User Manual

go2ANALYSE 2.2

by PLATH AG, Switzerland

Imprint

PLATH AG
Stauffacherstrasse 65
CH-3014 Bern

Phone: +41 311 6446
www.go2signals.ch
info@go2signals.ch

Printed: 5 August 2013

Contents

Introduction

The Bitstream Processor (go2ANALYSE) is powerful software for offline analysis of bitstreams, e.g. the investigation of decoder characteristics. It can either be used as an offline tool or in combination with a COMINT software suite such as go2DECODE. Easy-to-use functions allow fast and partially automatic processing of bitstream inputs. It can display the bitstream in several modes, the bitstream can be examined applying functions such as pattern search, tagging of relevant patterns or descrambling. Statistical analysis functions are available as well as autocorrelation functions, fast pattern searching, application of different code tables and the search for periodic and non-periodic sequences. User defined analysis functions can be added by use of the decoder description language DDL.

About this Manual

This instruction manual provides basic information about the operation and the application of go2ANALYSE. The first chapter gives an overview of the functional contexts. Installation instructions are included in the second chapter. The third chapter gives an overview of operating the go2ANALYSE software. The details about the functions and displays are explained in the subsequent chapters.

In the case of questions or suggestions, please use the contacts stated in “Service” on page 87.

Setup

Installation

This chapter describes the installation on a computer with Windows 7 operating system. For the installation of the application go2ANALYSE you need the installation data carrier (CD/DVD, USB-Memory, ZIP file) with the require files from PLATH AG. During the installation the application and the WibuKey driver are installed. If the installation of the dongle is not started automatically, locate the WibuKey folder on the installation data carrier and install the dongle manually. After the installation, the USB dongle must be plugged in to run the application.

Put the CD into the CD-drive of your workstation and execute the file Setup.exe on the CD. Follow the installation wizard as described in the following figures:

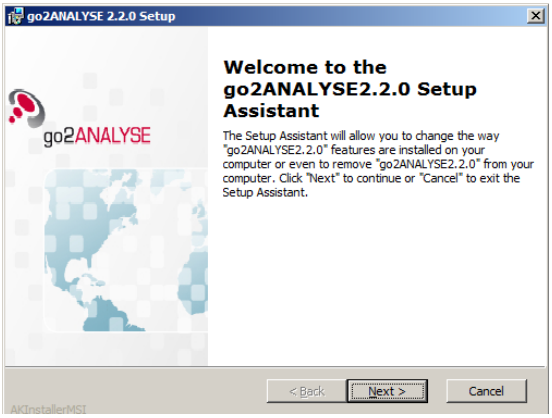


Figure 1: Welcome Screen of the Installation Assistant

Read the text in the dialog and press <**Next**>.

Figure 2: License Agreement

Please read the license agreement and select **I accept the terms in the License Agreement**. Click on <**Next**>.

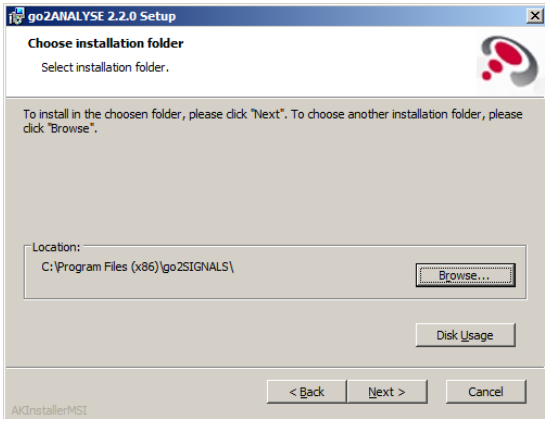


Figure 3: File Location

Check whether the installation location is correct. If necessary, browse for a different location. Click on <**Next**>.

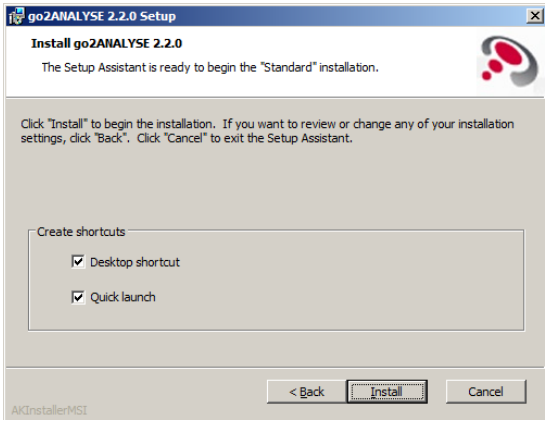


Figure 4: Standard Installation

Click on <**Install**>.

The WibuKey installation starts automatically. Depending on the language setting of your operating system the language in the following dialogs may vary.

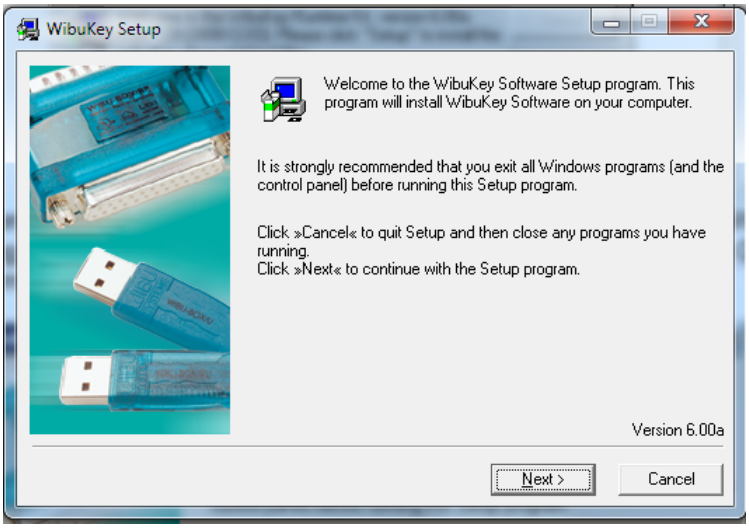


Figure 5: WibuKey Setup

Read the text and click on <**Next**>.

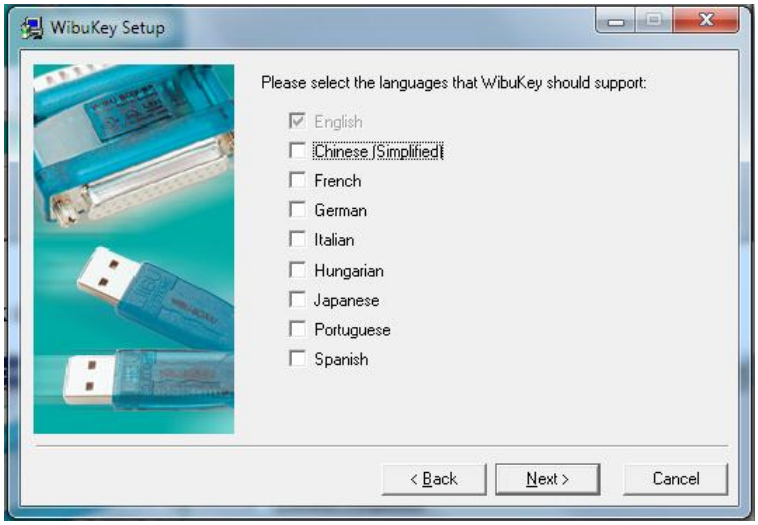


Figure 6: Select Language for WibuKey

Select the required languages. Click on <**Next**>.

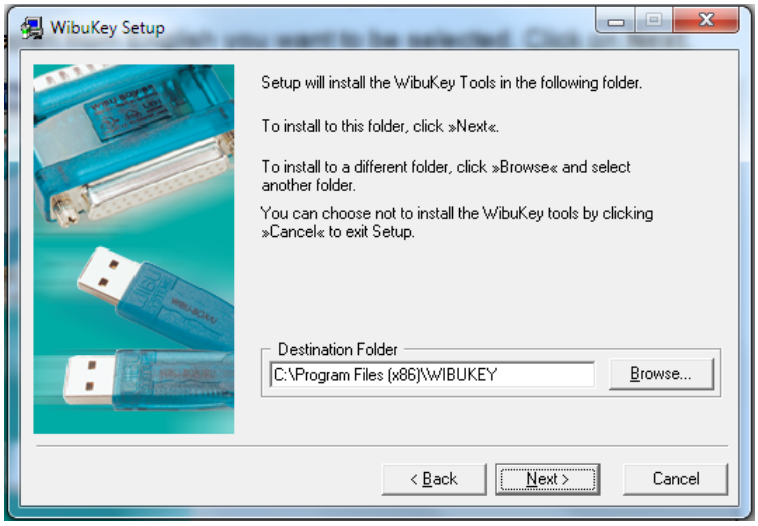


Figure 7: Select Installation Folder

Specify the target directory for the WibuKey tools. Click on <**Next**>. If the folder does not exist, the following message is displayed:

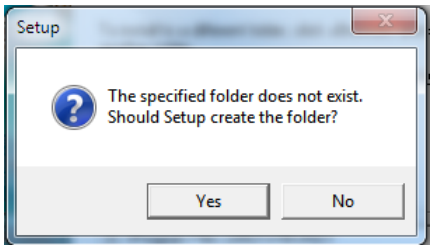


Figure 8: Create New Folder for WibuKey Installation

Accept to create the required folder with <**Yes**>.

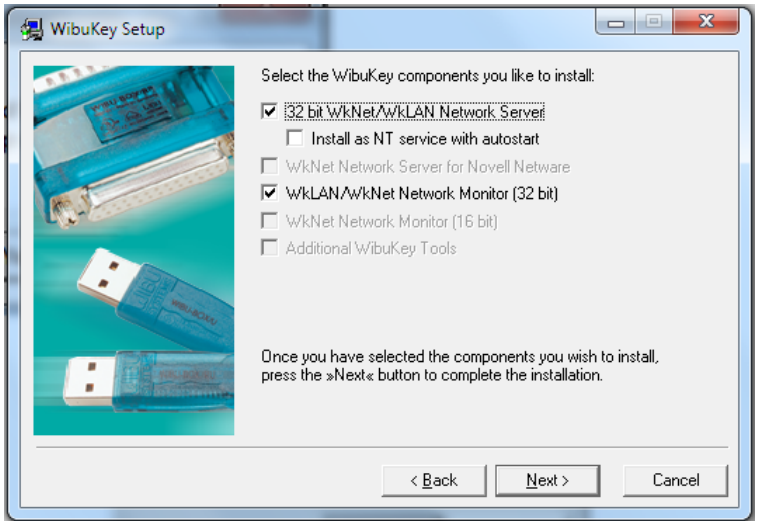


Figure 9: Select WibuKey Components

Click on <**Next**>.

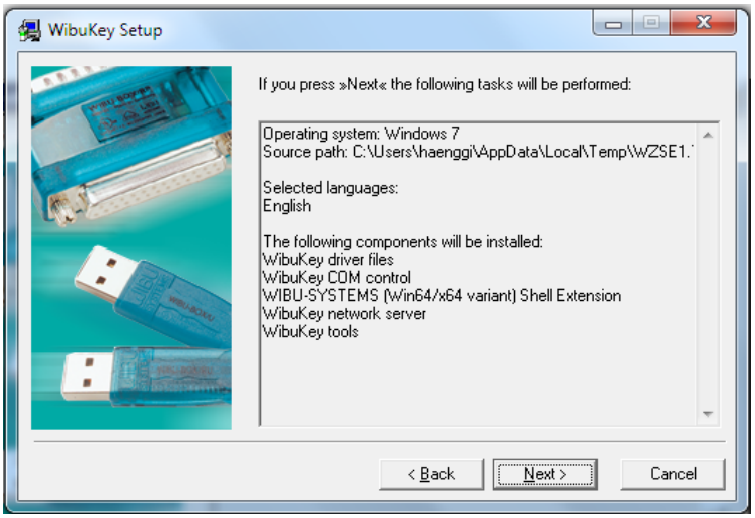


Figure 10: WibuKey Installation Tasks

Accept the installation tasks with a click on <**Next**>.

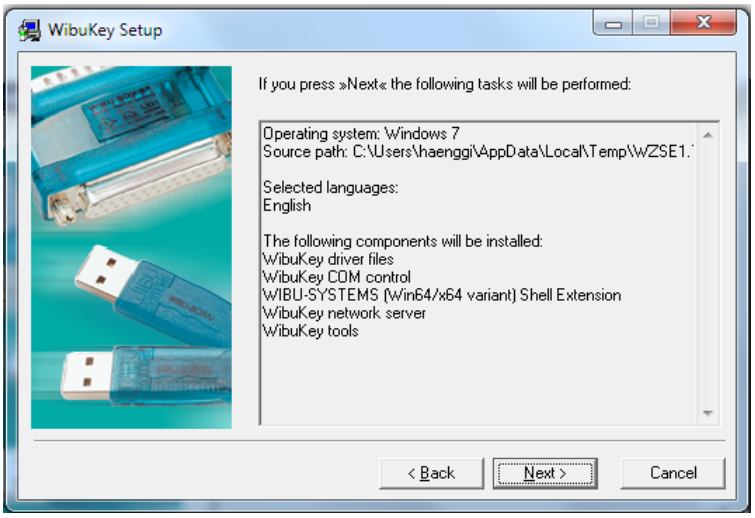


Figure 11: Confirmation Installation Tasks Finished

After all tasks have been finished, press <**Next**>.

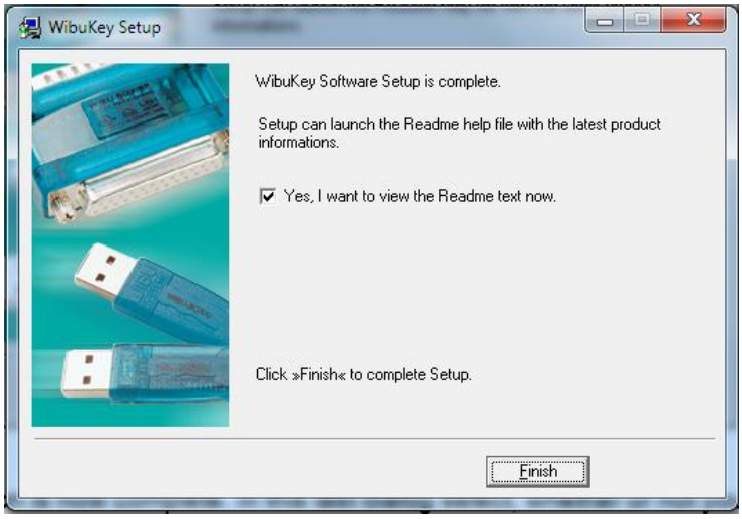


Figure 12: WibuKey Installation Finished

The WibuKey installation is now complete. In this last dialog select whether or not you want to read the help file after you clicked on <**Finish**>.

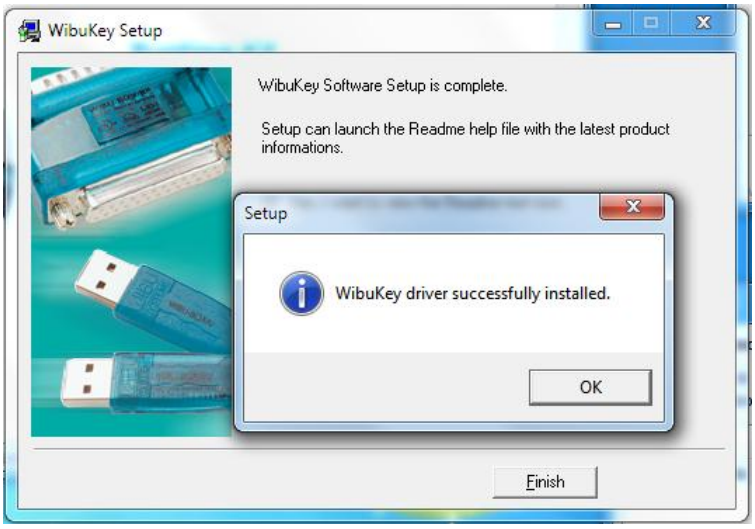


Figure 13: Help File of the WibuKey and Confirmation of Successful Installation

Click on <**OK**> to carry on with the installation of the go2ANALYSE application.

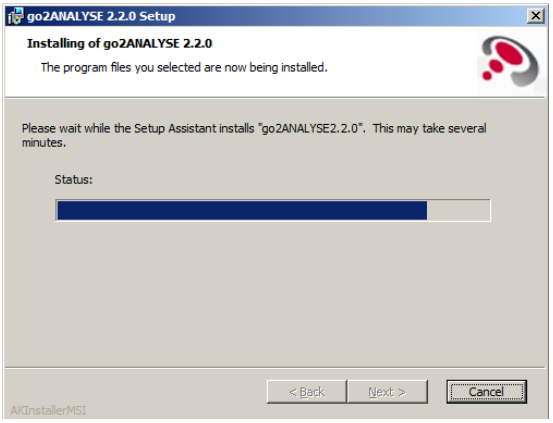


Figure 14: Progress of go2ANALYSE Installation

After successful installation the Setup Assistant displays the following message:

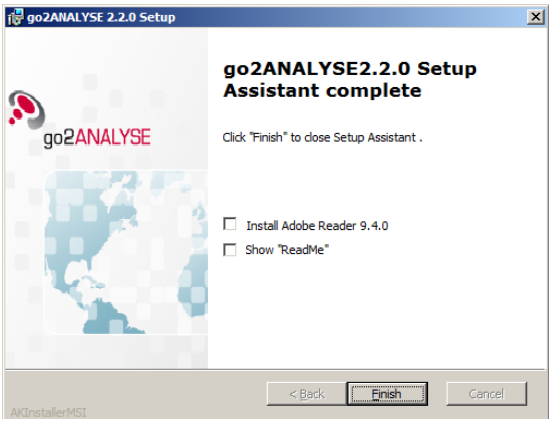


Figure 15: Successful Installation of the Application go2ANALYSE

Click on <**Finish**> to exit the setup.

Connecting the Dongle

Once the installation is finished connect the dongle you received to an USB port. In case the dongle has already been connected, remove it from the port and reconnect. You can run the application go2ANALYSE only with a connected dongle.

License

The dongle is part of the software's copy-protection. Through the combination of an USB-dongle and a license-file the application can be installed on more than one PC, but can at a given time run only on the PC to which the dongle is connected.

A key is coded into the USB-dongle. The license-file, with the extension *.maw*, holds information about the functionality which is available due to the license paid. It has to be installed on each PC on which the application is supposed to run. At run-time, the application compares the information coded into the license-file with the dongle and unblocks the selected software if they are consistent.

The license-file *default.maw* is located in the go2ANALYSE related subfolder of the User-directory.

```
C:\Users\<your user name>\go2SIGNALS\go2ANALYSE 2.2\default.maw
```

Hardware Locked

A license may be linked to a dedicated PC. The user has to support some hardware-related information about the PC which is coded into the license-file. While this license is restricted to a single PC, it eliminates the requirement of a dongle.

Uninstallation

- Select Uninstall in the go2ANALYSE program group of the start menu. As an alternative you may select the item Software in the System Control. Then select the item go2ANALYSE… and press <**Change/Remove**>. Follow the instructions on screen. Continue in the same way with the items Adobe…" and WIBU-KEY… To remove the program entirely, you will have to remove the installation directory manually (Default = c:\Program Files\ go2ANALYSE...).

The Application

Starting the Software

Start go2ANALYSEvia desktop icon or the operating system's taskbar.

Click the go2ANALY  icon:

The graphical user interface of the go2ANALYSE is shown without any display window when the go2ANALYSE is started for the first time after installation.

Overview

The figure below gives an overview of the go2ANALYSE interface, showing all displays available in go2ANALYSE:

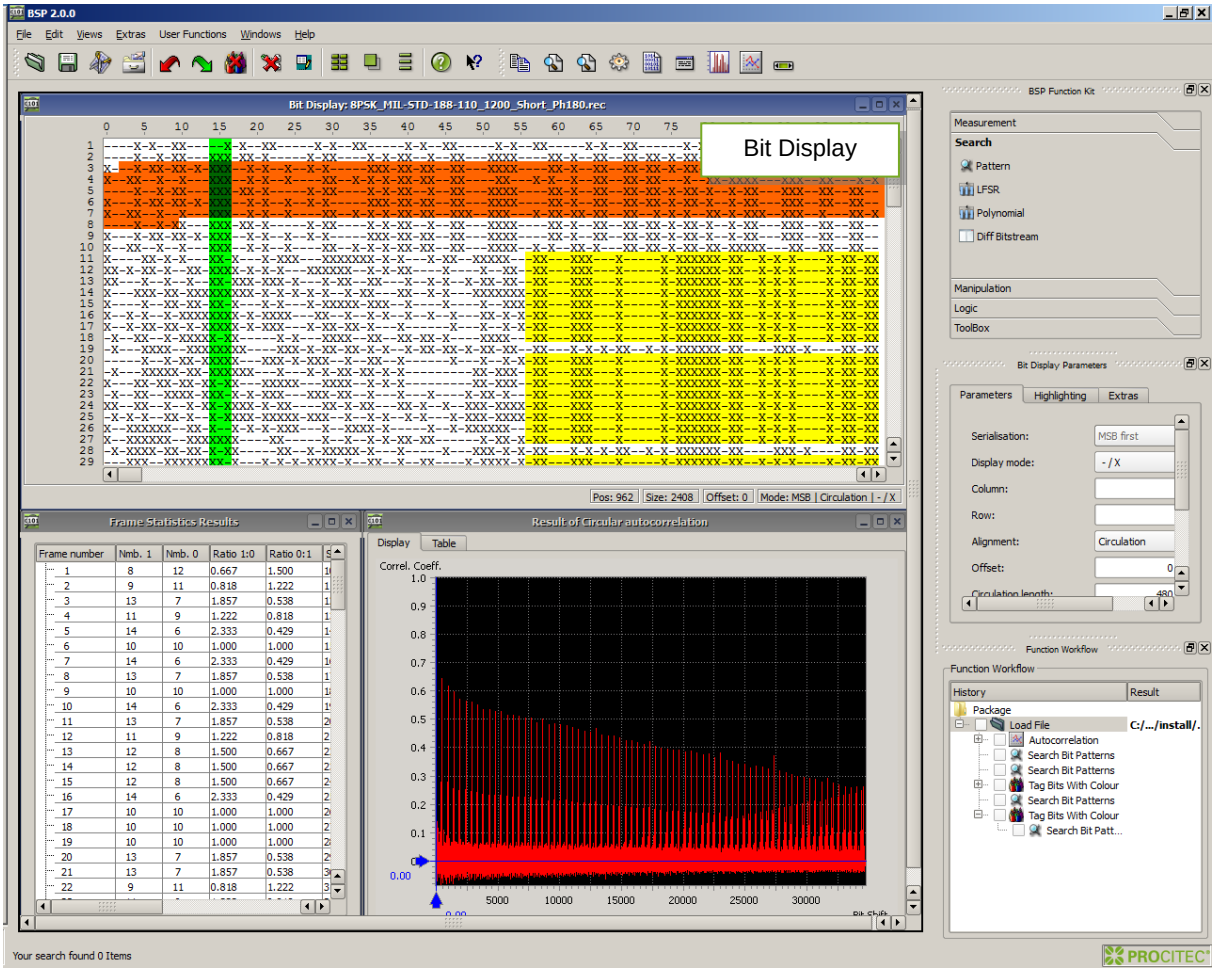


Figure 16: go2ANALYSE Interface

go2ANALYSE is divided into a main window with the menu bar, the toolbar and the docking area where the Function Kit resides, and a workspace where the various displays are inserted.

