.html.

[3] Telenordia, [Cited Nov 2001], http://www.telenordia.se.

[4] Microdatorn, "Tre olika tekniker dominerar", 2000, [Cited Nov 2001], http://mikrodatorn.idg.se/guider/md0006/bredband/3.asp.

[5] Telia, [Cited Nov 2001], http://www.telia.se.

[6] K Ehrsson, "ADSL-tips", 2001, [Cited Nov 2001], http://w1.857.telia.com/~u85710476/docs/adsl02se.html.

[7] ZD Net News, "The History Of ADSL", 1998, [Cited Nov 2001], http://www.zdnet.com/zdnn/stories/news/0,4586,281734,00.html.

[8] R. Baines, "DMT vs. CAP Line Codes", 1997, [Cited Nov 2001], http://www.analog.com/publications/whitepapers/whitepaper_html/content.html.

[9] Acceleration, "Different types of DSL", [Cited Nov 2001], http://www.acceleration.net/accessproducts/TypesofDSL.html.

[10] Analog Devices, Inc., "Status & Issues in xDSL", 1995-2001, [Cited Nov 2001], http://www.analog.com/publications/whitepapers/products/xDSL.html.

[11] J. Bingham, "ADSL, VDSL, and Multicarrier Modulation", John Wiley & Sons, Inc., 2000.

[12] T. Starr, J.M. Cioffi and P.J. Silverman, "Understanding Digital Subscriber Line Technology", Prentice Hall PTR, 1999.

[13] M. Karlsson-Rudberg, "DSP algorithms and architectures for telecommunicatin", PhD thesis 705, Dept. of Electrical Engineering, Linköpings universitet, 2001, [Cited Nov 2001], www.es.isy.liu.se/staff/mikaelr/phd/phd.pdf.

[14] M. Schlegel, "High Bit Rate Data Transmission over the Telephone Loop Plant, Emphasising on DMT Modulation Scheme", 1999, [Cited Nov 2001], http://www.bib.fh-lippe.de/volltext/dipl/schlegel/.

[15] K. Saarela, "ADSL", 1995, [Cited Nov 2001], http://www.cs.tut.fi/tlt/stuff/adsl/pt_adsl.html.

[16] GAO Research & Consulting Ltd., "G.992.2 (ADSL G.Lite)", 1996-2001, [Cited Nov 2001], http://www.gaoresearch.com/resources/whitepapers/g992.html.

[17] "Digital Subscriber Line (xDSL) FAQ v20010108", [Cited Nov 2001], http://www.cs.uu.nl/wais/html/na-dir/datacomm/xdsl-faq.html.

[18] A. Chan, S. So and W. Shum, "Project in Data Communication", 2000, [Cited Nov 2001], http://www.comm.toronto.edu/~karen/projects/10.ADSL/.

[19] Orckit Communications Ltd., "How does ADSL work", 1998, [Cited Nov 2001], http://www.orckit.com/fr_newsa.html?/how_does_ads_works2.html.

[20] M. Korpi, "DSP Architecture for Viterbi Decoding of Trellis-Coded Modulation Codes", LiTH-ISY-EX-3020, 2000.

[21] L. Torres-Méndez, "The Viterbi Algorithm", [Cited Nov 2001], http://www.cim.mcgill.ca/~latorres/Viterbi/va_alg.htm .

[22] Engineering Productivity Tools Ltd., "The FFT Demystified", 1999 [Cited Nov 2001], http://www.eptools.com/tn/T0001/INDEX.HTM.

[23] Acterna, "Technical resources", 2001, [Cited Nov 2001], http://www.acterna.com/technical_resources/application_notes/index.html.

[24] G. Arslan, "Equalization for Discrete Multitone Transceivers", Ph.D. Dissertation, Dept. of Electrical and Computer Engineering, The University of Texas at Austin, 2000, [Cited Nov 2001], http://www.ece.utexas.edu/~bevans/students/phd/guner_arslan/phd.pdf.

[25] G. Arslan and B. Lu, “MATLAB DMTTEQ Toolbox”, 2001, [Cited Nov 2001], http://anchovy.ece.utexas.edu/~arslan/dmtteq/dmtteq.html.

[26] J. Wu, G. Arslan, B. L. Evans, "Efficient Matrix Multiplication Methods to Implement a Near-optimum Channel Shortening Method for DMT Transceivers", [Cited Nov 2001], http://www.ece.utexas.edu/~bevans/papers/2000/shortening/TEQdesign2000talk.pdf.

[27] G. Shankar and P. Makineni, "LMS-algorithm", 1996, [Cited Nov 2001], http://www.ee.duke.edu/~gsm/projects/adap-filter/subsection3_4_2.html.

[28] D. L. Anair, "LMS", 1999, [Cited Nov 2001], http://backhoe.ee.cornell.edu/BERG/downloads/sim_tools/spanc_me/report/node16.html.

[29] D. J. Rauschmayer, "ADSL/VDSL Principles", Macmillan Technical Publishing, 1999.

[30] Search Networking, "Near-end Crosstalk", 2001, [Cited Nov 2001], http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci214338,00.html.

[31] T. Randahl, "Noise model A & Noise model B", MATLAB routines, Ericsson Microelectronics AB, Internal material.

A Toolbox

This chapter contains a list of all files in the toolbox. For each of the functions all input- and output- arguments are specified and explained.

Modem

The argument modem is used in several functions. It is realized as a struct that contains the following variables:

- ADSL_carriers - The variable is an integer, N that describe the number of carriers. This toolbox is only tested for a downstream transmission, also N=256. It does not work for other values of N because the channel model that is used simulate a downstream transmission.
- ADSL_channel_model_parameters - This is a vector of length 512. Because this model is a simulation of a downstream transmission, it has a specific value and it is not possible to use it for other transmissions.
- ADSL_FEQ_filter_coefficients - This is a vector of N complex values.
- ADSL_TEQ_filter_coefficients - This is a vector with 16 real values.
- ADSL_cyclic_prefix - This is an integer. Normally it has the value 32.
- ADSL_init_signal - This is one symbol of the training signal reverb. The length of the vector is N.
- ADSL_bit_table - This vector of length N is initialized during the training. The first value is always 0 and the pilot tone is also set to 0 to avoid transmission on that tone.
- ADSL_direction - This is a integer that work as a flag. It has the value 0 for downstream.
- ADSL_snr - This is the snr value for each of the tones. The vector has the length N.
- ADSL_noise - This is the noise from a noise model. The vector has the length N if the noise is applied.

The modem have some variables that are stored here temporary only because it is a practical way to transport them.

- const_point_in - All transmitted symbols during a test period. They are stored in an array of size $N\cdot$ testrounds.
- const_point_out - All received symbols during a test period. They are stored in an array of size $N\cdot$ testrounds.
- ber - This is a vector of length N that contain the bit error rate for each of the tones.
- signal1 - A vector of length $2\cdot N$+prefix_length that contain the real values that are transmitted during one test symbol.
- signal2 - A vector of length $2\cdot N$+prefix_length that contain the real values that are received during one test symbol.

A.1 Script

In this section all scripts used for various types of simulations are described.

1. adsldemo.m

adsldemo()

The main demo function. See Appendix B.

2. init_modem.m

modem = init_modem()

init_modem() initiate a modem to the file modem.mat. This script creates modem structure. The modem is trained during a filter training operation for the best possible transmission. The SNR are calculated for these values and the bits are allocated to the tones with the best SNR values through an allocation algorithm.

3. performance.m

[const_point_in, const_point_out, ber] = performance(modem, rounds)

performance() calculate the bit error rates for each channel and the constelleation point that are transmitted and received. The function transmit random bits over the channel.

const_point_in: A matrix of size N· rounds with the transmitted constellation points.

const_point_out: A matrix of size N· rounds with the received constellation points.

ber: A vector of length N with the bit error rate for each channel.

rounds: The number of rounds (symbols) that are transmitted.

4. tmpresults.mat

In this file all the temporary results are saved. This file will be saved in the directory where the adsldemo is started from.

5. *plot_const_points.m*

plot_const_points(in, out, bits)

plot_const_points() plot the transmitted and the received points that are transmitted over a given channel. The function plot transmitted points in red, received points in blue and the lines in green. A plot can be seen in *Figure 26*.

in: A vector with the transmitted signals.

out: A vector with the received signals.

bits: The number of bits that are transmitted over the channel.