In the figure above all displays available are open.

- The *Bit Display* is the main display of the go2ANALYSE. It shows the current bitstream, is used for navigating in the bitstream and allows for changing of the bitstream visualization.
- The *Measurement Display* shows the result of measurement functions which have been applied to the bitstream, it allows for verifying the results and provides cursors.
- The *Result Display* shows the result of statistical functions and of the LFSR (linear feedback shift registers) sequence search, it lists the sorted results.
- Additional *Text Displays* (not shown here) may appear showing results of the User Functions.

All displays are explained in detail in separate chapters (Bit Display, Text Display, Measurement Display, Result Display).

Using Help

The go2ANALYSE opens the present manual after selecting <**F1**> or after clicking the icon.

You will be able to search any help information using the Adobe® Reader's built-in search function.

Help Features

For quick help the go2ANALYSE shows a tooltip to any button or text/spin box if the mouse cursor resides longer than approximately 3 seconds on this button or box.

The next image shows an example of a tooltip:

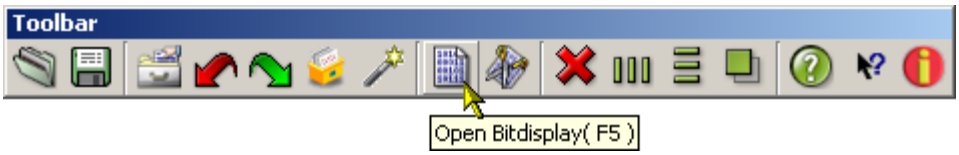


Figure 17: Toolbar with Tooltip for Bit Display Icon

All major features of the go2ANALYSE interface also have a *What's this?* help, as it is called. *What's this?* help is displayed on clicking a feature after pressing <**Shift**>+<**F1**> or on clicking the toolbar icon .

First Steps

Open Bitstream

Open the file to be analyzed by pressing the file open icon on the toolbar or choose *Open* from the menu *Files*. You may also open one of the previously processed files by selecting one of the file names listed below the *Exit* item in the menu *Files* as shown in the screenshot below:

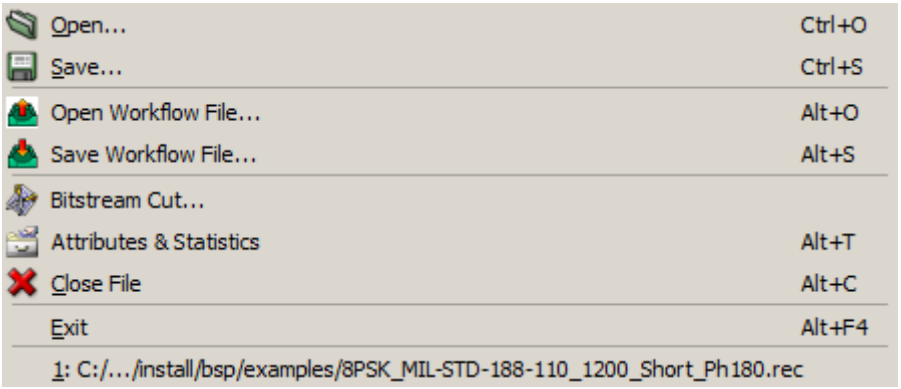


Figure 18: Files Menu with Selected File Entry

The standard file dialogue of the operating system is opened and you are prompted to choose a directory and a file.

There are three different file types for bitstreams to choose from: **.txt* based bitstream and **.rec* based bitstream, all other file types will be read as pure binary bit streams.

*Text-based Bitstreams: *.txt*

A **.txt* bitstream only contains the information about the bits in the bitstream and no extra information can be viewed when such a stream is loaded. go2ANALYSE converts the characters of a **.txt* stream using the following table - any other characters will be ignored!

Character	Converted Bit
0	0
-(Dash)	0
_(Underline)	0

Character	Converted Bit
L	0
. (Dot)	0
1	1
X	1
x	1
H	1

Table 1: Text Based Bitstream Conversion

*Record-Based Bitstreams: *.rec*

go2ANALYSE, go2DECODE and APC/SDA are able to record demodulator outputs containing the full set of information, which is used be succeeding decoders. Besides the bitstream itself this is symbol qualities, symbol times, symbol bounds, burst bounds etc. Some go2ANALYSE functions rely on the availability of these additional information.

*Binary Bitstreams: *.**

Any other file, i.e. a file with neither *.txt* nor *.rec* extension, will be read as a pure binary bitstream in bytewise order.

Cut Bitstream (Partial Bitstream)

With go2ANALYSE large *.rec* files can be resized. To do so, open the *Partial Bitstream* dialogue box by selecting the menu item *Files – Bitstream Cut*. In the *Open File* dialogue box displayed, choose the desired *.rec* file and press the button <**Open**>. go2ANALYSE shows the following dialogue box:

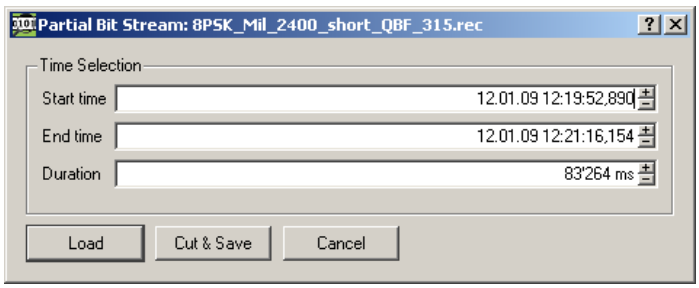


Figure 19: Partial Bitstream Dialogue Box

Edit the spin boxes for *Start Time, End Time* or *Duration,* if desired, and press the button <**Load**>. go2ANALYSE shows the bitstream section in the *Bit Display* in accordance with the time stamp selected between start and end time. Save the partial bitstream by use of the button <**Cut & Save**> and the *Save As* dialogue box displayed.

Note: Repeat loading the bitstream until the desired partial bitstream has been selected.

Navigate within Bitstream

To navigate within the bitstream, move the scrollbars of the *Bit Display* or edit the parameters *row* and *column* in the parameter display. If no scrollbars are visible, the display shows the entire bitstream.

Changing the parameter *circulation length* (wrap length) will change the alignment of the bits, thus patterns occurring periodically will be visible.

Example: The first bitstream is displayed with a circulation length of 51 and no structure or periods are visible:

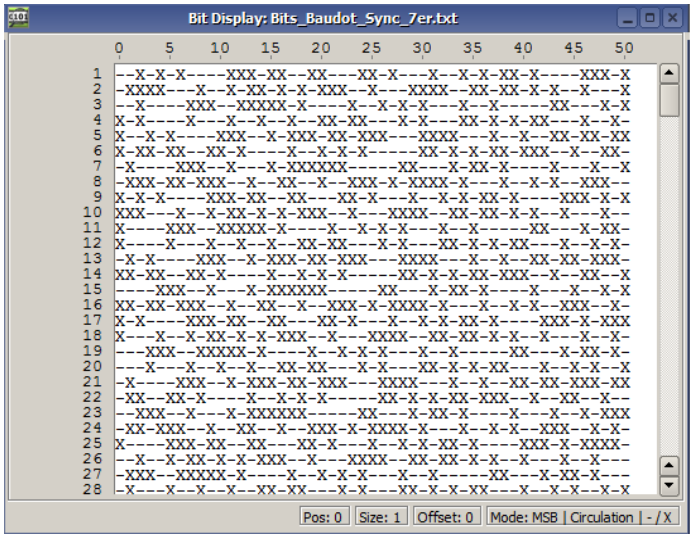


Figure 20: Bitstream with Circulation Length of 51

After changing the circulation length to 49, the same stream looks as follows:

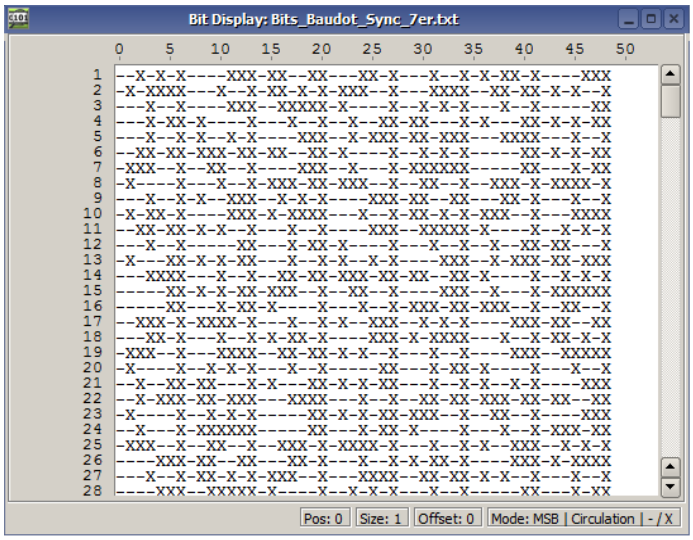


Figure 21: Same Stream as above with Circulation Length now 49

Basic Editing Features

go2ANALYSE provides editing features for bitstreams known from typical text editors. These features include *Copy, Paste, Insert* and *Delete* of bits.

To copy some bits to the clipboard highlight them with the mouse, click the right mouse button and select *Copy* from the popup menu.

To insert the bits into the stream, select (highlight) the position at which the bits are to be inserted, click the right mouse button and choose *Insert*. The bits are written into the stream right before the highlighted/selected bits, the selected bits are not replaced by the insertion.

Note: The *Bit Display* has no cursor as a usual text editor. To select the position at which to insert the bits you have to make a selection of at least one bit with the mouse.

If you choose *Paste* instead of *Insert*, the selected bits of the bitstream are replaced with the bits from the clipboard.

go2ANALYSE also provides editing functions which are uncommon to text editors but basic features for a bit editor such as inversion, mirroring and tagging of interesting bit sequences.

To invert the bitstream (inversion means changing the value of each bit from 0 to 1 and from 1 to 0) click the <**Not**> button on the *function kit* window (for details see screenshot in chapter Overview).

After pressing the button <**Not**>, the following dialogue box is shown:

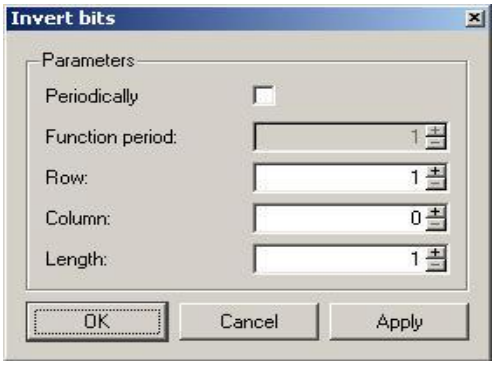


Figure 22: Exemplary Dialogue Box

Check ☑ *Periodically* and click <**OK**>: The entire bitstream is inverted.

Note: When pressing <**Apply**> instead of <**OK**>, the current dialogue box will remain visible. This will facilitate the repeating of functions.

The *Tag Bits* function serves to highlight interesting bit sequences in a chosen color. The tag remains on the bit until it is cleared by the *Clear Tags* function, replaced by another tag or until the bitstream is closed.

The *Mirror* function swaps the bits defined by the parameter of the function.

For details to *Tag Bits*, *Clear Tags* and *Mirror* see the corresponding chapters.

Define Bits for Application of go2ANALYSE Functions

Most of the built-in functions of go2ANALYSE cannot be applied until you click the adequate button in the function kit. The click always causes the dialogue box to be displayed (see previous chapter).

This dialogue displays the current function parameters before the function is applied to the stream. The function parameters can be changed to define exactly what the function should do and, even more important, to determine exactly which bits the function is to be applied to.

How do these parameters determine which bits the function is applied to?

Example: Assume the following bitstream with a circulation length of 8:

Row \ Column	0	1	2	3	4	5	6	7
1	0	0	1	1	0	0	1	1
2	0	1	1	1	0	0	0	1
3	0	0	0	1	1	1	0	1
4	1	1	1	1	0	1	0	1
5	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	1	1

Figure 23: Exemplary Bitstream

The rows are numbered from 1 to 6, the columns from 0 to 7.

To apply the function to the bits in row 1 in columns 2 to 4 (bit 4 should be included) enter the following parameters:

Row 1, Column 2, Length 3

To apply the function to the bits in rows 1 to 6 and columns 2 to 4 check ☑ *Periodically* and set *Function Period* to 8.

To apply the function to the bits on squared background in the next figure enter the following parameters:

Row 1, Column 0, Length 1 and Function Period 9. ☑ Periodically must be checked.

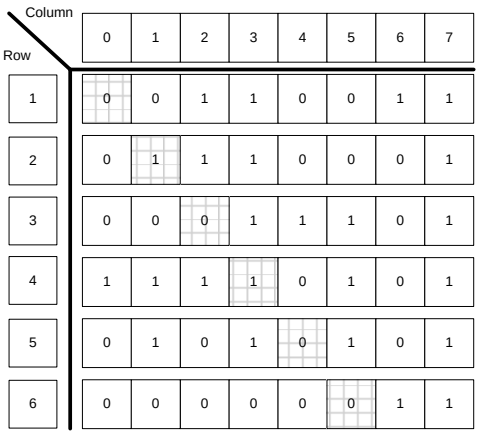


Figure 24: Example of Periodic Function Mode

In the following, a function has been defined to work in periodic mode only if it is executed periodically on a stream.

Note: Some functions always work on the entire bitstream. In this case the dialogue box does not provide the parameters r*ow, column, length, function period* and the checkbox ☑ *Periodically,* or these parameters will be disabled.

Disabled parameters are easy to recognize because they are greyed as shown in the following screenshot:

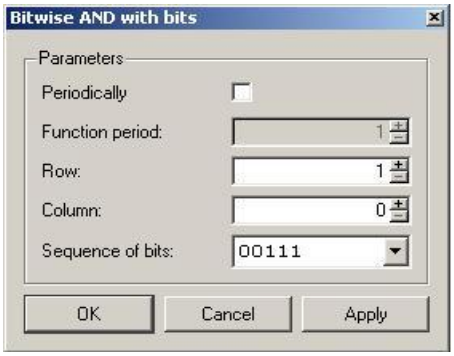


Figure 25: Dialogue Box for AND Operation with Disabled Function Period Parameter

These function parameters will be enabled by checking an option or changing the value of another parameter.

Example: Look at the screenshot above. On checking ☑ *Periodically* the parameter *Function period* will be enabled. By unchecking this option, the parameter will be disabled again.

Fundamental: It is possible to select a function period which is lower than length. If you choose a function period lower than length, be aware that the function is applied to the same bits twice or several times!

Undo Changes to the Bitstream

To undo your last action which affected the bitstream either

- click the *Undo* icon or
- choose the menu item *Undo* in the *Edit* menu or

- press CTRL + Z

go2ANALYSE recovers the bitstream to the state prior to the last action.

go2ANALYSE is only able to undo the last ten actions affecting the bitstream.

Redo Changes Undone

To redo the last action undone either

- click the *Redo* Icon or
- chose the item *Redo* from the *Edit* menu or
- press CTRL + Y

go2ANALYSE recovers the bitstream to the state prior to undoing the last action. It is not possible to redo more than the undone steps.

Adjust Bitstream View

go2ANALYSE provides various visualization modes for the bitstream. The bitstream view can be changed by changing the parameter *Display type* (on the parameter display of the *Bit Display*).

These are the different display types:

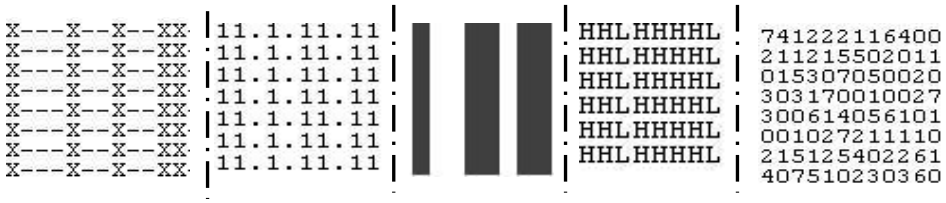


Figure 26: Bitstream View Display Types From Left to Right: -/X; ./1; White/Black; L/H and Symbol Numbers

If *Display type* is set to symbol numbers the *Bit Display* shows the values of the symbols of the bitstream. Any **.txt*-based bitstream will only show the values 0 and 1 if *Display type* is set to symbol numbers. Only **.rec*-based bitstreams will show greater symbol values because the **.rec* files also store information on whether the stream was demodulated with a multi-valued demodulator.

Note: The area of the *Bit Display* where no bits are displayed is green if the display type chosen is *White/Black*. If it were displayed in white, it would be impossible to distinguish the area without bits from an area with only zeros.

go2ANALYSE also provides the option to change the size in which the bits are displayed. Choose the correct font (bit) size by clicking the <*Extras*> tab in the parameter display of the *Bit Display* and change the font (bit size).

When processing a record file-based bitstream, go2ANALYSE provides more options to adjust the view of the bitstream.

View Bursts

Proceed recorded bitstreams (**.rec*-files) allow burst mode signals for viewing each complete burst in one line.

To do so, choose the entry *Burst* in the parameter *Alignment.*

Normally the parameter *Alignment* is set to *Circulation*. In this mode the bits (or characters respectively) shown are aligned vertically in rows. The length of each row is set to the current value of the parameter *Circulation length*, this way “frames” become visible if the value of circulation length corresponds to the repetition rate.

If *Alignment* is set to *Burst,* each row shows one burst to a maximum length of 10,000 bits.

The length of the rows may differ from each other. See the next figure:

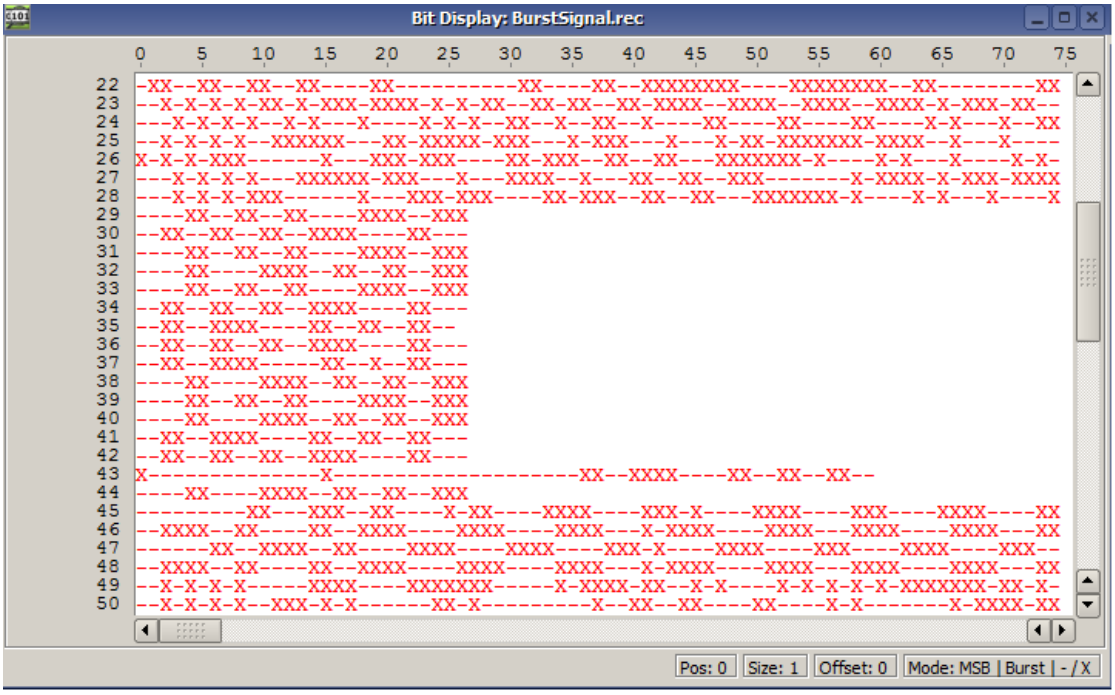


Figure 27: Bitstream with Different Burst Lengths shown with Alignment Burst

View Quality Information

Record-based bitstreams also allow for viewing quality information of each stream symbol. The demodulator which produced the record file has produced quality information for each stream symbol. This means that if the communication modem uses single valued symbols (in this case a symbol is equal to one bit) for broadcasting, the record file stores quality information for each bit, otherwise if multi-valued symbols are used, the same bit of quality information is paired to several bits.

To do so, choose the option ☑ *Show Quality* from the <*Extras*> tab of the *Bit Display* property sheet. After this option is selected the background of each bit is drawn in a scale of grey. The lighter the background of the bit, the better is the quality of this bit.

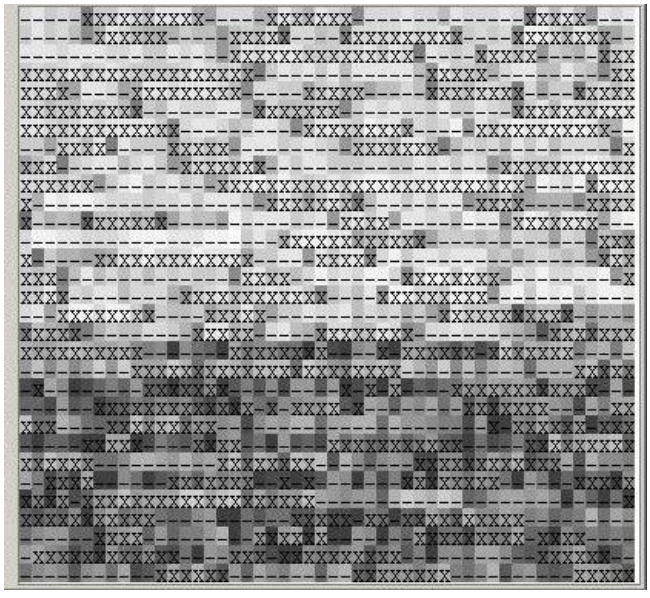


Figure 28: Bitstream with Quality Information

Sixteen different shades of grey are defined for displaying the quality. They range from white to dark grey.

When trying to display a **.txt*-based bitstream with the ☑ *Show Quality* option all bits are displayed with a white background, because **.txt*-based bitstreams do not provide any quality information.

View Bitstream as Text

go2ANALYSE provides two different ways to view a bitstream as code symbols of a predefined code table.