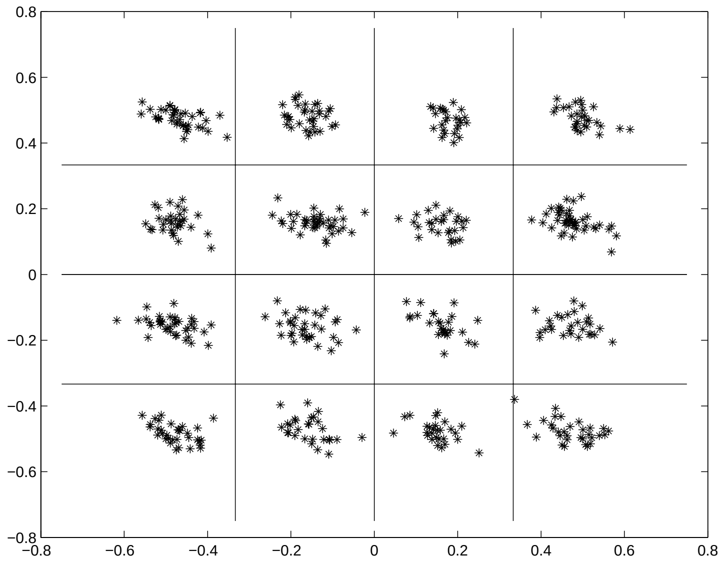


FIGURE 26. Result plot from the function plot_const_points.

6. *signals.m*

[signal1, signal2] = signals(modem)

signals() calculate the signal before and after the transmission. This signals are later plotted in the demo and that is the only idea with this function.

signal1: A signal of size 2· N+prefix_length real values. The signal before transmission.

signal2: A signal of size 2· N+prefix_length real values. The signal after transmission.

A.2 Transmitter

7. crc.m

crc-bits = crc(message)

crc() calculate the cyclic redundancy check bytes. The formula that are used is $c(D) = M(D) \cdot D^8 mod G(D)$ there $G(D) = D^8 + D^4 + D^3 + D^2 + 1$

message: A vector with binary numbers. The length can variate from 67 bytes to approximately 14875 bytes. The length depends on different initialization parameters for the frame.

crc-bits: A vector with eight bits.

8. scramble.m

[symbol, scramble_memory] = scramble(symbol, scramble_memory)

scramble() avoid bursts of zeros or ones in a long bitstream. The function is an xor operation between the incoming bit and the bits that appeared 18 and 23 bits before. Actually this is a bitstream operation but in this model it work instead on a symbol. To make this possible and still have a fast operation the memory vectors are saved in variable scramble_memory.

symbol: A vector with bits. The length of the vector depends on different initialization parameters that not will be further commented in this thesis.

scramble_memory: A vector with 23 bits containing the last bits from the symbol that are saved to the next round.

9. rs_encode.m

symbol = rs_encode(modem, symbol)

rs_encode() encode a symbol with the Reed-Solomon code operations. This function is not implemented.

symbol: A vector with bits. The length of the vector depends on different initialization parameters that will not be further commented in this thesis.

10. interleave.m

[symbol, interleave_memory] = interleave(symbol, interleave_memory)

interleave() spread the burst of errors over a large area. Symbol is a vector with the incoming bytes and interleave_memory is the bytes that have been saved from the last round. The function result in an interleaved symbol and the new memory

symbol: A vector with bytes. The length of the vector depends on different initialization parameters that will not be more commented in this thesis.

interleave_memory: A vector with bytes. The length of the vector depends on different initialization parameters.

11. tone_order.m

symbol = tone_order(modem, bitstring)

The tone order algorithm assign bits for each tone starting with the tones with the least number of bits. This operation is going on until all bits are assigned. Only values between 2 and 15 are used.

bitstring: A vector with bits. The bitstring must not have more bits than the modem can transmit in one symbol.

symbol: A matrix of size $N \cdot 15$ with binary values.

12. constellation.m

symbol = constellation(modem, symbol)

constellation() generates a complex output vector as the tone value for each frequency bin.

out_symbol: A vector of length N with complex values.

symbol: A matrix of size $N \cdot 15$ with binary values.

13. constellation_viterbi.m

out_symbol = constellation_viterbi(modem, symbol)

constellation_viterbi() generates a viterbi encoded complex output vector as the tone value for each frequency bin.

out_symbol: A vector of length N with complex values.

symbol: A matrix of size $N \cdot 15$ with binary values.

14. vitenc.m

[v,w] = vitenc(u, op)

vitenc() produces bits according to the ADSL specification. This function is called from constellation_viterbi.

v, w: Output bits according to the ADSL specification

u: Input bits

op: If op = ’reset’ is specified, the FSM state is reset. If ’op’ = ’force’ is specified, a 4-d symbol is produced which forces the FSM back towards zero.

15. dsl_idft.m

out_symbol = dsl_idft(symbol, carriers)

dsl_idft() calculate the inverse discrete fourier transform of the symbol in the number of points that are defined by carriers. The result is a vector of real numbers. Symbol is a vector with complex values and carriers are the number of points in which the function is applied. These idft use the matlab ifft function for the transform.

out_symbol: a vector with $2\cdot$ N real values

symbol: a vector with N complex values

carriers: The number of carriers that are defined in modem. The integer N.

16. add_prefix.m

out_symbol = add_prefix(symbol, prefix_length)

add_prefix() copy the last part of the symbol to the beginning.

out_symbol: A vector with $2\cdot$ N+prefix_length real values.

symbol: A vector with $2\cdot$ N real values.

prefix_length: The length of the prefix.

17. da.m

out_symbol = da(symbol, N)

da() is a converter that convert the digital symbol to analog values. If the values are too large they are clipped. This function is not implemented.

out_symbol: A vector with real values between -1 and 1.

symbol: A vector with real values between -1 and 1.

N: The number of bits in the DAC. The DAC has also 2^N possible values.

18. channel.m

out_symbol = channel(symbol, old_symbol, h)

The function filters the symbol through the channel model. To avoid cut off problem the symbol must also filter through the latest symbol. Actually, this is a bitstream operation but in this model it works instead on a symbol. To make this possible, and still have a fast operation, the old_symbol is saved.

out_symbol: A vector with $2 \cdot N$+prefix_length real values.

symbol: A vector with $2 \cdot N$+prefix_length real values.

old_symbol: A vector with $2 \cdot N$+prefix_length real values.

h: The channel model.

A.3 Receiver

19. ad.m

out_symbol = ad(symbol, N)

ad() is a converter that convert the analog symbol to digital values. If the signal amplituds are too big they are clipped. This function is not implemented.

out_symbol: A vector with real values between -1 and 1.

symbol: A vector with real values between -1 and 1.

N: The number of bits in the ADC. The ADC has 2^N possible values.

20. add_noise.m

out_symbol = add_noise(symbol, noise)

add_noise() apply noise to the symbol

out_symbol: A vector with 2· N+prefix_length real values.

symbol: A vector with 2· N+prefix_length real values.

noise: A vector with 2· N+prefix_length real values.

21. noise_model_a.m

noise = noise_model_a()

noise_model_a() calculate the noise model.

noise: The model that is used in *add_noise*. A vector with 2· N+prefix_length real values.

22. noise_model_b.m

noise = noise_model_b()

noise_model_b() calculate the noise model.

noise: The model that is used in *add_noise*. A vector with 2· N+prefix_length real values.

23. teq.m

out_symbol = teq(symbol, old_symbol, filter_coeff)

The function is a filter but both the symbol and the previous symbol are used to filter without cut-off problems

out_symbol: A vector with 2· N+prefix_length real values.

symbol: A vector with 2· N+prefix_length real values.

old_symbol: A vector with 2· N+prefix_length real values.

filter_coeff: The TEQ filter coefficients. A vector with 16 values.

24. strip_prefix.m

out_symbol = strip_prefix(symbol, prefix_length)

strip_prefix() remove the first part of the symbol and result in a new vector that have prefix_length lesser values.

out_symbol: A vector with 2· N real values.

symbol: A vector with 2· N+prefix_length real values.

prefix_length: The length of the prefix..

25. dsl_dft.m

out_symbol = dft(symbol, carriers)

dsl_dft() is a discrete Fourier transform of the symbol. The transform is applied in carrier points and result in a vector with complex values. These function use the matlab fft function

out_symbol: A vector with N complex values.

symbol: A vector with 2· N real values.

carriers: The number of carriers, N.

26. feq.m

symbol = feq(symbol, feq_filter)

feq() is a frequency domain equalizer that correct the received symbol through a complex multiplication.

symbol: A vector with N complex values.

feq_filter: The FEQ filter coefficients is a vector with N complex values

27. constdecodode.m

out_symbol = constdecode(modem, symbol)

constdecode() decode adsl constellation points.

out_symbol: A matrix of size N· 15 with binary values.

symbol: A vector with N complex values.