View Bitstream as Text Using Bit Display

One option to view a bitstream as code symbols is provided by the *Bit Display*:

The *Bit Display* has built-in code tables which allows for viewing the bitstream as code symbols of the chosen code table.

To do so, choose the code table the bitstream is to be coded to by the parameter *Code table*. After choosing the appropriate code table check the parameter ☑ *Bits to Code* symbols.

Example:

Open the bitstream *ExampleCodeSymb*. Choose *Baudot* as code table and check the parameter ☑ *Bits to Code Symbols.* Now change the circulation length to 32.The text displayed in the *Bit Display* starts with: *rown fox jumps over the lazy dog.*

View Bitstream as Text Using Text Display

To view bits as text go2ANALYSE provides the option to map the bits into text using a code table. The output of this mapping function is displayed in the *Text Display*.

The *Text Display* opens after activating the <**Map bits to text**> button in the function kit window. It shows the complete stream in one line. By default the *Text Display* uses the familiar -&X code table. On changing the parameter *code table* the bits are converted to text using the chosen code table.

For detailed information on using the *Text Display* please refer to chapter Text Display.

Example:

The *Bit Display* shows the stream *ExampleCodeSymb*. After the start of the *Text Display*, this stream is shown as –X. Now choose the code table *Baudot*. The display will show the following result:

Figure 29: Bitstream ExampleCodeSymb shown in the Text Display with Code Table Baudot

In order to use other code tables or define your own code tables see chapter Code Tables for details.

Customize Workspace

When opening the first bitstream after the installation, the *Function Kit* window and the *Parameter Display* will be docked to the right side of the main window (we say they are docked in the right docking area) and the toolbar will be located below the menu bar.

In order to change the appearance of the workspace, go2ANALYSE provides the following customizing options:

- To undock a docked window (i.e. the toolbar, the *Function Kit* or the *Parameter Display* at first) double click the dock window you wish to undock.
- To re-dock any undocked window simply double click the undocked window, it will be docked in the docking area it was docked in last.
- To dock any undocked window in another docking area (left or right side), move the window to the right or left as far as you can and release the left mouse button.
- To close a docked window, choose the *Dockwindows* command in the *Views* menu and click the entry for the dock window you wish to close.
- If you closed a dock window by opening the menu *Views*, choose the *Dockwindows* command and select the entry for the dock window you wish to be visible again.
- The toolbar can be undocked as well. It is also possible to close the toolbar or to dock it on the right or left side of the main window.
- The dialogue box which is always displayed when pressing a button of the *Function Kit* can also be docked to the right or left side. If this dialogue is docked it will not be closed on pressing the <**OK**> button of this dialogue but remains visible.

Note: The docking areas of the main window are located on the right or left side resp. of the main window. Only the toolbar can be docked below the menu bar. Displays such as the *Bit Display* or the *Measurement Display* cannot be docked.

go2ANALYSE will store the current appearance of the workspace at the end of each session.

When starting go2ANALYSE the following settings will be restored:

- The positions of all dock windows except the toolbar
- The list of recent files
- The *Bit Display* with the file you worked on last (if any)

Save Bitstream

go2ANALYSE provides three options to save changes in the bitstream.

Save via File Menu

The first way to save the current bitstream is to select *Save* from the *File* menu, enter a path and a name for the file in the file dialogue shown and press the button <**Save**> in the file dialogue. The current bitstream is saved to the chosen path with the file name entered. If the file name was changed, the new file name is displayed in the caption of the *Bit Display* and in the caption of the go2ANALYSE main window.

The file format for bitstreams can be text file (*.txt)* and decoder input format (*.rec)* or binary format (any other extension).

Save via Popup Menu of Bit Display

The second way to save the current bitstream is to select *Save All* in the popup menu of the *Bit Display*. The popup menu becomes visible on clicking the right mouse button inside the *Bit Display*. Having pressed *Save All,* enter a path and a name for the file in the file dialogue shown and press the button <**Save**> in the file dialogue. The current bitstream is now saved to the chosen path with the file name entered. If the file name was changed, the new file name is displayed in the caption of the *Bit Display* and in the caption of the go2ANALYSE main window.

Save Section of Bitstream

The third way to save the changes to the current bitstream is to select *Save* from the popup menu of the *Bit Display*.

Selecting *Save* serves to save only part of the bitstream or to save the bitstream as code symbols of the current code table.

To save only part of the bitstream, proceed as follows:

Select (highlight) the area of bits to save with the mouse (press left button and move the mouse up or down). Having selected all bits of interest release the mouse button and click the right mouse button. Select *Save out* from the popup menu. Having pressed <**Save**>, proceed as outlined in the paragraph above.

Note that the captions of the *Bit Display* and the main window will not change even if a new file name has been entered because only part of the bitstream was saved. Having saved part of the bitstream, the *Bit Display* still shows the original stream.

Save Bitstream as Code Symbols

go2ANALYSE provides two ways to save the bitstream coded as symbols:

When using the *Bit Display* for mapping the bits to code symbols you can save the shown symbols as follows:

Select the area of “text” you wish to save, and choose the *Save* command in the popup menu (see above for details), the procedure is described in paragraph Save via File MenuSave via File Menu. The current bitstream is saved as code symbols of the chosen code table.

When using the *Text Display* to view the bitstream coded to code symbols you can save the symbols shown by clicking the <**Save text as…**> button on the *Text Display* property sheet.

After clicking this button go2ANALYSE opens a file dialogue to specify the file type of the text you wish to save by means of the file name extension.

File-Extension	Data-Type
.txt	Plain ASCII text
.rec	Binary data
.utf8	UTF-8 based text file with Unicode characters

Table 2: Text-File Data-Types

Attributes and Statistics

For information on the last changes to the current bitstream or the number of ones in this stream, use the *Attributes and Statistics* command in the *File* menu.

On selection of this menu item, go2ANALYSE will show this dialogue:

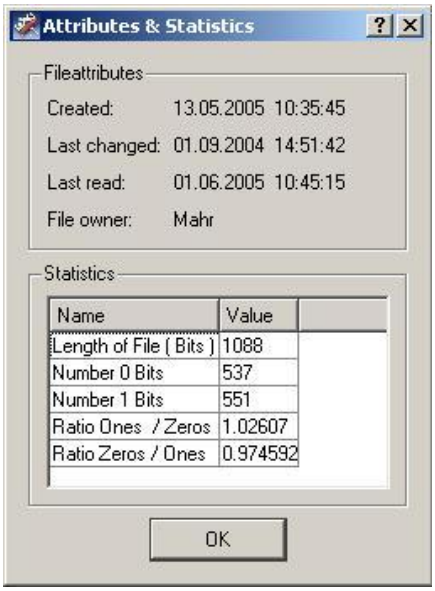


Figure 30: Attributes and Statistics

The dialogue shows:

- Create date and time of the file (bitstream)
- Last change date and time
- Last read date and time of the file (bitstream)
- Current owner of the file.
- Length of current bitstream in bits
- Total number of ones in the bitstream
- Total number of zeros in the bitstream
- Ratio of ones to zeros
- The reversal, i.e. the ratio of zeros to ones

Note: If the file is modified while this dialogue is open, the statistics displayed will not be updated. Update the statistics by closing this dialogue after a change to the stream and opening it again using <**Alt**>+<**T**>.

Close File and Terminate go2ANALYSE

To close the current bitstream choose *Close File* from the *File* menu or press <**Alt**>+<**C**>.

All displays associated to a specific bitstream are closed along with the file.

To terminate go2ANALYSE click the *Close* icon of the main window (cross in the upper right corner ☒) or choose the item *Exit* on the *File* menu or press <**Alt**>+<**F4**>.

In case the bitstream was modified before terminating go2ANALYSE or closing the file, you will be asked to save the modifications. If this message is acknowledged, the bitstream will be saved otherwise all changes will be lost.

Bit Display

The *Bit Display* can show bits or code symbols. A code symbol is a letter or a figure from a code table, representing the serialized sequence of bits.

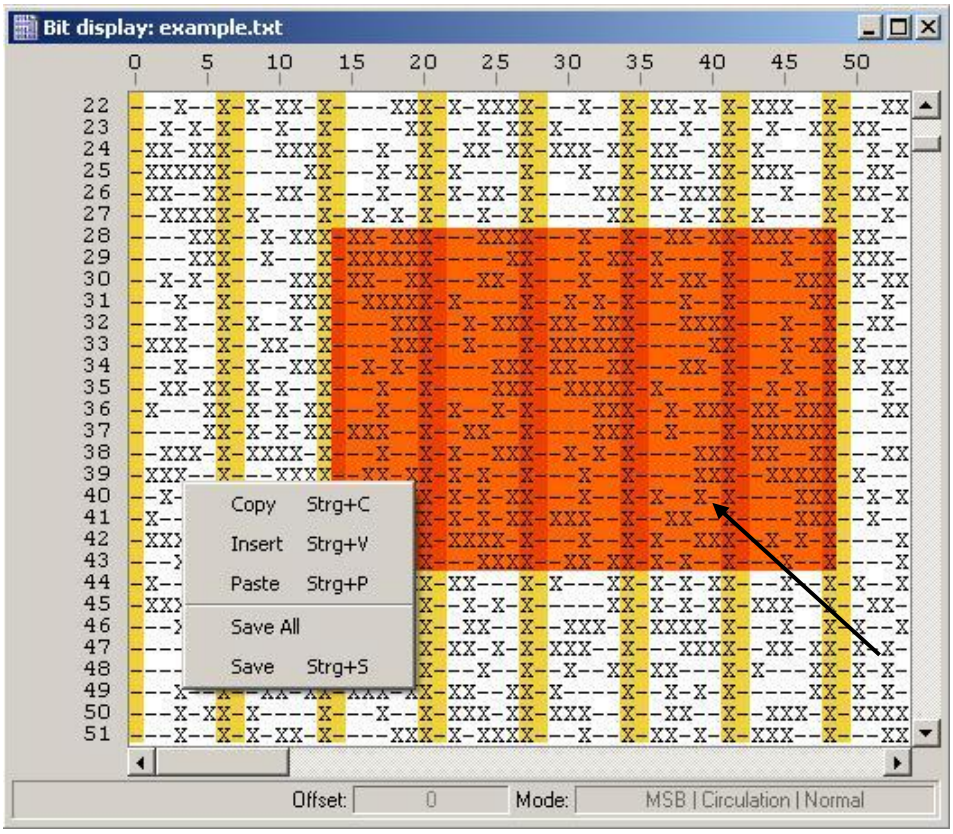


Figure 31: Exemplary Bit Display

The X-axis shows the current column of the bitstream, the Y-axis the current row.

The *Bit Display* allows for an overview of sequences of symbols. It is used for the analysis of repeating bit patterns. Parts of the bit patterns can be highlighted. The highlighted part of the bit pattern or all bits shown in the *Bit Display* can be saved in one file.

The scrollbars are only visible if the bitstream is too large to be fully displayed within the current size of the *Bit Display*.

The status bars shows the current mode and the current offset in bits.

If the parameter serialization is changed, *Mode* shows the current selection as *MSB* or *LSB*.

Parameters

The *Bit Display* parameters are displayed by activating the <*Parameters*> tab. The following parameters are available:

Parameter	Valid Range	Function
Column	0 -	Sets the start column of the displayed bitstream. A change of column is only possible if the current circulation length is greater than the visible section of the Bit Display. If column is increased by one the bits are “shifted” one column to the left, if it is decreased, the bits are “shifted” to the right. The minimum value of column is zero, the maximum value depends on the current circulation length.
Row	1 -	Sets the start row of the displayed bitstream. A change of row is only possible if the bitstream at the current circulation length has more rows than the visible section of the Bit Display. If row is increased the bits are “scrolled” upwards by one row, if it is decreased, the bits are “scrolled” downwards to the right. The minimum value of row is one, the maximum value depends on the length of the bitstream and on the current cir-

Parameter	Valid Range	Function
		culation length.
Offset	0 - 1000	Defines the number of the first bit displayed in the stream from which the bit-stream is displayed. If Offset is > 0 then the first offset bits will not be displayed or serialized.
Circulation Length	2 - 10000	Defines length of a row in bits/code symbols. Changing the circulation length changes the alignment of the bit/ code symbols which allows for a visualization of repeating bit patterns. If the circulation length is greater than the visible section of the Bit Display, the invisible section can be displayed by changing column or by means of the horizontal scrollbar.
Serialisation	LSB first MSB first	The combination mode of the bits to code symbols to is set by the following parameter: Mode — Description LSB first — The least significant bit, (the leftmost bit) has the highest bit of the code symbol. MSB first — The most significant bit (the rightmost bit) has the highest bit of the code symbol.
Display Mode	- / X . / 1 White / Black L / H Symbols	This parameter sets the mode in which the bits are displayed. When the bits are displayed the character left of the "/" stands for bit 0 and the one on the right of the /" stands for bit 1. Mode — Description – / X — Displays the zeros as"–" and the ones as "X" . / 1 — Displays the zeros as "."and the ones as "1" White / Black — Displays the zeros in white and ones in black squares L / H — Displays the zeros as "L" and the ones as "H"
Alignment	Circulation Burst	Changes the mode for defining the length of a row. Mode — Description Circulation — The parameter circulation defines the length of a row. Burst — The length of a burst (in symbols) defines the length of a row. Bursts are limited to a max. length of 10,000 bits.
☑ Bit to code symbols	unchecked / checked	If this option is checked, the bits will be displayed as code symbols using the current code table.
☑ Start with figure shift	unchecked / checked	If this option is checked the display will assume that FIGURESHIFT was set prior to the start of coding the bits to code symbols.
Figures – Letters only	Off Figures Only Letters Only	This parameter changes the mode of coding the bits to code symbols. Mode — Description Off — Figure shifts and letter shifts are allowed for coding Figures only — All bits are coded to figures of the current code table, no letters are displayed Letters only — All bits are coded to letters of the current code table, no figures are displayed
Code Table	Baudot ASCII CCIR 476 ITA 3	Determines which code table will be used for coding the bits to code symbols. These code tables may be selected individually: See chapter Built-In Code Tables for details.

Parameter	Valid Range	Function
	ITA 2P Hex	

Table 3: Bit Display Parameters

Select Bits (Highlighting Function)

The parameters of the current bit selection (the highlighted area) are displayed and changed by activating the <*Mark*> tab.

Use the mouse to change the current selection:

- Move the mouse cursor to the start position of the section you wish to select (highlight).
- Press the left mouse button.
- Drag the selection to the end position of the section, keeping the left mouse button pressed.
- After releasing the mouse button, the desired section is highlighted.

Use the displayed parameters to change the current selection:

- When increasing the start below column, the selected area will shrink towards the right of the *Bit Display*, or when decreasing it will increase towards the left.
- When increasing the end below column, the selected area will increase towards the right of the *Bit Display*, or when decreasing it, it will shrink towards the left.
- When increasing the start below row, the selected area will shrink towards the end of the *Bit Display* row by row, or when decreasing it, it will increase towards the beginning.
- When changing the end below row, the selected area will increase towards the end of the *Bit Display* row by row, or when decreasing it, it will shrink towards the beginning.

The end row is always greater than the start row, the same rule applies to *column*.

Use the right mouse button to open the popup menu in the *Bit Display*. This will enable you to

- Copy the selected bits to the clipboard
- Paste previously copied bits to the bitstream
- Insert previously copied bits to the bitstream
- Save all bits in a file
- Save the selected bits or code symbols in a file

Extras

The font size or bit size of the *Bit Display* can be edited activating the <*Extras*> tab, the valid range is 6 to 20 points.

Text Display

The *Text Display* can display bits or code symbols. A code symbol is a letter or a figure from a code table, it represents the serialized sequence of bits.

It is possible to synchronize the *Text Display* with the stream currently shown in the *Bit Display*.

In synchronized mode, the part of the bitstream currently shown in the *Bit Display* is sent to the *Text Display* where it is mapped to text. This function allows for tracking the effects of changes made to the bitstream to the text.

Figure 32: Exemplary Text Display

Parameters

Parameter	Valid Range	Function
Serialisation	LSB first MSB first	The combination mode of the bits to code symbols to is set by the following parameters: Mode — Description LSB first — The least significant bit (the leftmost bit) has the highest bit of the code symbol. MSB first — The most significant bit (the rightmost bit) has the highest bit of the code symbol.
Bitstream	Normal Inverse	Mode — Description Normal — All bits are serialized straight into a code symbol. Inverse — All bits are inversed prior to be serialized into a code symbol.
Code Table	-&X …	Determines the code table to use for coding the bits to code symbols. These code tables may be selected individually: See chapter Built-In Code Tables for details.
Force Level	Off Level 1 … Level n	You can change the mode of coding the bits to code symbols. Mode — Description Off — Figure shifts and letter shifts are allowed for coding Level 1 — All bits are coded to symbols from Level 1 of the current code table. No other symbols are displayed. Level N — All bits are coded to symbols from Level N of the current code table. No other symbols are displayed.
Start Level	Level 1 … Level n	If this option is checked the display assumes that a level shift to the chosen level has been set prior to the start of coding the bits to code symbols.
Sync Mode	On / Off	On activation, the Text Display is synchronised with the Bit Display. A change of certain parameters of the Bit Display will also affect the Text Display. The parameters are: Circulation length

Parameter	Valid Range	Function
		Offset Font size Row Column If any bits are affected by a bit-editing function the text in the Text Display will be affected as well.
Save Text As…		On activation, the display text can be saved either as ASCII based txt file or as Unicode Transfer Format 8 (UTF8) based file.

Table 4: Text Display Parameters

All parameters except *Save text as…* are disabled if the *Text Display* is used to view the output of a decoder.

Text Wrapping Parameters

Text wrapping can be changed by activating the <*Wrapping*> tab.

Parameter	Valid Range	Function
Mode	Word wrapping Column width	The combination mode of the bits to code symbols to is set by the following parameters: Mode: Word wrapping — Wraps the text at the current width of the display (this is the default). Wrapping is at blanks by default, this can be changed by means of Policy . Mode: Column Width — Wraps the text at a fixed number of character columns from the display's left. Wrapping is at blanks by default, this can be changed by the means of Policy.
Column Width	1 -	Sets the position (in characters) where text will be wrapped. Unless the policy setting is Anywhere the text will be wrapped at the nearest blank to the current column width.
Policy	Word Boundary Anywhere Word or Document Boundary	Mode: Word Boundary — Break lines at word boundaries, e.g. spaces or new lines. Mode: Anywhere — Breaks the text anywhere also within words. Mode: Word or Document Boundary — Break lines at blanks, e.g. spaces or new lines if possible. Break it anywhere otherwise.

Table 5: Text Wrapping Parameters

Extras

The font size of a bit in the Bit Display can be changed on the <*Extras*> tab, the valid range is 6 to 20 points.

Measurement Features

Autocorrelation and Partial Autocorrelation

The button <**Autocorrelation**> in the Function Kit serves to apply an autocorrelation or a partial autocorrelation to the bitstream.

When selected, the following dialogue box is displayed:

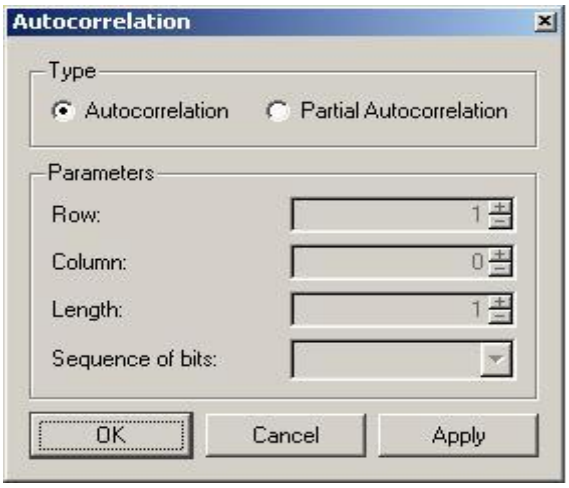


Figure 33: Dialogue Box for Autocorrelation Functions

Two types of autocorrelation are available.

Select ⊙ *Autocorrelation* to execute a circular autocorrelation of the bits after clicking <**OK**> or <**Apply**>.

Select ⊙ *Partial Autocorrelation* to enable the spin boxes for *Row, Column, Length* and *Sequence of Bits*. If the box *Sequence of Bits* is blank, the software correlates the bits that are part of the frame described by the parameters *Row, Column* and *Length* with the bitstream, otherwise the entered bit sequence is correlated with the current bitstream. Partial autocorrelation is non-circular!

Circular and Non-Circular Autocorrelation

Circular autocorrelation requires creating a duplicate of the original bitstream.

The duplicate is correlated with the original bitstream and then shifted by one bit towards the end of the stream. The last bit of the duplicate before the shift is wrapped and inserted as the first bit of the duplicate after the shift.

The result of the autocorrelation is calculated as

(Sum of consistent bits – Sum of inconsistent bits) divided by the sum of all bits.

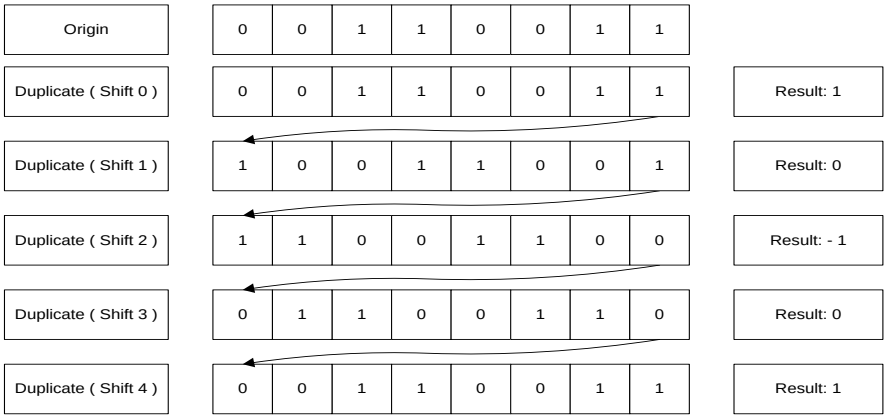


Figure 34: Circular Autocorrelation

The results of circular autocorrelation are repeated if the duplicate is shifted by more than half the length of the bitstream, without any additional information.

A non-circular autocorrelation will not duplicate the entire bitstream. It correlates the bitstream only with part of itself or with a bit sequence entered by the user. Said part, or the entered bit sequence respectively (assume that either of them is N bits long), is correlated with part of the original bitstream with same length.