28. tone_deorder.m

bitstring = tone_deorder(modem, symbol)

tone_deorder() take the bits from each tone in correct order. The algorithm start with the tones with the least number of bits.

bitstring: A vector with bits.

symbol: A matrix of size N· 15 with binary values.

29. deinterleave.m

[symbol, interleave_memory] = deinterleave(symbol, interleave_memory)

deinterleave() change order of bit to the order that was before the interleave operation.

symbol: A vector with bytes. The length of the vector depends on different initialization parameters that will not be more commented in this thesis.

interleave_memory: A vector with bytes. The length of the vector depends on different initialization parameters.

30. rs_decode.m

symbol = rs_decode(modem, symbol)

rs_decode() decode a symbol with the reed solomon code operations. After the operation the number of bits had decreased. Also in this operation the number of bits depends on different initialization parameters. This function is not implemented.

symbol: A vector with bits. The length of the vector depends on different initialization parameters that will not be more commented in this thesis.

31. descramble.m

[symbol, scramble_memory] = descramble(symbol, scramble_memory)

descramble() rearrange the bits that were changed to avoid burst of zeros or ones in a long bitstream. The function is a xor operation between the incoming bit and the bits that appeared 18 and 23 bits before. Because the memory vectors are saved it is possible to have a fast operation.

symbol: A vector with bits. The length of the vector depends on different initialization parameters that will not be more commented in this thesis.

scramble_memory: A vector with 23 bits. It is the last bits from the symbol that are saved to the next round.

A.4 Init

32. make_modem_fast.m

modem = make_modem(loopnr, noisenr)

make_modem_fast() initialize the struct modem with default values and values after the TEQ training, FEQ training and SNR calculation operations.

loopnr: The number of one of the eight possible loops that the function support.

noisenr: The number of the noise model. 0 for no noise at all, 1 for noise model A and 2 for noise model B.

33. make_TEQ_results.m

make_TEQ_results()

make_TEQ_results() are used to speed up the calculation. The TEQ training results are calculated and the result is saved in TEQ_results.mat. The function call the *my_teq* function.

34. TEQ_results.mat

Saved values.

35. make_FEQ_results.m

make_FEQ_results(trainrounds)

make_FEQ_results() are used to speed up the calculation. The FEQ training results are calculated and the result is saved in FEQ_results.mat. The function call the filter_train function.

trainrounds: The number of rounds that are used for the calculation. For the saved values in the program, 50000 rounds are used.

36. FEQ_results.mat

Saved values.

37. make_SNR_results.m

make_SNR_results(snrrounds)

make_SNR_results() is used to speed up the calculation. The FEQ training results are calculated and the result is saved in FEQ_results.mat. The function call the calc_snr function.

snrrounds: The number of rounds that are used for the calculation. For the saved values in the program, 500 rounds are used.

38. SNR_results.mat

Saved values.

39. make_modem.m

modem = make_modem()

make_modem() initialize the struct modem with default values.

40. my_teq.m

[W, channel] = my_teq(loopnr)

my_teq() calculate the TEQ filter and the channel model.

loopnr: The number of one of the eight possible loops that the function can simulate.

W: The TEQ filter coefficient. A vector with 16 values

channel: The channel coefficients. A vector with 512 values.

41. filter_train.m

modem = filter_train(modem, rounds)

filter_train() return the same struct that it has as input, but with a trained FEQ filter. The function transmits a special training sequence and uses an adaptive training algorithm, LMS, to find a good filter

rounds: The number of rounds (symbols) that are used for the training.

42. feq_train.m

Hnew = feq_train(vpre_eq, v, vexpected, H, lambda)

Adaptive MSE frequency doimain equalizer. Complex LMS algorithm implemented.

vpre_eq: The received vector of length 2· N, before FEQ filtering.

v: The input vector of length 2· N in frequency domain.

vexpected: The expected value of v after adaption. A vector of length 2· N.

H: The old FEQ filter. A vector of length 2· N.

lambda: Parameter that controls adaption rate, a value close to zero means slow adaption.

Hnew: The new FEQ filter. A vector of length 2· N.

43. calc_snr.m

modem = calc_snr(modem, rounds)

calc_snr() return the same struct that it has as input, but it has changed the value of SNR. Modem is a struct with all information about the modem and rounds is the number of rounds that the test transmission make. The function transmits a special sequence and calculates the SNR value for this transmission.

rounds: The number of rounds (symbols) that are used to calculate SNR.