After the correlation for shift 0, the part of the *original* bitstream is shifted by 1 bit. Now the former first bit is deleted, all other bits are shifted towards the beginning of the part and the last bit is removed from the original stream. See next figure (the bitstream is the same as in the example above):

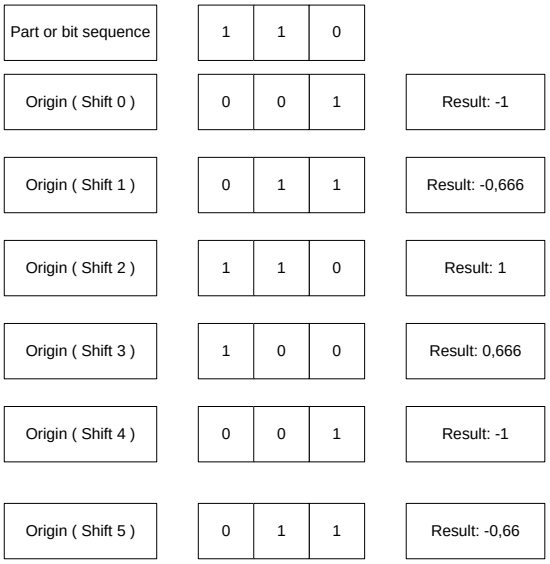


Figure 35: Non-Circular Autocorrelation

The results of the non-circular autocorrelation will not be repeated if the duplicate is shifted by more than half the length of the bitstream.

Note: go2ANALYSE restricts the use of autocorrelation to bitstreams with a size greater than 32 bits.

Example:

Apply the autocorrelation to the file *LFSR-Sequence1:*

Open the file, click the radio button ⊙ *Autocorrelation* and click the <**OK**> button in the dialogue box.

go2ANALYSE first displays a progress dialogue box which serves to control the progress of the autocorrelation and to cancel its further execution.

To abort the calculation of the autocorrelation, select <**Cancel**>.

Once the progress dialogue box is closed, go2ANALYSE opens a *Measurement Display* (unless already displayed) and shows the result:

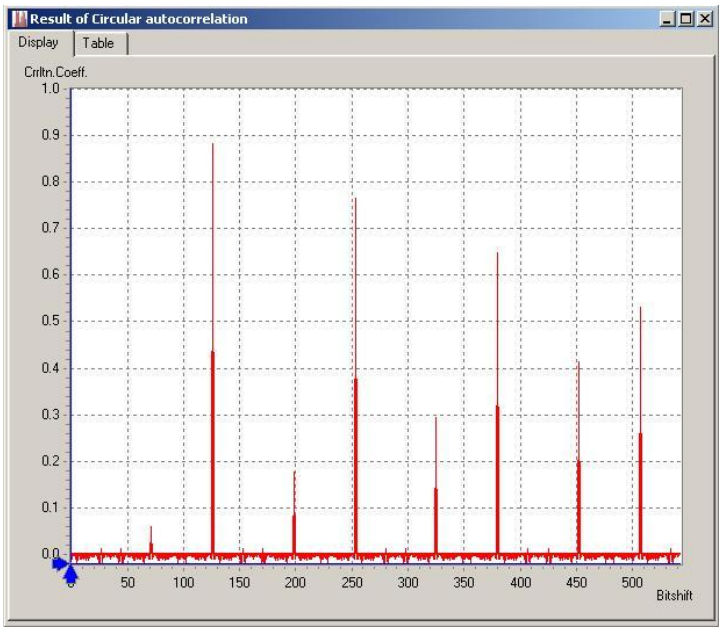


Figure 36: Exemplary Autocorrelation Results

(Please ignore the white background in the picture. It is only used for enhancing the visibility in the Manual. Default background is black).

Significant positive peaks indicate a high self-similarity of the bitstream with its duplicate after the attributed number of shifts.

When using circular autocorrelation, the peak at shift 0 must be ignored. The duplicate at shift 0 is a copy of the origin, so the peak at shift 0 is always 1.

Significant negative peaks indicate an inverse similarity of the bitstream with its duplicate

A high similarity of a bitstream usually occurs if the stream contains repeating bit sequences. Therefore, when entering the distance between two peaks as circulation length value, the similarity of the sequences will be easily visible in the *Bit Display*.

In the screenshot above, the first significant peak is located at shift 127, the second at shift 254, i.e. the distance between those peaks is 127. This is the number of bits after which the whole pattern is repeated.

Result Table for Autocorrelation and Partial Autocorrelation

This is the table for results of autocorrelation or partial autocorrelation:

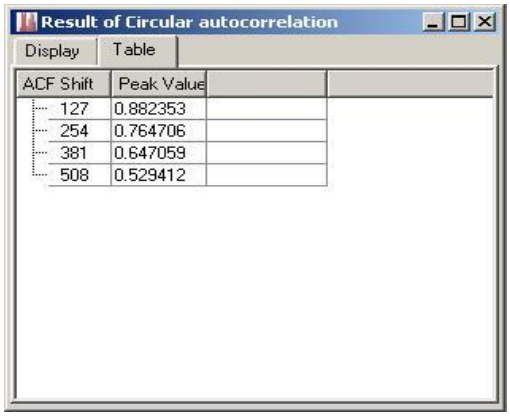


Figure 37: Result Table for Autocorrelation

The first column lists the *ACF Shift*, the second the result of the autocorrelation for this shift.

The table shows only results greater than half the maximum result.

If the number of results to display exceeds 10,000, only the first 10,000 results will be displayed.

When the result of an autocorrelation or partial autocorrelation is displayed in this result table, it is possible to double click the first column, and the value of the double-clicked shift will be set as new circulation length of the *Bit Display*.

Run Analysis

This function analyses the bitstream with respect to the number runs of ones and zeros with a definite length. On clicking the button <**Run Analysis**> in the Function Kit, the following dialogue box is displayed:

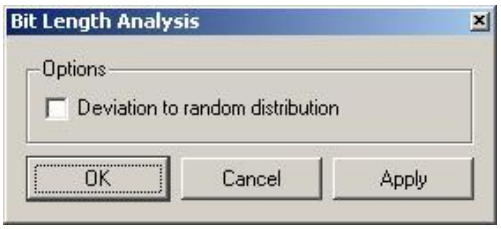


Figure 38: Dialogue Box for Run Analysis

If the option ☑ *Deviation to random distribution* is checked, the function will show the deviation between the results of the run analysis of the current bitstream and the result of a run analysis of a bitstream with a random sequence of bits. If the option is unchecked, the function will show the absolute result of the run analysis.

The definition of the terms run, block and breach is outlined below:

- A run is the designation for a sequence of the same bits, either zeros or ones.
- A block is the designation for a run of ones.
- A breach is the designation for a run of zeros.

Example:

We shall use the following bitstream example: 0001 1110 1001. This bitstream features (in order of occurrence):

- One breach with a length of 3
- One block with a length of 4
- One breach with a length of 1
- One block with a length of 1
- One breach with a length of 2
- One block with a length of 1

Consequently, there are two blocks with a length of 1, all other blocks occurring only once. Apply the run analysis to the exemplary bitstream *runanalysis.txt*. It consists of the example sequence repeated 27 times.

On clicking <**OK**> or <**Apply**>, go2ANALYSE opens a *Measurement Display* (unless already displayed) and shows the result for the current bitstream.

The displayed result is as follows:

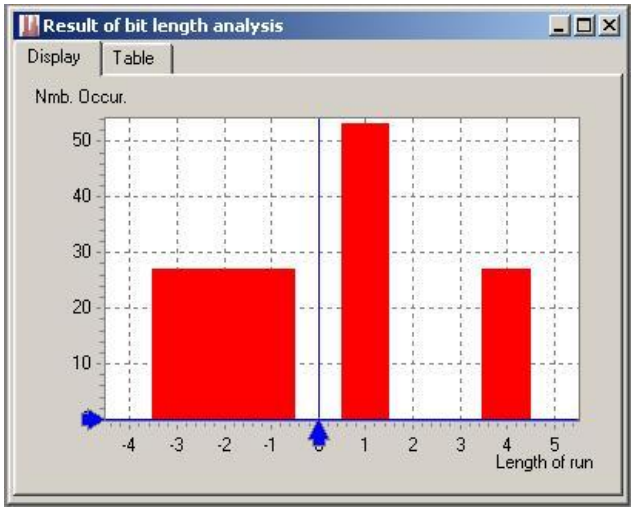


Figure 39: Exemplary Results of Run Analysis

The display shows a "bundle" of bars printed on the X-axis. Each bar represents one run, either block or breach. The height of the bar reflects the absolute number of occurrences of this run (since the option ☑ *Deviation to random distribution* was left unchecked), the width of all bars is always one. The location of the bar on the X-axis indicates the length of this run.

Negative lengths represent breaches, positive lengths represent runs of blocks. The absolute value must be taken to obtain the real length. The zero value is always blank.

The blocks occurred are plotted toward positive X, the occurred breaches are plotted towards negative X. The null is always blank.

When measuring the bars, the results are:

- Number of breaches with lengths of 3, 2 and 1: 27
- Number of blocks with a length of 1 (bar at 1): 54 (because this breach occurred twice in the primary example sequence)
- Number of blocks with lengths of 2 and 3: 0
- Finally, the block with a length of 4 occurred 27 times

Changes in the displayed result change when checking the option ☑ *Deviation to random distribution*:

As stated above, the absolute result of the run analysis is compared to the run analysis of the bitstream with a sequence of random bits.

In a bitstream with a random sequence of bits, when each bit has been generated by a random generator, the probability of a one or a zero is ½. Assuming a bitstream of random bits with a size of 500 bits, there should be 250 ones and 250 zeros in this stream. The probability of a block with a length of 2 can be calculated as:

½ * Size of Stream * Probability for a One * Probability for a One

thus ½ * 250 * ½ * ½ = 31.25.

Generally, the probability of occurrence for a block or a breach of a length N can be calculated as:

$$p(N) = \frac{0.5 * Bitstreamsize}{2^{N+1}}$$

If the option ☑ *Deviation to random distribution* is checked, go2ANALYSE will calculate the total number of occurrences for each run as it would occur in a bitstream with random bits.

The calculated results are then compared with the results of the run analysis of the current bitstream, and the deviation between these results is displayed.

When applying the option ☑ *Deviation to random distribution* to the example, the *Measurement Display* will show the following result:

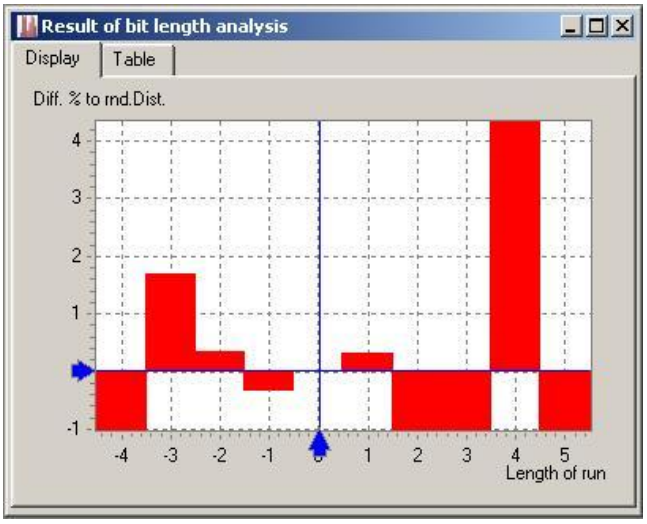


Figure 40: Exemplary Results with Option ☑ Deviation to Random Distribution Checked

Result Interpretation

The bars for blocks and breaches are printed at the same positions as with the option unchecked, but the value (height) of each bar indicates the difference (deviation) to the random bitstream.

A value of -1 indicates that this run has *never* occurred in this bitstream.

Bars with a value less than zero values indicate that the run in question has occurred less often in the current bitstream than it would in a random bitstream.

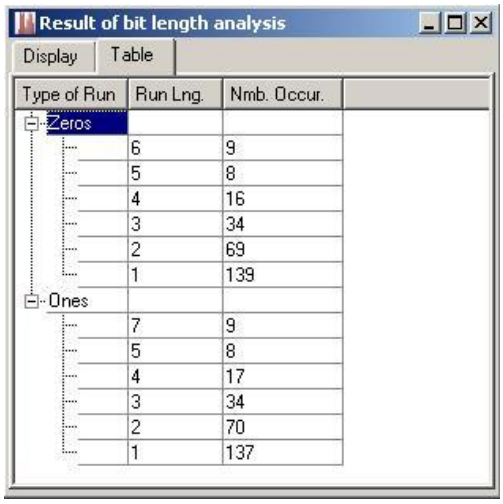
If a bar has a value of 0 (except the bar at zero which is always 0), this bar occurs in the current bitstream as many times as it would in a random one.

Bars with a value greater than zero indicate that the run in question has occurred more often in the current bitstream than it would in a random bitstream.

Note: If a run with a length greater than 32 bits shows in the bitstream, the value will always be 2!

Result Table for Run Analysis

The result table for a run analysis looks as follows:



Type of Run	Run Lng.	Nmb. Occur.
Zeros		
	6	9
	5	8
	4	16
	3	34
	2	69
	1	139
Ones		
	7	9
	5	8
	4	17
	3	34
	2	70
	1	137

Figure 41: Result Table for Run Analysis

The first column separates the blocks (ones) from the breaches (zeros). The second column contains the length of a run found while the total number of occurrences is shown in the third column.

When the option ☑ *Deviation to random distribution* is checked, the third column lists the deviation between the current result and the run analysis of a bitstream with random bits.

The interpretation of the results is the same as in the previous paragraph.

Measurement Display

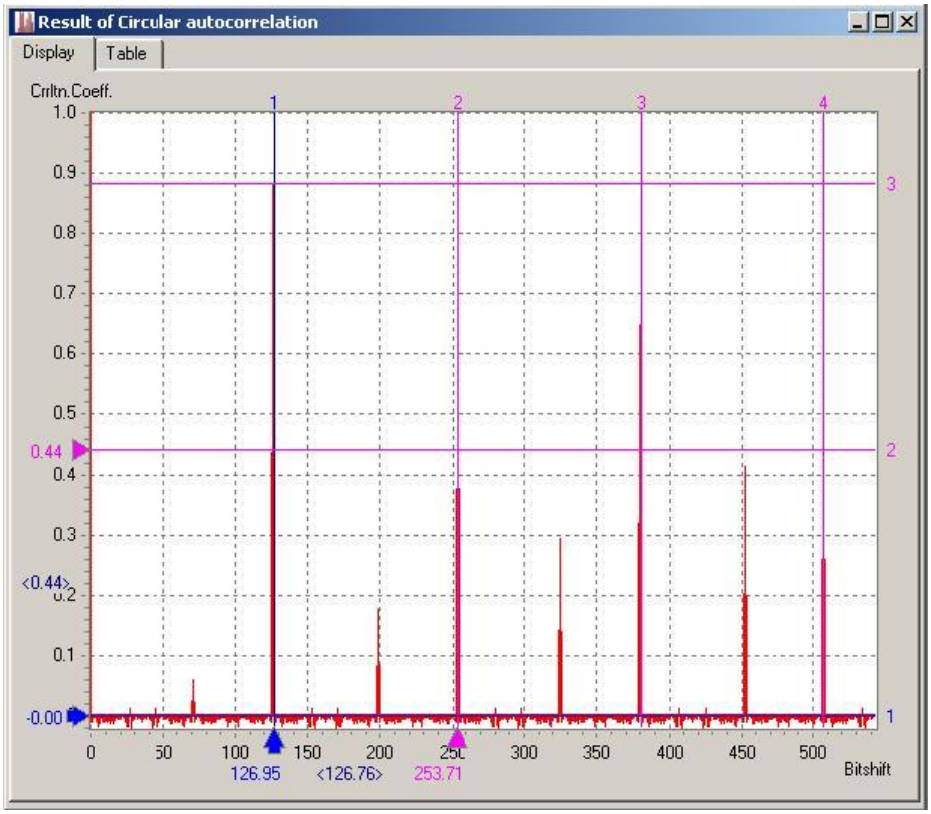


Figure 42: Measurement Display

Parameters

On the one hand, the displayed area of the measuring results changes when editing the listed parameters.

On the other, when applying a measurement function to the active display, the *Measurement Display* changes the current parameters to fit the current result.

Parameter	Valid Range	Does This
Minimum X	-1,000,000 – +999,999	Sets the start value of the X-axis
Maximum X	-999,999 – +1,000,000	Sets the end value of the X-axis. Is always greater than Minimum X.
Minimum Y	-1,000,000 – +999,999	Sets the start value of the Y-axis
Maximum Y	-999,999 – +1,000,000	Sets the end value of the Y-axis. Is always greater than Minimum Y.

Table 6: Measurement Display Parameters

Cursors

The cursor parameters are displayed on the <*Cursor*> tab. You may insert cursors into the display to measure the displayed results. The respective cursor positions are displayed and editable.

<table>
<tr><th>Parameter Group</th><th>Parameter</th><th>Does This</th></tr>
<tr><td>Cursor Options</td><td></td><td>Activate or deactivate cursor directions</td></tr>
<tr><td></td><td>X-Cursors On/Off</td><td>Toggle X-Cursor On/Off (activated / deactivated)</td></tr>
<tr><td></td><td>Y-Cursors On/Off</td><td>Toggle Y-Cursor On/Off(activated / deactivated)</td></tr>
<tr><td>Cursor Mode</td><td></td><td>Changes of cursor mode will apply to all active cursors</td></tr>
<tr><td></td><td>Normal</td><td>For each activated cursor direction two cursors are displayed (cursor 1: blue, cursor 2: magenta). The lowest parameter field labeled Distance 1-2 shows the distance between the cursors. Each cursor is moved separately.</td></tr>
<tr><td></td><td>Harmonic Cursor</td><td>For each activated cursor direction, up to 20 cursors are displayed at equidistant intervals.
The cursors will move by dragging the first cursor.
The width of the interval between the cursors is changed by dragging the second or any following cursor.
The lowest parameter field labeled Period shows the width of the cursor interval. Harmonic cursors serve for measuring periodic results.</td></tr>
<tr><td></td><td>X-Y Cursor</td><td>For each activated cursor direction only one cursor is displayed.
The lowest parameter field is labeled Value X / Value Y and shows the current position of the X- or the Y-cursor respectively.
The X- and Y-cursors are moved together (if both are activated) by dragging the intersection point of the cursors lines.</td></tr>
<tr><td>Cursor Positions and Measurement</td><td></td><td>Read out the current positions of the cursors and change them by editing the values of the parameters for Cursor 1 and Cursor 2.</td></tr>
<tr><td></td><td>Cursor 1</td><td>Value of the first (blue) X-cursor or Y-cursor.
When changing this value, the first cursor will change position. The cursors cannot be moved outside the visible area by editing this parameter.</td></tr>
<tr><td></td><td>Cursor 2</td><td>Value of the second (magenta) X-cursor or Y-cursor
When changing this value, the first cursor will change position. The cursors cannot be moved outside the visible area by editing this parameter.</td></tr>
<tr><td></td><td>Difference 1-2
Period
Value X / Value Y</td><td>This parameter changes the mode the bits are coded to code symbols.

Cursor Mode | Name | Parameter Description
Normal | Difference 1-2 | Difference between cursors 1 and 2
Harmonic | Period | Interval between cursors
X-Y | Value X/ Value Y | Position of X-cursor Position of Y-cursor</td></tr>
</table>

Table 7: Measurement Cursor Parameters

On double clicking the cursors, the current *Difference/Period/Value X* will be set as new circulation length of the *Bit Display*. This is especially useful to measure a periodical result e.g. the result of an autocorrelation. All cursors can be moved using the mouse. To move the cursor, place the mouse pointer on the cursor handle (arrow or triangle) or on the cursor line, press the left mouse button and move the mouse. The cursor is dragged with the mouse until you release the left mouse button.

- Remember: In X-Y Mode, both cursors will follow the mouse, so if you move to the side and upwards the X-cursor is moved to the side while the Y-cursor is moved upwards.

- On clicking inside the Measurement Display, the cursor closest to the current position of the mouse pointer will be moved to the position clicked.

Zoom Display

The buttons <**Zoom X-Axis**> and <**Zoom Y-Axis**> are only active when the cursor mode is not set to *X-Y Cursor*. On selection of <**Zoom X-Axis**>, the *Measurement Display* is zoomed in showing the area between the first two cursors.

On selection of <**Zoom Y-Axis**>, the *Measurement Display* is zoomed in showing the area between the first two cursors.

Extras

The <*Extras*> tab features the *Display* setting which can be used to change the current color scheme for the display:

Color Scheme	Cursor Color	Data Color	Background Color
Standard	blue, magenta	various colors	black
Inverse	blue, magenta	various colors	white
Monochrome	shades of grey	different shades of grey	white

Table 8: Measurement Display Color Schemes

Measuring Result Table

Each result displayed in the *Measurement Display* is also listed in the built-in result table. The layout of this result table depends on the type of result shown in the *Measurement Display*.

Frame Statistics

This function serves to analyze the bitstream by defining frames and calculating statistical data for each frame, displaying the results in a special display which facilitates the comparison of the frame statistics.

On clicking the button <**Frame Statistics**> in the Function Kit, the following dialogue box is shown:

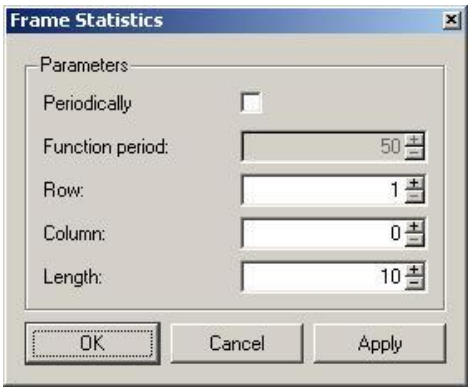


Figure 43: Dialogue Box for Frame Statistics

The parameters *Row, Column, Function Period* and the option ☑ *Periodically* define the frames to which the function will be applied.

The parameter *Function Period* is only applied if the option ☑ *Periodically* is checked.

Example:

The bitstream example used features a circulation length of 8.

Row \ Column	0	1	2	3	4	5	6	7
1	0	0	1	1	0	0	1	1
2	0	1	1	1	0	0	0	1
3	0	0	0	1	1	1	0	1
4	1	1	1	1	0	1	0	1
5	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	1	1

Figure 44: Bitstream Example

Apply the frame statistics function to the exemplary bitstream. To do so, enter the following parameter values:

- Row 1
- Column 2
- Length 8
- Function Period 8
- Check the option ☑ *Periodically*

Press <**OK**> or <**Apply**>.

go2ANALYSE opens the *Result Display*. Do not confuse this display with the result table of the *Measurement Display*. The *Result Display* is an independent display which is not part of any other display:

Frame Statistics Results

Frame Numb	Nmb. 1	Nmb. 0	Ratio 1/0	Ratio 0/1	Start Row	Start Colum	End Row	End Column
0	4	4	1.000	1.000	1	0	1	7
1	4	4	1.000	1.000	2	0	2	7
2	4	4	1.000	1.000	3	0	3	7
3	6	2	3.000	0.333	4	0	4	7
4	4	4	1.000	1.000	5	0	5	7
5	2	6	0.333	3.000	6	0	6	7

Figure 45: Exemplary Results of the Frame Statistics

Each frame analyzed is listed in the order of its occurrence in the bitstream. Consequently, the frame with number 0 must be the first frame analyzed, the frame with the highest number must be the last. In our example, frame number 0 covers the bits in row 1, frame number 5 the bits in row 6.

For each frame, the result table shows:

- The number of Ones in this frame
- The number of Zeros in this frame
- The ratio of Ones to Zeros in this frame
- The ratio of Zeros to Ones in this frame
- And the “coordinates” (start and end row, start and end column) of the frame indicating where each frame is located in the bitstream.

Note: When the current circulation length of the *Bit Display* is changed while the *Result Display* is open, the “coordinates” for each frame will be adapted to the new circulation length.

Parity and Weight Statistics

This function serves to analyze the bitstream by defining frames, to calculate the parity bit for even and odd parities for each frame, and to show the results in a special display which facilitates the comparison with the statistics of other frames.

On clicking the button <**Parity & Weight**> in the Function Kit, the following dialogue box is shown:

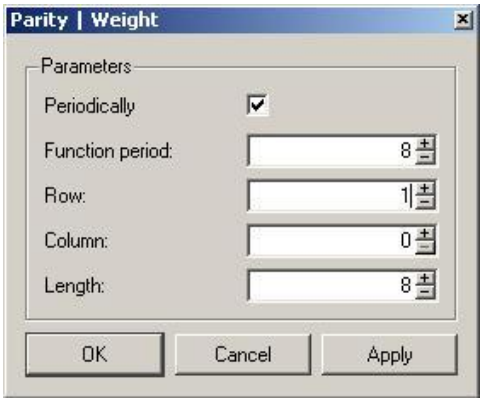


Figure 46: Dialogue Box for Parity and Weight

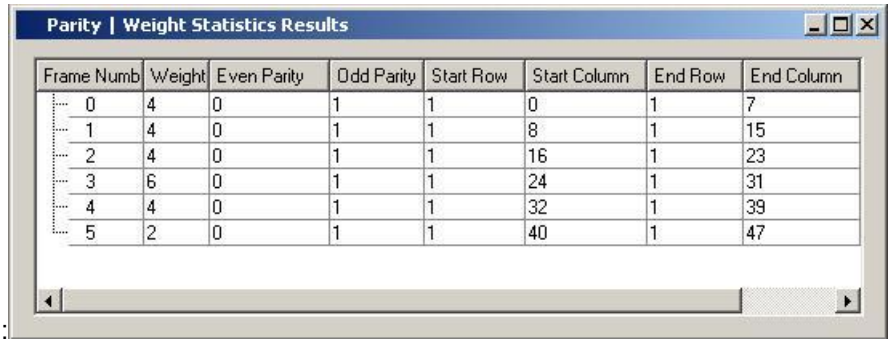
The parameters *Row, Column, Function period* and ☑ *Periodically* define the frames the function is applied to.

The parameter *Function period* is only applied if the option ☑ *Periodically* is checked.

Example:

This function is applied to the bitstream used in the frame statistics example.

The result display shows the following results:



Frame Numb	Weight	Even Parity	Odd Parity	Start Row	Start Column	End Row	End Column
0	4	0	1	1	0	1	7
1	4	0	1	1	8	1	15
2	4	0	1	1	16	1	23
3	6	0	1	1	24	1	31
4	4	0	1	1	32	1	39
5	2	0	1	1	40	1	47

Figure 47: Exemplary Results of the Parity Weight Analysis Function

Each frame analyzed is listed in the order of its occurrence in the bitstream. Consequently, the frame with number 0 must be the first frame analyzed, the frame with the highest number must be the last. In the example, the frame number 0 covers the bits in row 1, frame number 5 the bits in row 6.

For each frame, the result table shows:

- The “weight” (weight meaning the sum of ones) of this frame
- The even parity bit, i.e. the bit which must be added to the sum of ones so the sum of ones is an even number.
- The odd parity bit, i.e. the bit which must be added to the sum of ones so the sum of ones is an odd number.
- The “coordinates” (start and end rows, start and end columns) of the frame indicating the position of each frame in the bitstream.

Note: When changing the current circulation length of the *Bit Display* while the *Result Display* is open, the “coordinates” for each frame will be adapted to the new circulation length.

Result Display

The third display in go2ANALYSE is the *Result Display*. The main element of this display is a table showing the various function results. Operation and handling of the result display are explained in detail in the respective chapters describing the go2ANALYSE functions whose results are displayed in this display.

These functions are:

- Frame Statistics function
- Parity and Weight Statistics function
- Search for bit sequences from LFSRs

Search Features

Pattern Search

This function searches all occurrences of the entered bit sequence in the bitstream and tags them with the chosen color.

On clicking the button <**Pattern**> in the Function Kit, the following dialogue box is shown:

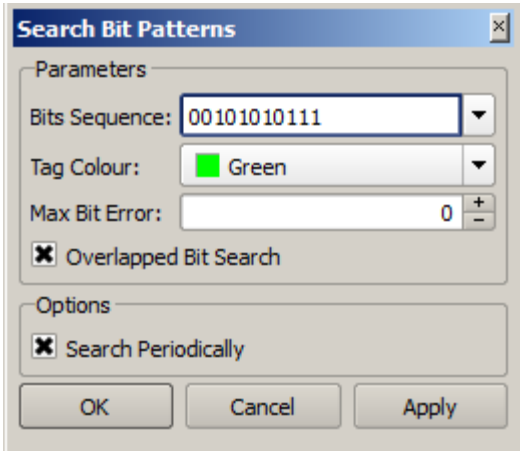


Figure 48: Dialogue Box for Pattern Search

Enter the bit sequence to search for in the list box *Sequence of bits* (or choose an existing bit sequence on the property sheet and select an existing entry*)*. Choose the color to tag the bits with on the property sheet *Tag colour*.

On pressing <**OK**> or <**Apply**>, this function will find all occurrences of the entered sequence in the current bitstream. Leave the option ☑ *Search periodically* unchecked for a normal search.

Note: The characters entered in the list box *Sequence of bits* will be converted to bits using the following conversion table:

Character Input	Equivalent Bit
0	0
- (dash)	0
_ (underline)	0
L	0
. (Dot)	0
1	1
X	1
x	1

Character Input	Equivalent Bit
H	1

Table 9: Bit Sequence Conversion

Example:

Open the file with the *LFSR-Sequence1*, copy the first 10 bits in row 2 to the clipboard and paste them in the field *Sequence of Bits*, choose the tag color *Green* and press <**OK**>.

Subsequently, the *Bit Display* shows the following image (circulation length is 50):

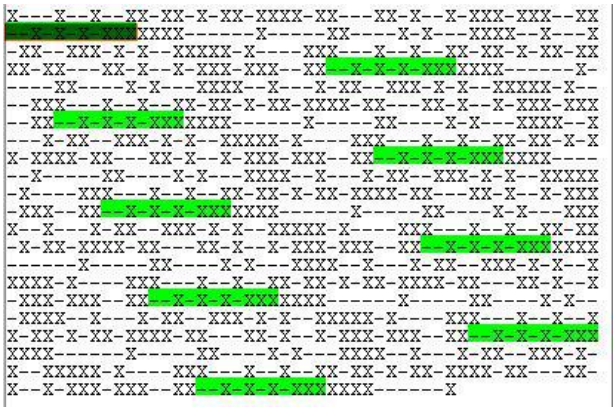


Figure 49: LFSR-Sequence Bitstream after Search

Every bit in the bitstream can only have one tag color. Therefore, after searching the same pattern one more time with a different color, the bits will not be tagged with two colors or a mix of them but with the new color chosen.

Search Periodically

On checking the option ☑ *Search periodically* and pressing <**OK**> or <**Apply**> after entering the search string and choosing the tag color, the software will execute two steps.

First, go2ANALYSE carries out a normal search of the entered bit sequence as described above.

Second, go2ANALYSE does a partial autocorrelation of the bitstream with the entered bit sequence and opens the *Measurement Display* (see chapter Autocorrelation and Partial Autocorrelation for details).

Using periodical search with our example, the *Measurement Display* will show the following image:

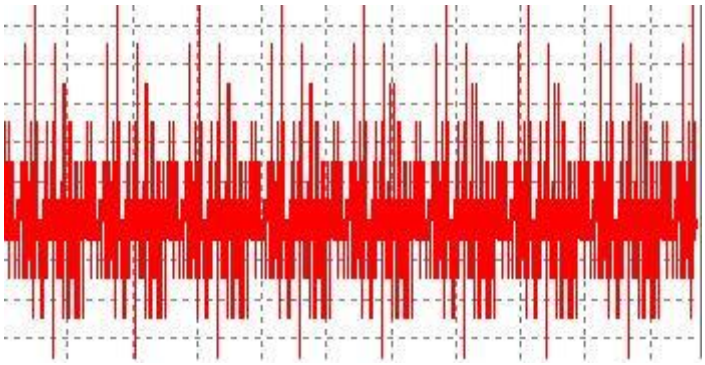


Figure 50: Exemplary Results of a Periodic Search

Purpose

When measuring the period between the highest peaks with the cursors in the *Measurement Display* and entering this period as circulation length of the bitstream, the result is as follows:

Figure 51: Bitstream with a Circulation Length Equal to the Measured Period

Consequently, the option ☑ *Search periodically* facilitates the search for periodical repetitions of sequences in the bitstream.

Search for LFSR Sequences

This function serves to search for bit sequences generated by linear feedback shift registers (acronym: LFSR).

Introduction to LFSR Sequences and Berlekamp-Massey Algorithm

A *linear feedback shift register* is a shift register whose input is the Exclusive-OR (XOR) of some of its outputs. The outputs that influence the input are called *taps*. An LFSR with N taps (N is normally referred to as the linear complexity or length of the LFSR) is able to generate a pseudorandom binary sequence with a maximum length of 2^N-1 bits if it is constantly "fed" with zeros. As a precondition, the cells of the shift register have to be set to any non-zero state and then cycled. After the output of this bit sequence of 2^N-1 bits, the same sequence is repeated if the LFSR is continued to be fed with zeros. If the precondition is not met, the output bits of the LFSR are all zero if the input bits are zero.

The tap sequence of an LFSR can be represented as a polynomial modulo 2 - called the feedback polynomial. For example, if the taps are at the 1st and 3rd bits, the polynomial is *1+ x1 + x3* (see figure below).

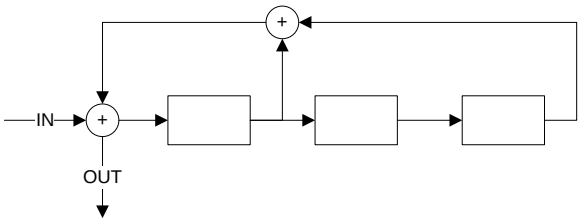


Figure 52: LFSR with Polynomial 1+x^1+x^3

The length of the pseudorandom binary sequence is maximal if the polynomial of the LFSR is a primitive one as it is called, i.e. it cannot be reduced to terms.

Given an output sequence of an LFSR you can reconstruct the generating LFSR of minimal size using the Berlekamp-Massey algorithm (acronym: BMA). This algorithm can be used to find the shortest linear feedback shift register (LFSR) for a given output sequence.

To do so, the algorithm needs to test at least 2 * N bits to reconstruct a generating LFSR with a linear complexity N, however, it is possible that the algorithm reconstruct only an LFSR with M < N because this LFSR was the shortest LFSR which could generate the tested sequence.

The algorithm always returns the linear complexity and the polynomial of the LFSR which generated the tested bit sequence.

It is fundamental to know that a primitive LFSR with a length X > N can generate the bit sequence of a primitive LFSR with a length N but the LFSR with a length N cannot generate the bit sequence of the LFSR with a length X.

Using LFSR Search

On clicking the button <**LFSR**> in the Function Kit, the following dialogue box is displayed:

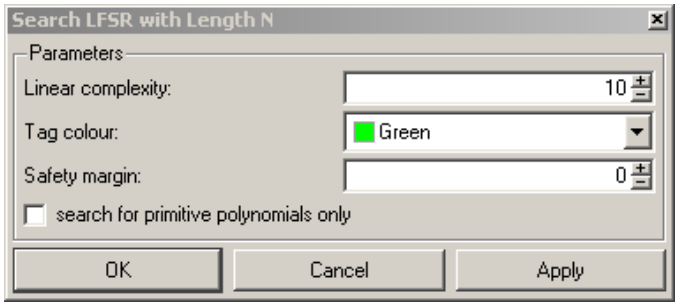


Figure 53: Dialogue Box for LFSR Search

Adjust the maximum linear complexity of the LFSR to search for by entering the appropriate values in the parameter box *Linear Complexity*. The parameter may range from 1 to 256.

All bit sequences found by the function are tagged with the current color chosen by means of the parameter *Tag Colour*.

The parameter *Safety margin BMA* allows for improving the reliability of the BMA's “statement” about the LFSR which generated the tested bit sequence.

Normally the BMA needs to test a number of bits equal to two times the entered linear complexity (N) to reconstruct the LFSR. If the parameter *Safety Margin BMA* (we use *K* to refer to this parameter) is set to any value greater than zero, then 2*N + K bits will be tested by the BMA.

If the linear complexity returned after the test of 2*N +K bits is still N, then the reliability that this bit sequence was generated by the LFSR with length N and not with an LFSR with a length greater than N has improved.

Note: A value of 5 for *K* is a good compromise between improving the reliability and the required length of a bit sequence to test for LFSRs.

Example:

To apply the LFSR to the exemplary bitstream *LFSR-Sequence1,* proceed as follows:

Load the bitstream, press the <**LFSR**> button and enter the following parameters:

- Linear complexity: 7
- Tag colour: Red
- Safety Margin BMA: 0
- Leave the option ☑ Search for primitive polynomials only unchecked

Press <**OK**> or <**Apply**>. The *Bit Display* then shows the following bitstream:

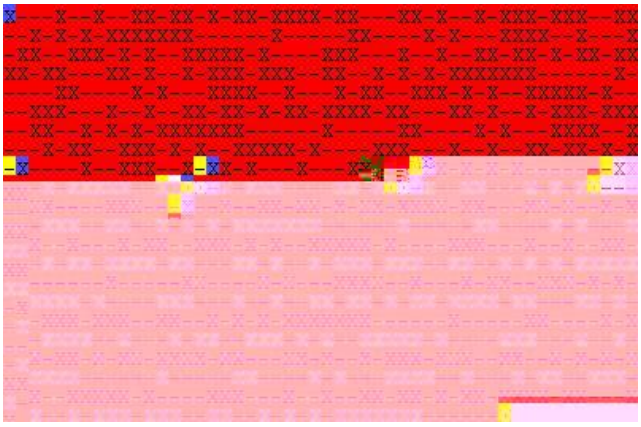


Figure 54: Part of the Exemplary Bitstream after LFSR Search

Bits tagged in blue indicate the start of a bit sequence from an LFSR matching the search criteria, bits tagged in yellow indicate the end of this bit sequence. All bits tagged in red are part of an LFSR sequence.

Note: If you choose a tag color other than red, the start and end bits will not be tagged in blue and yellow but always tagged in contrasting colors which facilitate the search for the start and end of the bit sequence.

go2ANALYSE will also open a new *Result Display* (unless already open) and display the results of the example:

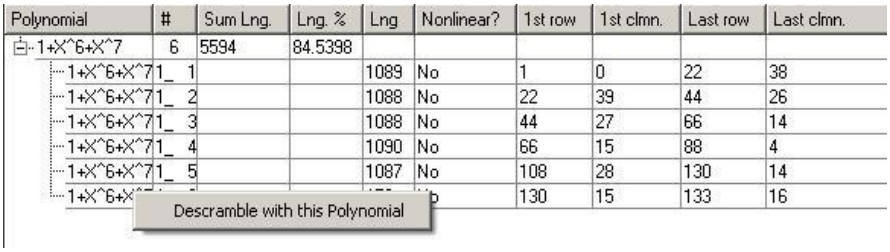


Figure 55: Exemplary Results of LFSR Search

Item	Explanation
Polynomial	The column Polynomial contains the polynomial of the LFSRs found. All polynomials with the same equation are grouped in one “family” and listed below a “parent” polynomial as it is called (see first row in the above screenshot). Below each “parent” polynomial is a list of the “child” polynomials, these are all polynomials of the same order and with the same equation (the second and following rows in the screenshot show a list of the child polynomials listed in the first row). Once a new result of an LFSR search is displayed, all child polynomials are hidden. To see the list of child polynomials for a parent polynomial click the plus sign next to the equation of polynomial. If the list of child polynomials is open, the plus sign has changed to a minus (see first column in the screenshot above).
#	The column Number indexes all polynomials the indices of any child polynomials consist of the parent number followed by a consecutive index separated by an underscore.
Sum Lng	The Sum Lng. column lists how many bits of this bitstream have been generated by LFSRs with this equation.
Lng %	The column value in Lng.% is calculated as: 100 * Sum Lng divided by the summed length of all LFSRs found.
Lng	The value in the column Lng is the length of the bit sequence found in bits.
Nonlinear?	No in the column Nonlinear indicates that the BMA did not find any unusual jumps in the linear complexity as it reconstructed the LFSR. Yes indicates that this LFSR is probably nonlinear.
1st Row 1st Col' Last Row Last Col	All child polynomials are sorted by their start row (Column: 1st Row). The “coordinates” of the start bit of each bit sequence the child polynomial belongs to is given by the values of start row and start column, the “coordinates” of the last bit of this sequence is given by the value of last row and last column.

Table 10: LFSR Parameters

Note: If the current circulation length of the *Bit Display* is changed while the *Result Display* remains open, the “coordinates” for each frame are adapted to the new circulation length.

When double clicking any polynomial, a popup menu with the item *Descramble with this Polynomial* will be displayed. Clicking this item opens the dialogue box for descrambling, and the *LFSR-Polynomial* parameter is set to the double-clicked polynomial. For details on descrambling the bitstream see chapter Descramble Bitstream.

Polynomial Check

This function is helpful when analyzing the true length of an LFSR bit sequence.

On clicking the button <**Polynomial**> in the Function Kit, the following dialogue box is shown:

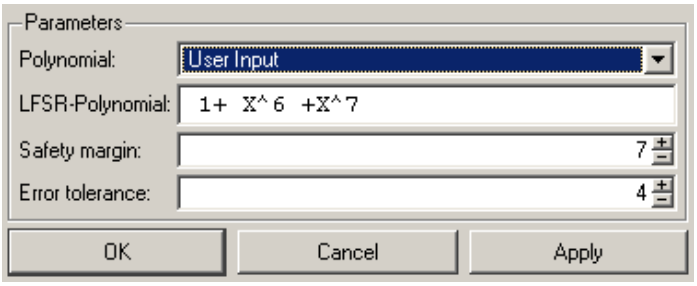


Figure 56: Dialogue Box for LFSR-Polynomial Search

To check which bit sequences in the current bitstream have been generated by a specific LFSR and to see how long these are, either choose a predefined polynomial from the list in the dropdown list box of the parameter *Polynomial* or enter your own polynomial.

Note: It is always possible to edit a predefined polynomial by editing the box *LFSR-Polynomial*.

Name of Polynomial	Equation
V.27	1 + X^6 + X^7
V.22	1 + X^14 + X^17
511	1 + X^5 + X^9
V.12, V.29, V.32, V.33	1 + X^18 + X^23

Table 11: Predefined LFSR Polynomials

All V-polynomials have the equation according to ITU Modem Recommendations (formerly CCITT) V-Specifications.

On selection of a predefined polynomial or selection of *User Input*, the parameter *LFSR-Polynomial* allows for entering a new polynomial or editing the current entry.

Polynomials can be entered by the following four notations:

- 1+x^-2+x^-3+x^-7
- 1+x^2+x^3+x^7
- 1+x2+x3+x7
- 1+2+3+7

On clicking <**OK**> or <**Apply**> go2ANALYSE opens a new *Measurement Display* (unless already open) and visualizes the result.

Manipulation Features

Delete

Deletes the bits defined by the current function parameters from the bitstream.

On clicking the button <**Delete**> in the Function Kit, the following dialogue box is shown:

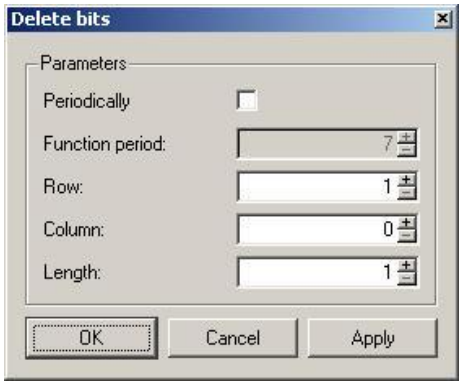


Figure 57: Dialogue Box for Delete

The parameters *Row, Column, Function Period* and ☑ *Periodically* define which bits the function is applied to (see chapter Define Bits for Application of go2ANALYSE Functions for details).

On clicking <**OK**> or <**Apply**> the function is applied to the stream. It may take a while until all bits defined have been deleted from the bitstream.

Tag Colours

Tags the bits defined by the current function parameters with the chosen color.

On clicking the button <**Tag Bits**> in the Function Kit, the following dialogue box is shown:

Figure 58: Dialogue Box for Tagging Bits

The parameters *Row, Column, Function Period* and ☑ *Periodically* define which bits the function is applied to (see chapter Define Bits for Application of go2ANALYSE Functions for details).

The parameter *Tag Colour* defines the color in which the bits are tagged.

On clicking <**OK**> or <**Apply**> the function is applied to the stream.

Note: Each bit in the stream can only have one tag color at a time. Consequently, after searching the same pattern one more time with a different color the bits will not be tagged with two colors or a mix of them but with the new color chosen.

Mirror

This function mirrors the bits in the frame which is defined by the parameters *Row, Column, Length, Function Period* and ☑ *Periodically*.

On clicking the button <**Mirror**> in the Function Kit, the following dialogue box is shown:

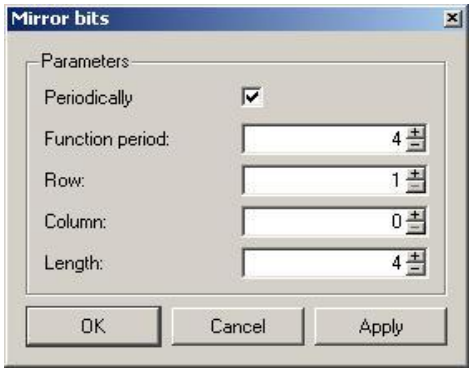


Figure 59: Dialogue Box for Mirroring

The parameters *Row, Column, Function Period* and ☑ *Periodically* define which bits the function is applied to (see chapter Define Bits for Application of go2ANALYSE Functions for details).

Example:

The example uses the following bitstream with a circulation length of 4:

1	1	1	0
1	0	0	0

Figure 60: Bitstream Before Mirroring

Apply the mirror function with the parameters shown in the Figure *Dialogue Box for Mirroring* (i.e. ☑ *Periodically* checked, *Function period* 4, *Row* 1, *Column* 0, *Length* 4).