44. allocate_bits.m

modem = allocate_bits(modem, margin)

allocate_bits() returns an updated modem struct with a new bit loading table. From the incoming struct modem the function use the SNR values. The allocation algorithm is easy. Only 3 dB is given to each bit.

margin: The number of dB:s that are taken from the SNR value before the allocation to get a better performance.

45. bit_table_results.mat

Saved values.

A.5 Signals

46. c_ect.m

c_ect is a training signal that has no special form. It is up to the transmitter to choose

47. c_pilot.m

c_pilot is a function that transmits a symbol in only one tone, the pilot tone.

48. c_reverb.m

c_reverb is a pseudo random sequence defined through a simple xor algorithm.

49. c_medley.m

c_medley is a pseudo random sequence that is defined by a simple xor algorithm.

50. c_quiet.m

c_quiet is a signal that do not transmit anything.

51. c_segue.m

c_segue is a signal that is defined as the inverse to *c_reverb.*

52. r_reverb.m

r_reverb is a signal for upstream transmission.

A.6 Channels

This section have eight files, named csaloop1.time - csaloop8.time. Each of these files hold the filter coefficients for one test channel. These eight different files are simulations for loops that are constructed to use for testing of adsl modems. The loops are further described in [12].

A.7 Newdmtteq

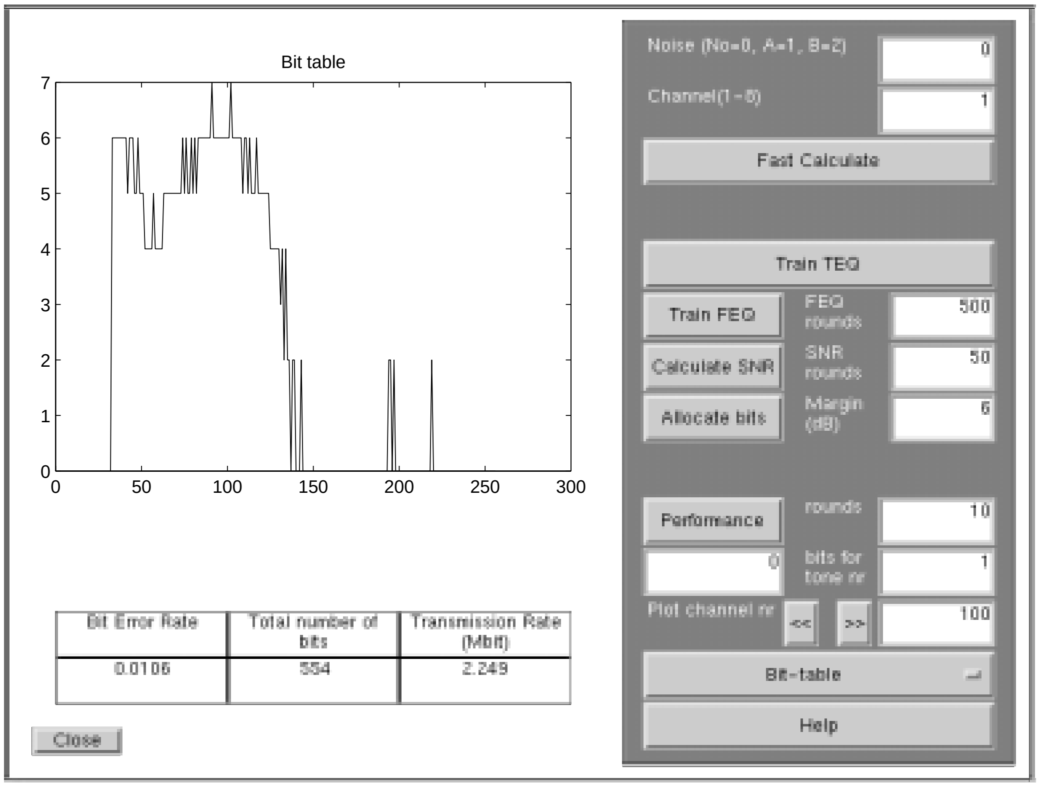
This is a toolbox found on Internet [25]. The toolbox is used to create the TEQ filter. The only change I have done is to remove one line in the function perform to avoid writing to a file.