Subsequently, the following bitstream is obtained:

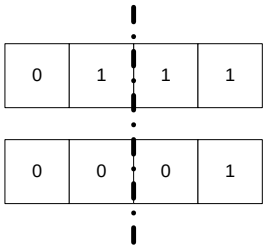


Figure 61: Bitstream After Mirroring

Each row has been mirrored across the dash-dotted line.

Note: If the parameter *Length* is even, all bits in the frame will be mirrored, otherwise if *Length* is odd, the bit in the center of the frame will remain unaffected by the mirroring function.

Clear Tags

Clears either all tags of the chosen color, or all tags of any color.

On clicking the button <**Clear Tags**> in the Function Kit, the following dialogue box is shown:

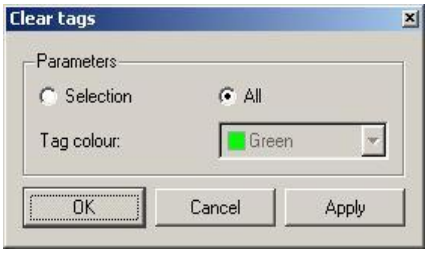


Figure 62: Dialogue Box for Clearing Tags

If the radio button ⊙ *All* is selected, then all tags regardless of their color are cleared from the bitstream. If the radio button ⊙ *Selection* is enabled, only the tags with the color defined by the parameter *Tag Colour* will be cleared.

Insert and Paste

The bits to be inserted into the current bitstream must be previously copied to the clipboard.

To do so, either highlight the bits in the stream using the mouse, click the right mouse button and select *Copy* from the popup menu or copy a number of characters from a text editor to the clipboard to insert them into the bitstream. However, please note that the only characters admissible for this purpose are the ones listed in the table in chapter Open Bitstream. Copying other characters to the clipboard will have the effect that no bits are inserted.

To insert the copied bits into the stream, select (highlight) the position at which to insert the bits, click the right mouse button and choose *Insert*. The bits are inserted into the bitstream, the highlighted bits are not replaced by the ones inserted.

Note: The *Bit Display,* as any other text editor, has no cursor. To select the position at which to insert the bits highlight least one bit with the mouse.

When choosing *Paste* instead of *Insert*, the bits highlighted in the bitstream are replaced with the bits from the clipboard.

Logic Features

AND

This function is located in the Function Kit and does a bitwise AND operation of the bits defined by the parameters *Row, Column,* the *Length of the entered bit sequence, Function period* and ☑ *Periodically,* using the entered bit sequence.

The bits are combined by means of the following logic table:

Bit of Sequence	Bit of Stream	Result of Operation
0	0	0
0	1	0
1	0	0
1	1	1

Table 12: Logic Table Bitwise AND Function

Example:

The example uses a bitstream with two rows and a circulation length of 10 bits.

Row \ Column	0	1	2	3	4	5	6	7	8	9
1	-	-	X	X	-	-	X	X	-	-
2	X	-	X	-	X	-	X	-	X	-
3	-	X	-	X	-	X	-	X	-	-

Figure 63: Bitstream Before AND Operation, Area of Interest is Highlighted

It is intended to apply the sequence of bits –X-XX to the area of interest highlighted in yellow. To do so, enter the following parameters in the dialogue box of the AND function:

- ☑ *Periodically* Checked
- Function period: 10
- *Row:* 1
- Column: 1
- Sequence of bits: –X-XX

Now the AND operation is done periodically every 10 bits, starting in row 1, column 1. The sequence of bits is 5 bits long, so the bits in columns 1, 2, 3, 4 and 5 are affected by the function. On clicking the <**OK**> button of the dialogue box, the bits in all rows have been changed to:

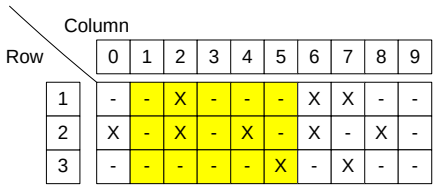


Figure 64: Bitstream After Applied AND Operation

OR

This function is located in the Function Kit and does a bitwise OR operation of the bits defined by the parameters *Row, Column,* the *Length of the entered bit sequence, Function period* and ☑ *Periodically,* using the entered bit sequence.

The bits are combined by means of the following logic table:

Bit of Sequence	Bit of Stream	Result of Operation
0	0	0
0	1	1
1	0	1
1	1	1

Table 13: Logic Table Bitwise OR Function

Example:

The example uses a bitstream with two rows and a circulation length of 10 bits.

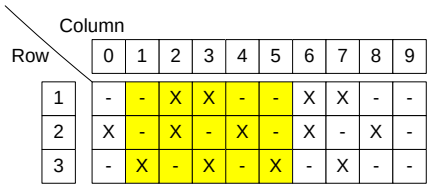


Figure 65: Bitstream Before OR Operation

It is intended to apply the sequence of bits –X-XX to the area of interest highlighted in yellow. To do so, enter the following parameters in the dialogue box of the OR function:

- ☑ *Periodically* Checked
- Function period: 10
- *Row:* 1
- Column: 1
- Sequence of bits: –X-XX

Now the OR operation is done periodically every 10 bits, starting in row 1, column 1. The sequence of bits is 5 bits long, so the bits in columns 1, 2, 3, 4 and 5 are affected by the function. On clicking the <**OK**> button of the dialogue box, the bits in all rows have been changed to:

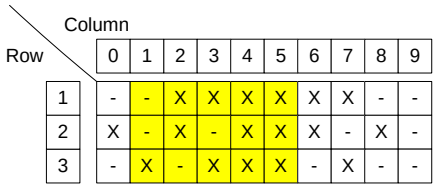


Figure 66: Bitstream After Applied OR Operation

NOT

This function is located in the Function Kit and does a bitwise negation of the bits defined by the parameters *Row, Column, Length, Function Period* and ☑ *Periodically*.

The bits are negated by means of the following logic table:

Bit of Stream	Result of Operation
0	1
1	0

Table 14: Logic Table Bitwise NOT Function

Example:

The example uses a bitstream with two rows and a circulation length of 10 bits.

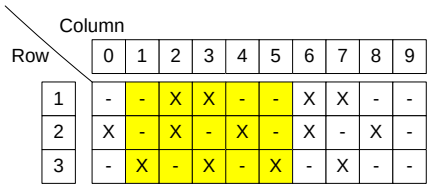


Figure 67: Bitstream Before NOT Operation

It is intended to apply the sequence of bits –X-XX to the area of interest highlighted in yellow. To do so, enter the following parameters in the dialogue box of the NOT function:

- ☑ *Periodically* Checked
- Function period: 10
- *Row:* 1
- Column: 1
- Length: 5

Now the NOT operation is done periodically every 10 bits, starting in row 1, column 1. The sequence of bits is 5 bits long, so the bits in columns 1, 2, 3, 4 and 5 are affected by the function. On clicking the <**OK**> button of the dialogue box, the bits in all rows have been changed to:

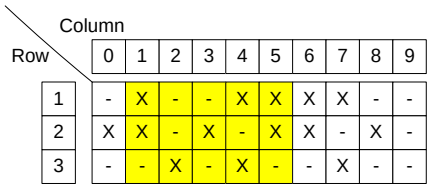


Figure 68: Bitstream After Applied NOT Operation

XOR

This function is located in the Function Kit and does a bitwise Exclusive OR (XOR) operation of the bits defined by the parameters *Row, Column,* the *Length of the entered bit sequence, Function period* and ☑ *Periodically,* using the entered bit sequence.

The bits are combined by means of the following logic table:

Bit of Sequence	Bit of Stream	Result of Operation
0	0	0
0	1	1
1	0	1

Bit of Sequence	Bit of Stream	Result of Operation
1	1	0

Table 15: Logic Table Bitwise XOR Function

Example:

The example uses a bitstream with two rows and a circulation length of 10 bits.

Row \ Column	0	1	2	3	4	5	6	7	8	9
1	-	-	X	X	-	-	X	X	-	-
2	X	-	X	-	X	-	X	-	X	-
3	-	X	-	X	-	X	-	X	-	-

Figure 69: Bitstream Before XOR Operation

It is intended to apply the sequence of bits –X-XX to the area of interest highlighted in yellow. To do so, enter the following parameters in the dialogue box of the XOR function:

- ☑ *Periodically* Checked
- Function period: 10
- *Row:* 1
- Column: 1
- Sequence of bits: –X-XX

Now the XOR operation is done periodically every 10 bits, starting in row 1, column 1. The sequence of bits is 5 bits long, so the bits in columns 1, 2, 3, 4 and 5 are affected by the function. On clicking the <**OK**> button of the dialogue box, the bits in all rows have been changed to:

Column \	0	1	2	3	4	5	6	7	8	9
1	-	-	-	X	X	X	X	X	-	-
2	X	-	-	-	-	X	X	-	X	-
3	-	X	X	X	X	-	-	X	-	-

Figure 70: Bitstream After Applied XOR Operation

User Functions

The *Decoder Description Language (DDL)* is a programming language for the implementation of software decoders. These software decoders convert the data from a bitstream into the output information.

DDL is a simple programming language developed especially for decoding tasks. It includes the individual commands which are processed in the application of a decoder in the appropriate sequence. A compiler translates this text into a code which is interpreted quickly and easily during the runtime of the decoder. Source code and compiled decoders are stored in files.

If you are interested in developing your own decoders then please contact us so that we can provide special tools and trainings. See the Service chapter for contact information.

go2ANALYSE provides the option to apply compiled software decoders to a bitstream. The decoder can supply different output types such as bitstream output, graphic output, marker output, progress bar output and text output, which are displayed with go2ANALYSE. The decoder also has the capability to receive user-defined parameters.

Use Decoders

The decoders can be used on record-based bitstreams (see chapter Open Bitstream). Text-based bitstreams can be decoded with the software decoders as well. To apply a decoder to the current bitstream, select the decoder on the *User Functions* menu. A decoder dialogue box is displayed with or without defined parameters.

Figure 71: Exemplary Histogram Decoder Dialogue Box without Parameters

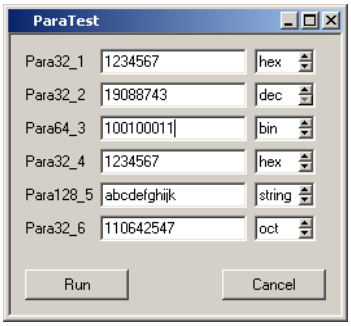


Figure 72: Exemplary ParaTest Decoder Dialogue Box with Parameters

Note:

The *User Functions* menu cannot be used unless at least one decoder is available in the decoder directory of your g go2ANALYSE installation. Having chosen the correct decoder, press <**Run**> to start decoding

or analyzing. The <**Cancel**> button closes the dialogue box. On successful decoding of the file, the decoder output is shown in go2ANALYSE. If the specific output is not displayed before applying the decoder, go2ANALYSE will open the output, otherwise the content will be overwritten by the new decoder output.

Configure User Functions

The *User Functions* are configured using an XML configuration file.

```
<go2ANALYSE version="1.5.0">
                          <guielement class="textbox">
                              <format>hex</format>
                              <name>Parameter</name>
                              <value>01234567</value>
                              <size>1</size>
                          </guielement>

                          ...

                          <guielement class="textbox">
                              <format>string</format>
                              <name>Search String</name>
                              <value>abcdefghijk</value>
                              <size>4</size>
                          </guielement>
</ go2ANALYSE>
```

The first line must be a root element of configuration:
`< go2ANALYSE>` or, with optional attribute `< go2ANALYSE version="1.2.1">`

The next line defines the type of a GUI element:
`<guielement class="textbox">`
Following types can be defined by assigning to class:

class	Description
textbox	General Input field for different formats
markstart	Like textbox, but the start position of a marker in an active bit display is automatically set here
marklength	Like textbox, but the length of a marker in an active bit display is automatically set here
checkbox	A check box, which can be just set on off
comment	A text in the head of the dialogue box, typically to explain the function

Table 16: GUI Elements

The next child element defines the default format of the text box.
`<format>hex</format>`
The following formats for text boxes are supported:

Format Definition	Description	Valid Input Characters
dec	Decimal number	0 – 9
hex	Hexadecimal number	0 – 9 | a – f | A – F
oct	Octal number	0 – 7
bin	Binary number	0 | 1
string	ASCII text	ASCII code

Table 17: GUI Text Box Formats

This element defines the name of the parameter.

```
<name>Para32_1</name>
```

The next element defines the default value of the parameter:

```
<value>01234567</value>
```

The next element defines the length of the parameter in bits (size x 32 bits = parameter length):

```
<size>4</size>
```

The next element closes the GUI element definition:

```
</guielement>
```

The next line can either be a GUI element definition or the last closing root element.

This element is the last closing root element

```
</ go2ANALYSE>
```

The result of this exemplary definition is a dialogue box like this:

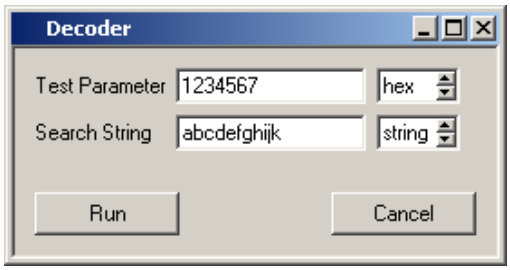


Figure 73: Exemplary Dialogue Box

Decoders without any parameters have the following XML definition:

```
< go2ANALYSE version="1.5.0">
</ go2ANALYSE>
```

User Functions Language Description

User Functions can control different output types in go2ANALYSE, and this output feature can also be used with DDL. The DDL function *OutVal* serves to supply a specific output type.

Text Output

The standard text output for the *User Functions* is the *Text Display*. The status bar of the *Text Display* shows the name of the decoder the current bitstream was decoded with, and *Mode: Text | ASCII*.

The parameters *Serialisation, Bitstream, Code Table, Start Level* and *Force Level* and the <**Sync Mode**> button are disabled.

The text shown in the *Text Display* is saved by means of the <**Save Text As…**> button. See chapter Text Display for details.

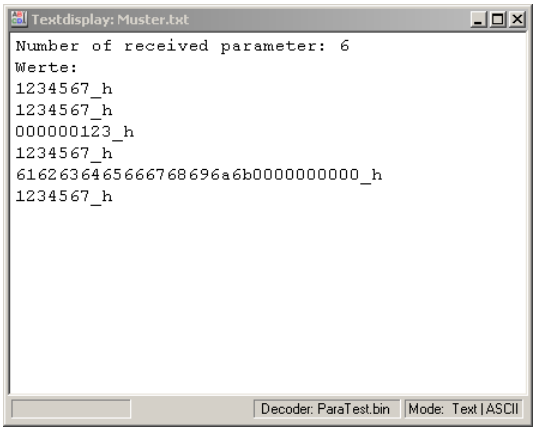


Figure 74: Text Display

OutText

OutText (Text, Addr)

Description:

Direct output of data package which consists of ASCII symbols (character set defined by target system / output system)

Parameter	Direction	Description	Valid Range
Text	Input	Variable containing 0-terminated ASCII string	Any value
Addr	Input	Output address	All output commands refer to an output address, which is defined by the parameter *Addr*. *Addr = 0* No output will be transferred, however, all remaining testing, validations, etc. are done. *Addr > 0* The output will be directed to the standard output (display, data base, etc.). The value of *Addr* is application dependent, e.g. a channel, type of information, or similar. The following values are admissible: Values 1 to 14: The output will be framed within XML tags <tex1> to <text14> Strings up to 16 characters: The XML tag contains this string. Consequently, output addresses e.g. *1* and *text1* have the same result. *Addr = -1* The output is directed to a variable serving as an output buffer. The variable must be defined using SetOutBuf. The output uses string format, i.e. outputs of succeeding calls are appended. On reaching the maximum string length (given by variable length), any further output to this address is stopped until the variable is reset (assigned) to 0.

Table 18: OutText Parameters

Graphic Output

The decoder can plot graphics in one or two dimensions. For 2D, *plot2dx* and *plot2dy* must be used in pairs otherwise *ploty* will be used for 1D.

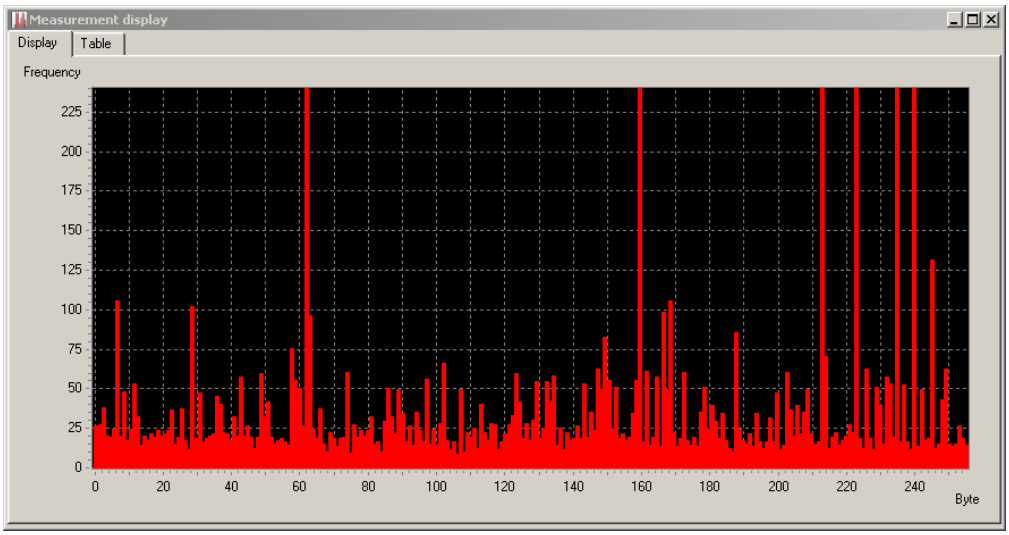


Figure 75: Exemplary Dialogue Box

graphcolor

OutVal *(Format*, *DisplayStyle*, "graphcolor")

Description:

Set display style of graphic output. It can be standard color, inverse color or monochrome color.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
DisplayStyle	DisplayStyle	0 standard 1 inverse 2 monochrome

Table 19: Graph Color Parameters

graphxmin

OutVal *(Format*, *Xmin*, "graphxmin")

Description:

Set minimum of X-axis.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
Xmin Xmax Ymin Ymax	Minimum of X-axis Maximum of X-axis Minimum of Y-axis Maximum of Y-axis	-2.000.000 to 2.000.000

Table 20: Graph Axis Parameters

graphxmax

OutVal *(Format*, *Xmax*, “graphxmax”)
Description:
Set maximum of X-axis.

graphymin

OutVal *(Format*, *Ymin*, “graphymin”)
Description:
Set minimum of Y-axis.

graphymax

OutVal *(Format*, *Ymax*, “graphymax”)
Description:
Set maximum of Y-axis.

graphxunit

OutVal *(Format*, *AxisUnit*, “graphxunit”)
Description:
Set unit of X-axis.

Parameter	Description	Admissible Values
Format	Output format	s string
AxisUnit	Unit of X/Y-axis	Any character

Table 21: Graph Unit Parameters

graphyunit

OutVal *(Format*, *AxisUnit*, “graphyunit”)
Description:
Set unit of Y-axis.

plot2dx

OutVal *(Format*, *x*, “plot2dx”)
Description:
Set X-coordinate of 2D graphical plot. It is used in pairs with *plot2dy.*

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
x y	x-ccordinate y-coordinate	0 to 2.000.000

Table 22: Graph Coordinate Parameters

plot2dy

OutVal *(Format*, *y*, “plot2dy”)
Description:
Set Y-coordinate of 2D graphical plot. It is used in pairs with *plot2dx.*

ploty

OutVal *(Format*, *y*, “ploty”)
Description:
Set Y-coordinate of graphical plot. The X-coordinate is automatically increased by one.

Bitstream Output

The decoder can output a bitstream with a variable bit block length. The default block length is 32 bits.

bitlen

OutVal *(Format*, *Length*, “bitlen”)
Description:
Set length of bit block. The default block length is 32 bits.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
Length	Bit packet length	1 to 65536

Table 23: Bitlen Parameters

bit

OutVal (*Format*, *BitValue*, “bit”)
Description:
Output a bit block with a defined length.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
BitValue	Bit block	Any value

Table 24: Bit Parameters

Mark Output

The decoder can highlight single or continuous bits in the bit view.

markstart

OutVal (*Format*, *StartPosition*, “markstart”)
Description:
Set start position of highlighting.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
StartPosition EndPosition	Bit position to start highlighting Bit position to stop highlighting	Any value

Table 25: Mark Position Parameters

markend

OutVal (*Format*, *EndPosition*, “markend”)

Description:

Set end position of highlighting.

markcolor

OutVal (*Format*, *Colour*, “markcolor”)

Description:

Set highlighting color from go2ANALYSE color table.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
Colour	Highlighting color	Any value

Table 26: Mark Color Parameters

Progress Bar Output

The decoder can open and control a progress bar.

progress

OutVal *(Format*, *Value*, “progress”)

Description:

Set progress bar position between 0 and 100.

Parameter	Description	Admissible Values
Format	Output format	d decimal h hexadecimal o octal b binary
Value	Progress bar position	0 to 100

Table 27: Progress Parameters

Function Workflow

All analyzing steps applied to the current bitstream are logged by go2ANALYSE in a command history, as it is called.

go2ANALYSE can show the *Workflow History* for the current modification steps, replay (repeat) all steps logged in the *Workflow History*, and load and save the *Workflow History* in a file.

View Function Workflow for Current Bitstream

Each analyzing step applied to the current bitstream is logged by go2ANALYSE. All logged steps are viewed in *Workflow History* dock window.

Example:

Open the file *Numb0-15asBits.txt*. Change the circulation length to 64. Apply the following go2ANALYSE functions:

- Tag the first four bits in row 1 with red color
- Autocorrelate the file (<**Autocorrelation**> and <**OK**>)
- Apply OR to the first four bits in row 3 with this pattern: X-X-
- Tag the first four bits in row 18 with blue color

Now the *Workflow History* window should look as follows:

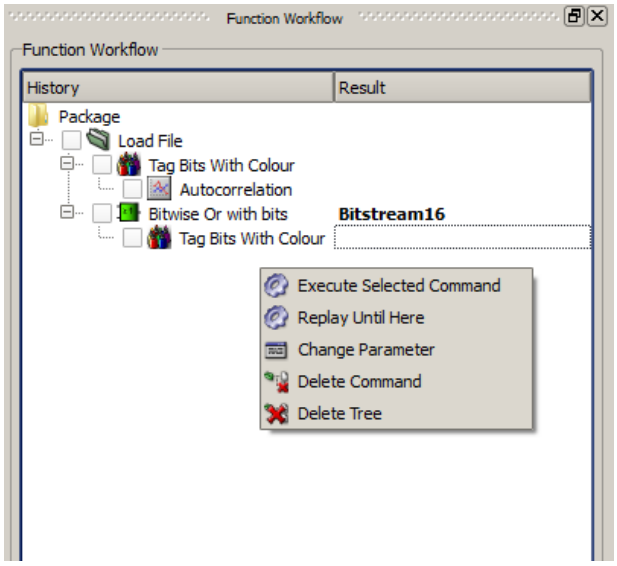


Figure 76: Workflow History

Replay Analysis Steps to File

It is possible to replay the current *Workflow History* partly or completely via the right mouse button context menu. In the same way parameters of each step can be changed before replaying..

Load and Save Workflow History

Workflow History can be saved and loaded under menu item *File*.

Recover Channel Codes

The recovering functions serve to recover the original bitstream from channel-coded bits.

On the *Function Kit,* select the group *Toolbox* and click the button <**Recover**>. The following dialogue box is shown:

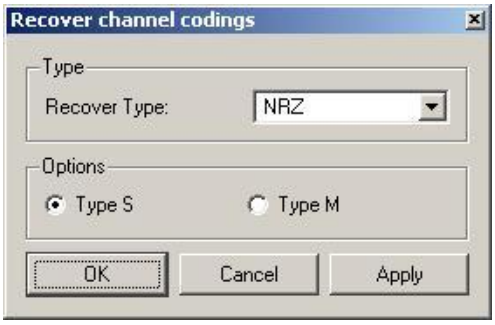


Figure 77: Dialogue Box for Recover

Choose from three different recovering functions by means of the list box *Recover Type*:

- *NRZ* (for no-return-to-zero coded bitstreams)
- *BIPH* (for bi-phase coded bitstreams)
- *Manchester* (sometimes also referred to as *BIPH-L)*

Recover No-Return to Zero: NRZ-M and NRZ-S

go2ANALYSE distinguishes two types of NRZ channel codes: Type M (NRZ-M) and Type S (NRZ-S).

The original bitstream is recovered from a no-return-to-zero coded stream by searching for a change in two successive bits.

Using the option ⊙ *Type M* in the *Recover* dialogue box, the output bit is 1 if there is a change between two successive bits, and 0 if no change occurred.

Using the option ⊙ *Type S* in the *Recover* dialogue box, the output bit is 0 if there is a change between two successive bits, otherwise the output bit is 1.

The *Recover No-Return-To-Zero* function assumes that the bit prior to the first bit of the bitstream is a zero. The bit rate remains unchanged after application of the function. The logical table for the two NRZ types is:

Input Bits		Output Bit NRZ-M	Output Bit NRZ-S
Bit 1	Bit 2	Bit	Bit
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Table 28: NRZ-M and NRZ-S Decoding

Recover Bi-Phase Code: BIPH-M and BIPH-S

The go2ANALYSE distinguishes two types of BIPH channel codes: Type M (BIPH-M) and Type S (BIPH-S).

The original bitstream is recovered from a bi-phase coded stream by calculating the value of the original bit from two successive bits, thus recovering one output bit from two input bits.

Using the option ⊙ *Type M* in the *Recover* dialogue box, the output bit is 1 if there is a change between two successive bits, and 0 if not.

Using the option ⊙ *Type S* in the *Recover* dialogue box, the output bit is 0 if there is a change between two successive bits, otherwise it is 1.

Note: The bit rate after application of the function is divided by 2.

The logical table for the two BIPH types is:

Input Bits		Output Bit BIPH-M	Output Bit BIPH-S
Bit 1	Bit 2	Bit	Bit
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Table 29: BIPH-M and BIPH-S Decoding

Recover Manchester Code: BIPH-L

The original bitstream is recovered from a Manchester-coded stream by calculating the value of the original bit from two successive bits, thus recovering one output bit from two input bits.

The *Manchester Code* is sometimes also known as BIPH-L code.

Note: The bit rate after application of the function is divided by 2.

The logical table for Manchester is:

Input Bits		Output Bit
Bit 1	Bit 2	Bit
0	0	0
0	1	1
1	0	0
1	1	1

Table 30: BIPH-L Decoding

Comparison

The next figure illustrates the recovering functions applied. The first line shows the output bits, the lines below show the input stream before the recovering function has been applied to the stream.

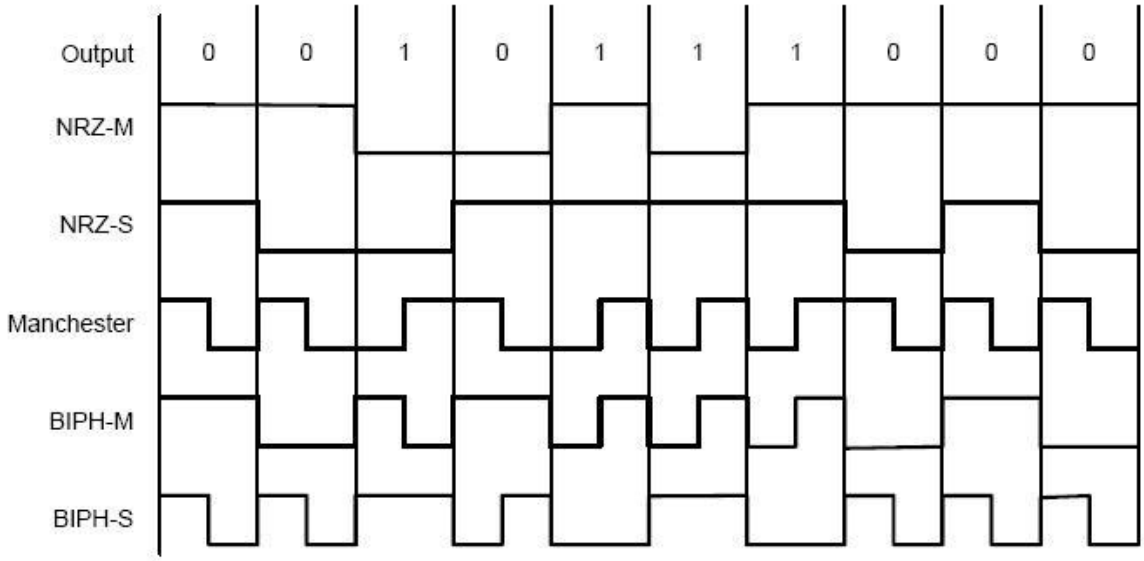


Figure 78: Bit Coding Diagram

Descramble Bitstream

The button <**Descramble**> serves to activate descrambling of the bitstream with a user-defined shift register equation (LFSR polynomial).

On clicking the button <**Descramble**> in the group *Toolbox* in the Function Kit, the following dialogue box is displayed:

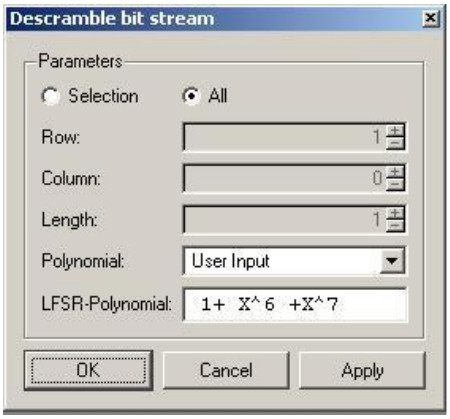


Figure 79: Dialogue Box for Descrambling

Select the option ⊙ *All* to apply the function to the entire bitstream. Choose the option ⊙ *Selection* to apply the function only to the bits defined by the parameters *Row, Column* and *Length* on the *Bit Display* property sheet.

In the list box *Polynomial,* select a predefined polynomial or the entry *User Input* which allows for entering an arbitrary polynomial for descrambling.

The text box *LFSR Polynomial* serves to enter new polynomials or to edit existing entries.

There are four notations to enter polynomials:

- 1+x^-2+x^-3+x^-7
- 1+x^2+x^3+x^7
- 1+x2+x3+x7

- 1+2+3+7

On clicking <**OK**> or <**Apply**>, the bitstream is descrambled using the current LFSR polynomial.

Example:

Load the bitstream file *LFSR1x1x3_0*. This exemplary stream was generated using an LFSR with the equation Bit $_{out}$ = Bit $_{in}$ (1 + x^1 + x^3). Take a look at the block diagram of this scrambler:

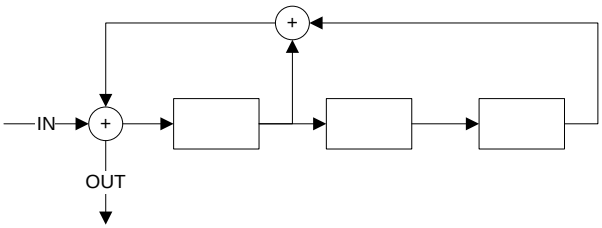


Figure 80: Block Diagram of Scrambler Used in Present Example

The boxes in the diagram represent cells of the scrambler, circles with a plus sign represent Exclusive OR (XOR) functions.

Enter the equation of the scrambler (1 + x^1 + x^3) in the text box *LFSR Polynomial*, select the option ⊙ *All* and press <**OK**> or <**Apply**>. The entire bitstream is descrambled with this type of descrambler:

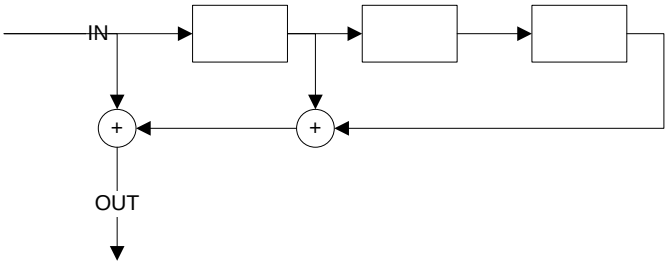


Figure 81: Block Diagram of Descrambler Used in Present Example

Having applied the descrambling function to the current bitstream, all bits of the bitstream are zero except for the second bit which is one.

Deinterleave

go2ANALYSE provides the option to deinterleave bits from an interleaving pattern by means of the *Deinterleaving* function.

On clicking the button <**Deinterleaving**> in the group *Toolbox* in the Function Kit, the following dialogue box is shown:

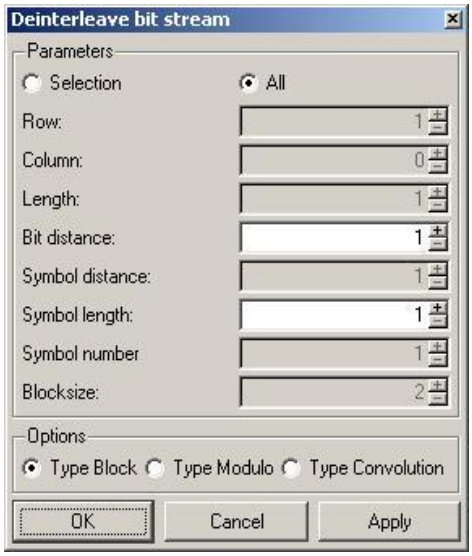


Figure 82: Dialogue Box for Deinterleaving

Select the option ⊙ *Selection* to apply the function to the bits defined by the parameters *Row, Column* and *Length*. Otherwise the function is applied to all bits in the current bitstream.

The parameters Bit Distance, Symbol Distance, Symbol Length and Block Size along with the options ⊙ Type Block, ⊙ Type Modulo and ⊙ Type Convolution define the interleaving pattern.

Recover Block Deinterleaving

A block interleaver normally uses a matrix to interleave the bits.

Assume the matrix has M rows and N columns: The block interleaver reads in N * M bits, the matrix is filled row by row while reading in.

The bits are interleaved by reading out the bits column by column. The following graphics shows an N-M block interleaver.

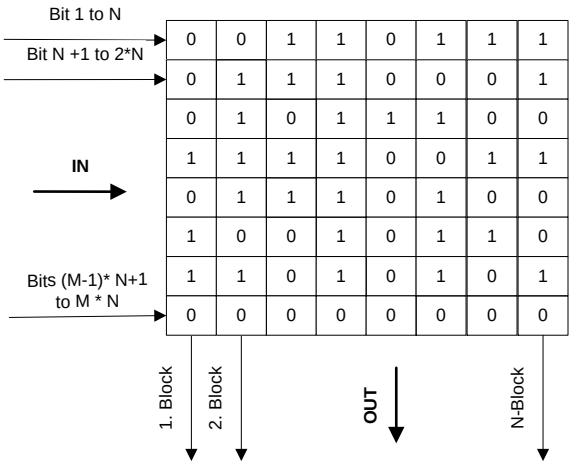


Figure 83: N-M Block Interleaver

Correct deinterleaving of the bits requires N blocks, each M bits long, thus the deinterleaver needs to read in M * N bits which is the block size (as it is called) of this deinterleaver.

The next graphics shows the M-N deinterleaver for the interleaver shown.

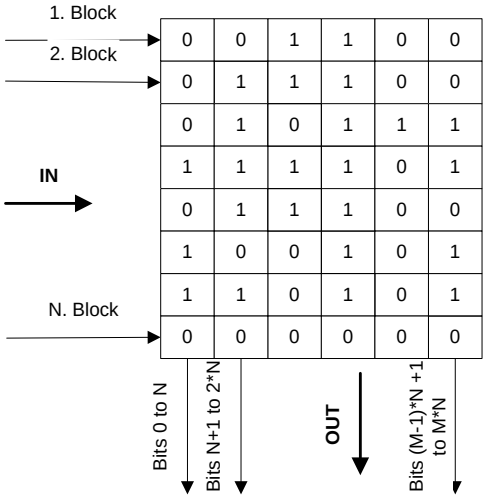


Figure 84: N-M Block Deinterleaver

The deinterleaving function in go2ANALYSE uses two parameters to describe block interleaving: *Bit Distance* and *Symbol Length.*

Bit Distance specifies the distance between two successive bits of the same symbol within the interleaving pattern. The distance is measured in number of bits. *Symbol Length* describes the number of bits for an output symbol.

For the deinterleaver shown, *Bit Distance* is 6 and *Symbol Length* is 8.

The deinterleaver reads the bits from the interleaving pattern by calculating the bit indices for a symbol by means of these parameters.

The next graphics shows the bits of the shown block deinterleaver arranged in a linear array. The arrows below the array indicate the position of the first three bits of the second symbol.

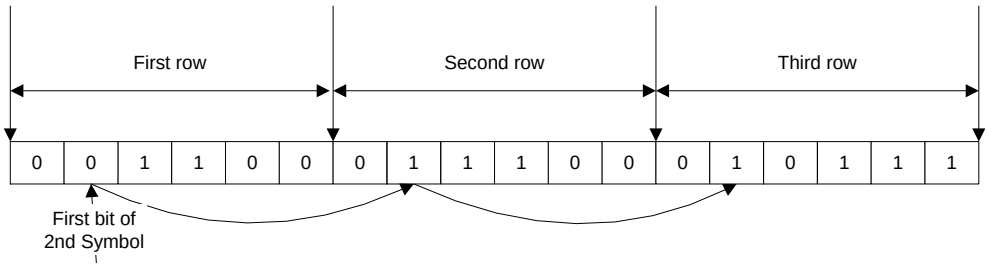


Figure 85: N-M Block Deinterleaver, Bits Arranged in a Linear Array

The current index is calculated using the equation

Index = Start Index for Current Symbol + (Current Bit for Current Symbol – 1) * Bit Distance.

Consequently, the bit index for the third bit of the second symbol for the above example is:

Index = 1 + (3-1) * 6 = 13

The start index of the current symbol is always the start position for the last symbol plus one.

The deinterleaver starts to deinterleave a new block of M*N bits once it has deinterleaved M symbols with a length N.

To deinterleave all bits from the exemplary interleaving pattern using the go2ANALYSE deinterleaving function pattern, enter the following interleaving parameters in the dialogue box:

- Select the option ⊙ All
- Set Bit Distance to 6 and Symbol Length to 8

- Be sure to select the ⊙ *Type Block* option.

Recover Modulo Interleaving

Modulo interleaving is a specialized form of interleaving, i.e. if the calculated bit indices exceed the value of the block size the indices are repositioned at the beginning of the block.

Example:

The parameters are Bit Distance 3, Symbol Length 4, Block Size 11, and Symbol Distance 1.

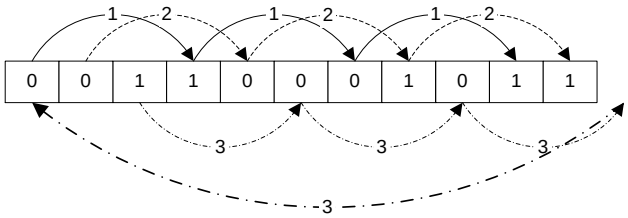


Figure 86: Example of Modulo Interleaving

The readout of the bits starts with the first bits. The numbers on the arrows are the numbers of the current symbol read out from this interleaving pattern.

This table shows the calculation for each bit index for symbols 1 to 3.

Symbol Number	Symbol Start Position	First Bit	Second Bit	Third Bit	Fourth Bit
1	1	1	1+3 = 4	4+3 = 7	7+3 = 10
2	2	2	2+3 = 5	5+3 = 8	8+3 = 11
3	3	3	3+3 = 6	6+3 = 9	9+3 = 12

Table 31: Example of Modulo Interleaving with Symbol Distance 1

Observations on the column *Fourth Bit* for symbol #3 **(bold)**:

The calculated bit indices (12) exceed the block size (11). In this case, the rule stated above applies: If the calculated bit index exceeds the value of the block size, the indices are repositioned at the block beginning.

The rule applies to the bit indices for the fourth bit, thus this index is calculated as:

(Last index	+	Bit distance	) modulo divided	block size	= Bit index for current bit
(9	+	3	) modulo divided	11	= 1

The bit index for the last bit of the third symbol is one.

Note: For the rest of this chapter, the character % is used as the operand for modulo divided.

It is required to define the number of symbols which should be deinterleaved from one block of modulo interleaved bits. If the maximum number of bits which to be read from a block of interleaved data is not defined, the algorithm of the modulo deinterleaver will continue forever.

Deinterleaving of modulo interleaving patterns in go2ANALYSE can be used to change the symbol distance. The symbol distance is defined as the distance between two symbols within the interleaving pattern. Symbol distance means the first bit-distance of two successive symbols. The distance is measured in number of bits.

Normally all modulo interleavers will operate with a symbol distance >1. As a rule of thumb, the symbol distance is a divisor of the block distance.

In all examples of modulo interleaving above, the symbol distance was assumed to be 1.

Changes of the symbol distance will change the start position of each symbol. The following is the result when applying a symbol distance of 2 to the above example:

Symbol Number	Symbol Start Position	First Bit	Second Bit	Third Bit	Fourth Bit
1	1	1	1+3 = 4	4+3 = 7	7+3 = 10
2	1+2 = 3	3	3+3 = 6	5+3 = 9	(9+3) % 11= 1
3	3+2 = 5	5	5+3 = 8	8+3 = 11	(11+3) % 11 = 3

Table 32: Example of Modulo Interleaving with Symbol Distance 2

Note: If the *Symbol Distance* value exceeds the block size of the interleaving pattern, then the symbol distance is calculated by the following equation:

New Start Index = (Last Start Index + Symbol Distance) Modulo (Block Size)

To deinterleave all bits from the exemplary interleaving pattern using the go2ANALYSE deinterleaving function pattern, enter the following interleaving parameters in the dialogue box:

- Select the option ⊙ All
- Set Bit Distance to 3
- Set Symbol Length to 4
- Set Symbol Distance to 2
- Set Symbol Number to 3
- Set *Block Size* to 11
- Be sure to select the ⊙ *Type Modulo* option.

Recover Convolutional Interleaving

Convolutional interleavers use shift registers of a length M and multiples of M, and synchronized switches for interleaving the source bits instead of a matrix, so that the bits can be spread over a larger “distance” than it would be possible by a block interleaver.

All convolutional interleaving patterns have symbol distances greater than one. Also, the bit distances may be very great. Conventional CW modems use bit distances of 129 and up with symbol distances of 16 or more bits.

This table shows a convolutional interleaving pattern with a bit distance of 5 and a symbol distance of 4. The length of each symbol is 4 bits. The input values are integer numbers.

In this example, four integer values are considered to be one symbol.

Symbol Number	Input	Interleaved Output		Symbol Number	Input	Interleaved Output
1	0	0		5	16	16
1	1	?		5	17	13
1	2	?		5	18	10
1	3	?		5	19	7
2	4	4		6	20	20
2	5	1		6	21	17
2	6	?		6	22	14
2	7	?		6	23	11
3	8	8		7	24	24
3	9	5		7	25	21
3	10	2		7	26	18
3	11	?		7	27	15
4	12	12		8	28	28

Symbol Number	Input	Interleaved Output		Symbol Number	Input	Interleaved Output
4	13	9		8	x1	21
4	14	6		8	x	22
4	15	3		8	x	19

Table 33: Example of Convolutional Interleaving

After the interleaving, each symbol is “spread” over a distance of 20 bits.

All convolutional interleavers have one special feature in common: At the start and at the end of the output of the interleaver, the interleaver has to insert undefined bits which are shown here as question marks.

Be careful to deinterleave any convolutional deinterleaving pattern: The deinterleaver will not move these undefined bits - thus they may be overwritten in the process of *Deinterleaving*.

Example:

To deinterleave bits 0 – 20 from the interleaving pattern shown in this table, enter the following interleaving parameters in the dialogue box of the go2ANALYSE deinterleaving function:

- Select the option ⊙ All

Set Bit Distance to 5

- Set Symbol Length to 4
- Set Symbol Distance to 4
- Set Symbol Number to 5
- Set *Block Size* to 20
- Be sure to select the ⊙ *Type Convolutional* option.

Demultiplex

go2ANALYSE can demultiplex bits from the bitstream into several bitstreams. This is necessary to segregate the information of two or more information channels which have been transmitted joint in a single stream.

On clicking the button <**Demultiplex**> in the group *Toolbox* in the Function Kit, the following dialogue box is shown:

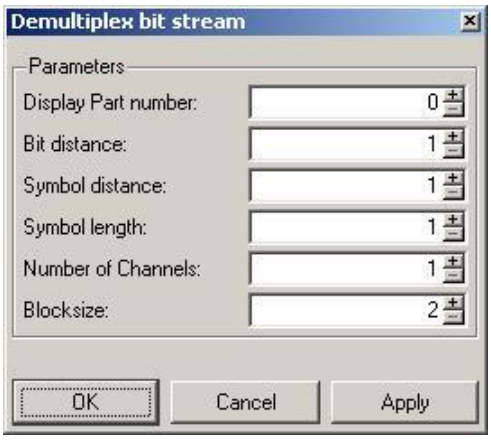


Figure 87: Dialogue Box for Demultiplexing

The function is applied to all bits in the current bitstream.

The parameters are Display Part Number, Bit Distance, Symbol Distance, Symbol Length, Number of Channels and Block Size.

[1] ? means that these input values are not of interest for the rest of the interleaving process

The spin box *Display Part Number* indicates which part of the streams will be shown in the *Bit Display* after the function has been executed.