TABLE 2. Files in the newdmtteq toolbox

calcsnrs	*compare*	*correlation*	*dcc*
dcm	*dht*	*dsl*	*eigapp*
eigen	*geo*	*geosnr*	*graphdata*
hankelize	*maxeig*	*mbr*	*mineig*
minisi	*mmpteq*	*mpteq*	*mssnr*
obje	*objective*	*perform*	*prd*
remainenergy	*selmeth*	*setprogbar*	*siggen*
specestim	*tabledata*	*teqdemo*	*trainsig*
uec	*updateprogbar*	*utc*	

B The demo

This chapter will explain all the things that appear in the ADSL Matlab Model Demo and also recommend which commands that will be best to use for an optimal learning time by this demo. The demo can be seen in *Figure 27*.



Noise (No=0, A=1, B=2): 0
Channel(1-8): 1
FEQ rounds: 500
SNR rounds: 50
Margin (dB): 6
rounds: 10
0
bits for tone nr: 1
Plot channel nr: 100

Bit Error Rate	Total number of bits	Transmission Rate (Mbit)
0.0106	554	2.249

FIGURE 27. The demo.

B.1 Buttons

The demo has several buttons. In this section the field of application for each of them are explained.

- **Fast Calculate**

Use precalculated values to speed up the simulations.

- **Train TEQ**

This is the first button that will be used if you have to do a complete simulation with a new channel. This function initialize the modem and train the TEQ. The training depends only on the channel.

- **Train FEQ**

Train FEQ use the adaptive training algorithm to find good FEQ filters. The training is done in the number of rounds that are defined in the FEQ rounds textbox.

- **Calculate SNR**

This function calculate the signal noise ratio. Through transmitting a lot of data the function produce an SNR value that is used later.

- **Allocate bits**

Takes the snr value and allocate the bits. This function also uses the margin from a textbox. The allocation is a fast operation and for now it is very easy. It just takes the SNR, subtracts the margin and give a bit for each three of the remaining dBs.

- **Performance**

Make a new calculation of the performance. This function calculate the constellations and the bit error rate. The calculation is dine in the number of rounds that are in the performance bits textbox.

- **Graphic menu**

The graphic menu have six different choices. The chosen alternative is plotted.

1. Bit-table - Plot the bit-table.
2. SNR - Plot the signal noise ratio.
3. Bit Error Rate - Plot the bit error rate for each channel.
4. Constellation - Plot transmitted and received constellations for each channel.
5. Signal before transmission - Plot a signal before the transmission.
6. Signal after transmission - Plot a signal after the transmission.

- **Help**

A helptext will be displayed when this button is pressed.

- **Close**

Close the demo.

B.2 Editable textboxes

In the editable textboxes the values can be changed and for some of them it will then be forced to a recalculation.

- **Noise**

Noise is 0, 1 or 2, there 0 means no noise, 1 means that noise model A is used and 2 means that noise model B is used.

- **Channel**

This is a choice between the eight different channels that the model support.

- **FEQ rounds**

This is the number of rounds the FEQ training will use.

- **SNR rounds**

This is the number of rounds the SNR training will use.

- **Margin**

The margin that are used in the bit allocation.

- **Performance bits**

The number of bits are transmitted in the performance calculation.

- **Set bits for tone**

Set a number of bits for one of the channels. It is used to block channels with a lot of errors.

- **Plot Channel nr**

Which channel that will be plotted in graphic menu choice constellation.

B.3 Results

Here are explained results that are shown.

- **Bit Error rate**

The number of wrong received bits divided by the total number of transmitted bits.

- **Total number of bits**

The number of bits that can be transmitted during one symbol.

- **Transmission Rate**

The transmission speed which only depends on the number of bits that can be transmitted.

B.4 Demonstrations

- **First demonstration**

The demo will be started with the command adsldemo from MATLAB. When the demo starts it has already loaded a model that has a transmission over channel nr 1. Things to test:

1. Look at the different graphic plots.
2. Look at the constellation plot and test different value in the plot channel textbox. Note that it is only the channels where anything is transmitted that are useful.
3. Increase the performance bits value. Push ***Performance***. Now there are more constellation points to look at because more bits are transmitted.
4. If some of the channels have much bit errors, test to change the number of bits that are transmitted over the channel. Push ***Performance***.

- **Second demonstration**

1. Choose another channel and/or noise model
2. Push ***Fast Calculate***.
3. Push ***Performance***.
4. Different models give different performances.

- **Third demonstration**

1. Do not use the fast calculation.
2. Train TEQ.
3. Train FEQ.
4. Calculate SNR.
5. Allocate bits.
6. Performance.
7. Check the results.