Number of Channels defines into how many “channels” the current stream will be segregated.

Use the parameters *Bit Distance, Symbol Length, Number of Channels* and *Block Size* to define a pattern by which the bits of the stream will be allocated to the symbols of each channel.

Description:

The function starts to extract the bits for symbol 1 from the stream and writes the information for this symbol to channel 1. Then the bits required for the next symbols are extracted and written to other channels. Once the symbol N (where N = number of channels) has been written to channel N, it starts to extract the bits for symbol N +1 which is destined for channel 0.

Example:

Assume a multiplexed stream which was joined together from four information channels.

The symbols from each information channel have been multiplexed to the current stream as shown in this graphics:

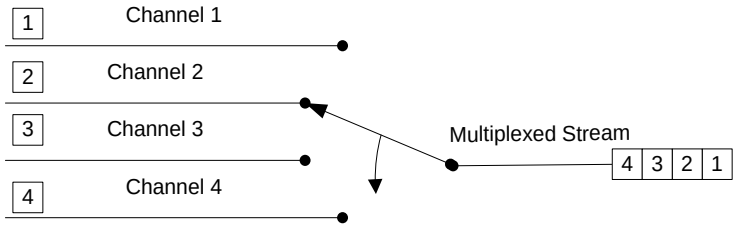


Figure 88: Example of Multiplexing Single Information Channels to a Stream

Assume each symbol consists of 4 bits. To demultiplex this example back into the original information channels, enter the following demultiplexing parameters in the dialogue box:

- Set Bit Distance to 1
- Set Symbol Length to 4
- Set Symbol Distance to 4
- Set Symbol Number to 4
- Set *Block Size* to 16
- To view all symbols of the first channel set the parameter *Display Part Number* to 0

Note: go2ANALYSE cannot display more than one channel of the demultiplexed stream. Therefore, go2ANALYSE saves all “channels” into which the stream is demultiplexed as files in the current working directory. The files are named as the original stream, each file name is extended with the channel number.

Assuming the file name of the exemplary stream is *Example*, the files *Example1* to *Example4* would be found in the working directory of the go2ANALYSE on completion of this example.

Map Bits to Text

Bits can be mapped to text manually in go2ANALYSE. Click the button <**Map Bits to Text**> in the *Toolbox* group in the Function Kit. The *Text Display* is opened showing the text result. The output of the mapping operation varies with the parameter setting on the display's property sheet.

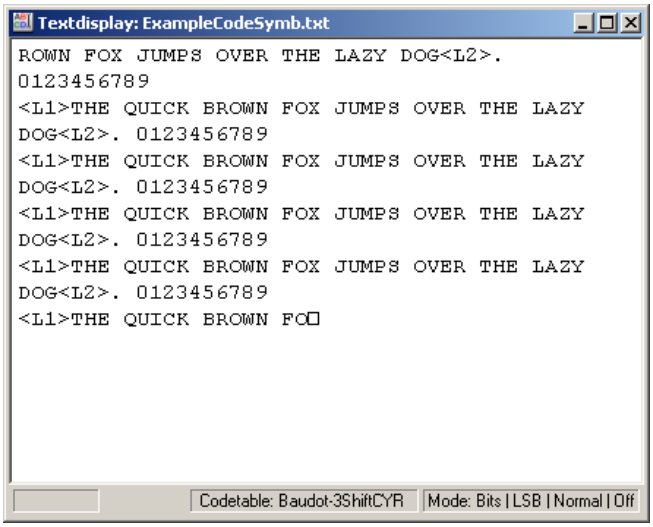


Figure 89: Bit Display Showing Mapping Result

Code tables are created and edited by use of the item *Configure Code Tables* on the *Extras* menu (see View and Edit Code Tables).

Code Tables

Code tables are used to convert bits of a bitstream into readable output. They consist of a code that pairs a set of characters (or code symbols) with a set of bits. Common examples include the code table for *Morse, Baudot* and *ASCII,* which encodes letters, numerals, and other symbols as 7-bit binary set of bits.

go2ANALYSE provides the option to create the code tables required to analyze bitstreams, to modify existing code tables or simply view the definition of available code tables.

All code table operations are controlled by means of the *Code Table* dialogue which opens on selecting the item *Configure Code Tables* on the *Extras* menu or the shortcut <**Alt**>+<**F7**>.

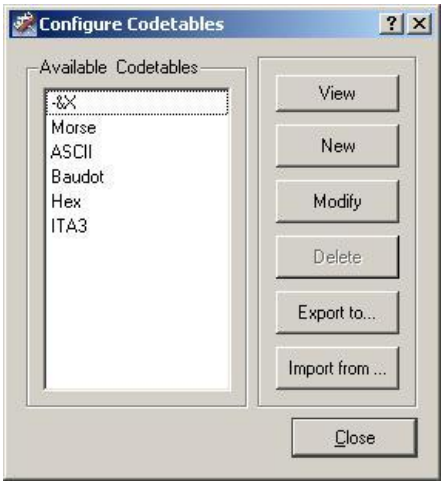


Figure 90: Configure Code Tables Dialogue Box

View and Edit Code Tables

View Code Table

The code tables listed are displayed by selecting the desired table and pressing <**View**>. The following dialogue is displayed:

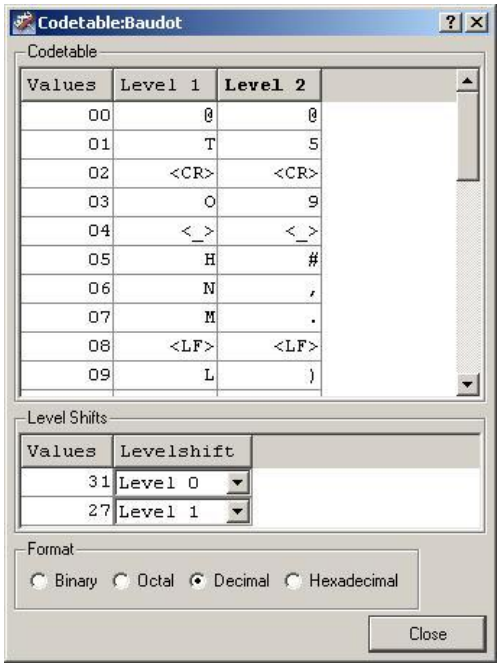


Figure 91: Dialogue Showing the Baudot Code Table

The group box *Code Table* in this dialogue shows the details of the selected code table. As stated above, code tables consist of a set of bits which is paired to a set of characters.

- The set of bits belonging to one set of characters is shown as an integer value in the column *Values*.
- Changing the options in the group box *Format* enables to view the values either in decimal, hexadecimal, octal or binary annotation.
- When selecting ⊙ *Binary*, each value shows the exact set of bits assigned to this value.
- The columns *Level 1* and *Level 2* show the set of characters belonging to the bits.
- For each set of bits defined (shown as decimal values in this example), at least one set of characters has been defined.

The following reserved character sequences have a predefined function:

Reserved Character	Defined Function
<>	No output
<_>	Space character
<LF>	Line feed
<CR>	Carriage return
<L1>	Switch to output level 1
<L2>	Switch to output level 2
<LN>	Switch to output level 3
{value}	Unicode value of symbol (0-65535)
{44}	comma
{59}	semicolon

Table 34: Code Table Reserved Characters

Note: If a code table is used for converting bits to text, then every time a symbol value is found which is paired with one of the reserved character sequences, this special symbol is appended to the text.

Some code tables have more than one defined output level (code table subset). In each subset, all sets of bits defined in this code table have a set of characters assigned.

There must be at least one special symbol, i.e. a level shift as it is called, which switches from one output level to another. The values assigned to these level shifts are displayed in the group box *Level Shifts*.

The group box *Level Shifts* is only shown if the code table currently displayed has more than one level defined.

Edit Existing Code Table

To modify an existing code table, select the respective code table from the list shown in the *Configure Code Tables* dialogue box and press the <**Modify**> button. Existing code tables cannot be edited on pressing <**View**>.

The following dialogue box is displayed (with ASCII code table selected):

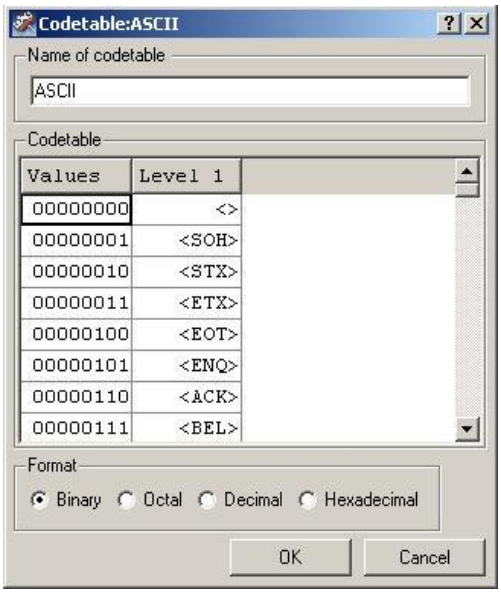


Figure 92: Modifying the ASCII Code Table

Rename Code Table

To rename the code table, edit the entry in the text box in the group box *Name of Code Table*.

Note: When editing the name of the current code table, the new name will be applied to this code table on activating the <**OK**> button. The code table file will be renamed as well. The file with the code table prior to renaming will be deleted.

Change Values and Characters

To modify any value simply double click the value to change and enter the value.

Press <**Return**> after editing the value. go2ANALYSE will verify the entered string and show an alert message:

- If the entered string is not a valid value
- If the same value is currently used in this code table in another row

go2ANALYSE will restrict the entered value to the maximum admissible value for this code table if the code table has a fixed number of bits per symbol. Otherwise the entered values are not restricted to a maximum value.

Example:

Assume an 8-bit ASCII code table. With 8 bits, one can define 2^8 -1 = 255 code symbols.

When entering a value of 256 or higher, go2ANALYSE will restrict the entered value to 255. Once the entered value has been restricted to 255, BPS verifies if this value is already assigned to another code symbol. If so, go2ANALYSE will display an alert message saying this value is used more than once.

To change a character set (symbol) of the code table, select the symbol to change and enter the characters using the keyboard. Mind the special characters mentioned in chapter View Code Table.

Insert Non-Latin Characters Into Code Tables

Some cases may require inserting characters from non-Latin alphabets such as Hebrew, Cyrillic or Arabic into a code table.

If you have a need for such characters in a code table, enter them by means of a special dialogue which provides the option to choose any character from the Unicode character table.

Select the cell in which to insert the special character and click the right mouse button.

A popup menu is displayed showing the items

- Copy
- Paste
- Unicode Character

On selecting the item *Unicode Character* in the popup menu, the following dialogue box is shown:

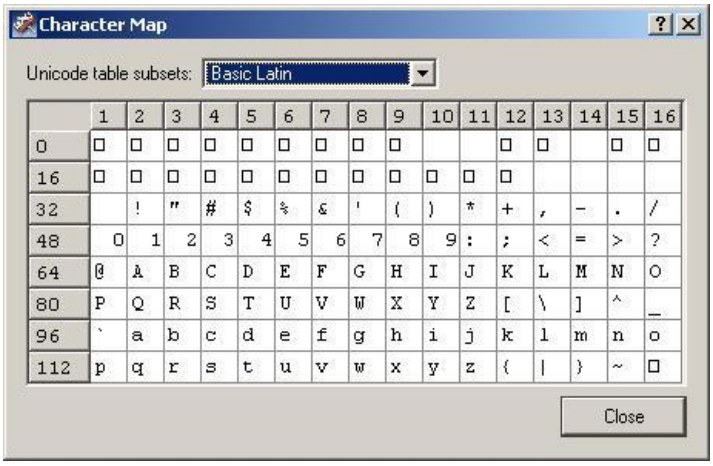


Figure 93: Dialogue Box to Insert Unicode Characters into Code Tables

The drop-down list box *Unicode Table Subsets* enables to choose alphabets such as

- Greek and Coptic
- Cyrillic
- Arabic
- Chinese Japanese Korean (CJK) Unified Ideographs

To insert Unicode characters to the code table, double click the Unicode character to insert.

To change the cell in which to insert the Unicode characters, click the mouse on the desired cell in the *Code Table* dialogue box.

To stop inserting Unicode characters into the code table, click the button <**Close**>.

Complete Editing

To complete the editing of a code table, either press <**OK**> or <**Cancel**>.

On pressing <**Cancel**> all modifications applied to the selected code table will be discarded, otherwise the modifications will be saved to the code table file, and the edited code table is ready for use.

Create and Delete Code Tables

Create Code Table

go2ANALYSE can be used to create new code tables. To create a code table, press the <**New**> button in the *Configure Code Tables* dialogue box. The following dialogue box is shown:

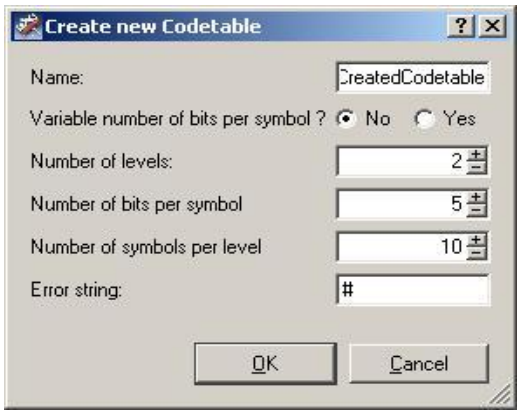


Figure 94: Dialogue Box to Set the Parameters of Code Tables to be Created

Parameter	Defined Function
Name	Name of new code table
⊙ Variable number of bits per symbol	Defines if the code table has a fixed number of bits per symbol or a variable number of bits per symbol. Example of a code table with a variable number of symbols is a Huffman code table.
Number of Levels	Number of code table subsets (levels) of the code table to be created. Note: If the option ⊙ Variable number of bits per symbol is set, the code table is only allowed to have a single level.
Number of Bits per Symbol	Defines the number of bits for each code table. Example: To define a Baudot code table this parameter must be set to 5.
Number of Symbols per Level	Defines the number of symbols in each level of the code table. Example: To define a Baudot code table with a third shift this parameter must be set to 3.
Error String	Defines the error string for this code table. The error string is used if bits are mapped to text via this code table. If the symbol value of the bits does not match any symbol of the code table, the error string will be inserted into the text.

Table 35: Code Table Creation Parameters

On pressing the <**OK**> button the code table created is shown in accordance with the parameters set in the *Create New Code Table* dialogue box. The code table created can now be edited as described in chapter Edit Existing Code Table.

When terminating the editing of the newly created code table by pressing <**OK**>, the code table created is saved to the code table directory of go2ANALYSE and the name of the code table is added to the list of available code tables.

Example: Create Code Table with Fixed Number of Bits per Symbol

Press the <**New**> button.

Enter the name *CreatedCodetable*.

Enter the following parameters in the text and spin boxes of the *Create new code table* dialogue box:

Name:	CreatedCodetable
Variable number of bits per symbol:	No
Number of levels:	2
Number of bits per symbol	5
Number of symbols per level:	10

Press <**OK**>.

The *Create New Code Table* dialogue box is closed and the created code table is displayed:

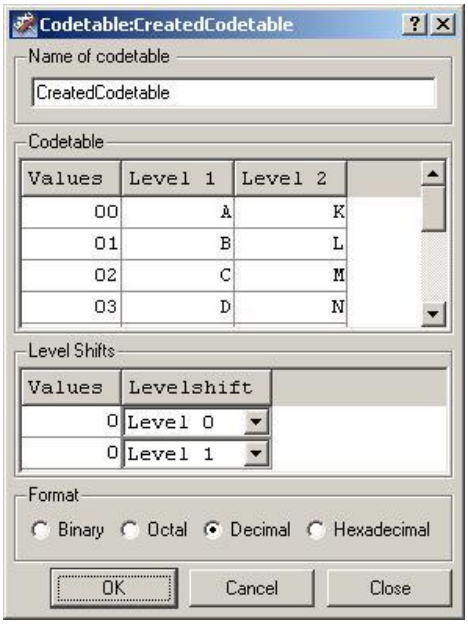


Figure 95: Example of Created Code Table

Example: Create Code Table with Variable Number of Bits per Symbol

Assume a *Huffman* code table must be created. To do so, it must be possible to assign a varying number of bits to each symbol.

Enter the following parameters in the text and spin boxes of the *Create New Code Table* dialogue box:

Name:	CreatedVarCodetable
Variable number of bits per symbol:	Yes
Number of symbols per level:	10

Press <**OK**>.

The *Create new code table* dialogue is closed and the created code table is displayed:

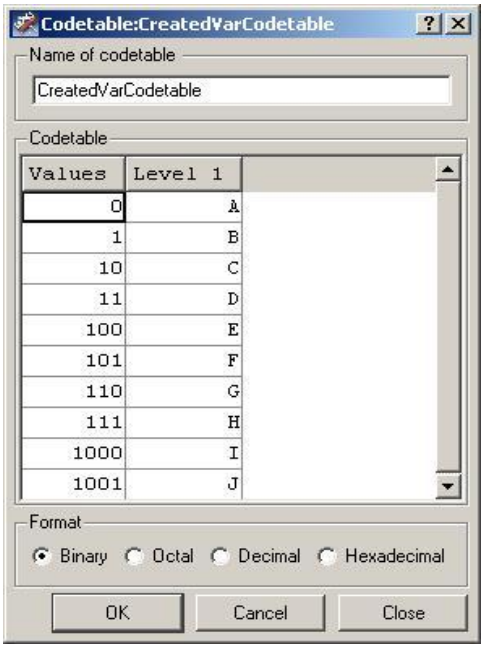


Figure 96: Example of Code Table Created with a Variable Number of Bits per Symbol

Change the option in the group box *Format* to ⊙ *Binary.* As you will see, the symbols have a variable length of symbols.

Delete Code Table

To delete a code table, select the code table to delete and press the <**Delete**> button on the *Configure Code Tables* dialogue box.

Be careful: go2ANALYSE will delete the code table file from the directory where go2ANALYSE is installed.

Import and Export Existing Code Tables

Export go2ANALYSE Code Tables

go2ANALYSE can backup all available code tables by means of the <**Export**> button on the *Configure Code Tables* dialogue box. To do so, press the button <**Export To…**> on the *Configure Code Tables* dialogue box.

go2ANALYSE shows a dialogue box for selection of any directory on your file system, e.g. *C:\CodeTableBackup*.

On clicking the <**OK**> button all files listed in the *Configure Code Tables* dialogue box will be saved to the selected directory.

Import Code Tables from other directories

go2ANALYSE can import code tables as files *(*.ctb* files, see specification of Code Table File Format for details) from other directories into go2ANALYSE. To do so, press the <**Import From...**> button on the *Configure Code Tables* dialogue box.

An import dialogue box is displayed in which **.ctb* files can be selected in all directories of your file system.

Having selected one or several **.ctb* files to import, click the <**OK**> button. The selected files are copied to the current directory.

Each file is loaded by go2ANALYSE and the list of code tables is updated.

Note: go2ANALYSE will display an alert message when attempting to import a code table whose name is identical to the name of a code table already loaded by go2ANALYSE.

When the alert message is acknowledged by <**OK**> go2ANALYSE will show the directory with all code tables available and provide the option to enter a name for the code table to be imported. Please enter a unique name.

Linking External Applications

Any displayed bit stream can be passed to external applications in a quick way. As go2ANALYSE is optimized for pure bit level processing, useful add-ons may be text editors, hex-editors, disassemblers, decompression tools etc.

External applications can be configured by the menu item "Extras" - "Configure External Tool".

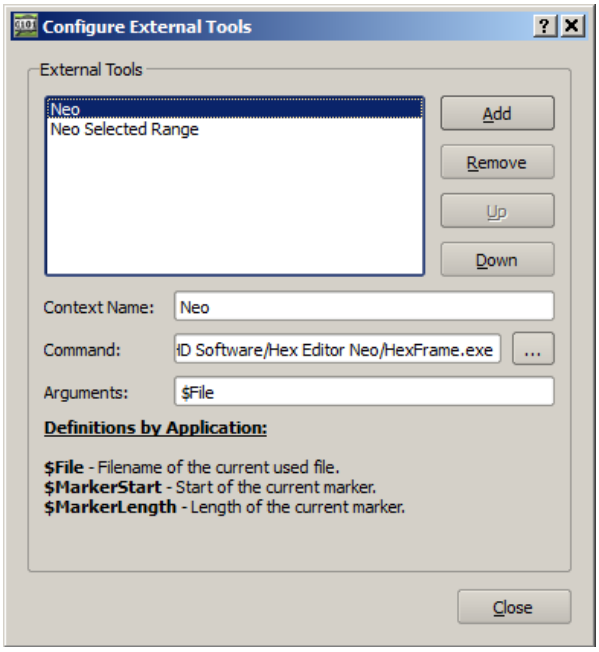


Figure 97: Configuring External Applications

The displayed input fields are used as follows:

Context Name: The application name, which shall be displayed as an additional menu item.

Command: Put in the command as you would start the application via command line.

Arguments: Additional command line arguments needed for application start. The following replacement expressions can be used to pass go2ANALYSE internal parameters:

$File: This will include the name of an internal temporary file generated to transfer the contents of the bit display.

$MarkerStart: The bit displays marker start position (byte number) will be added as an argument

$Marker Length: The bit displays marker length position (number of bytes) will be added as an argument

The following picture shows a setting for the Hex-Editor "Neo", to pass the bit displays contents as well as the marker selection.

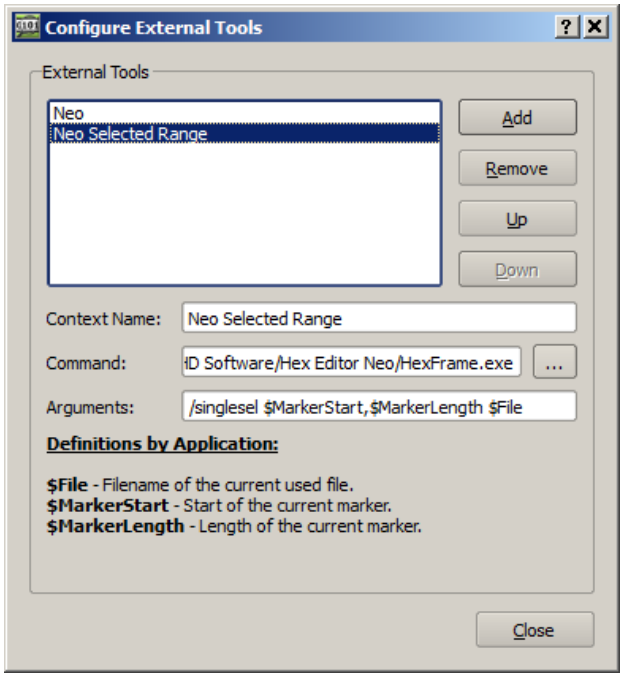


Figure 98: Configuring External Applications with Marker Values

The application can be called from the Extras submenu.

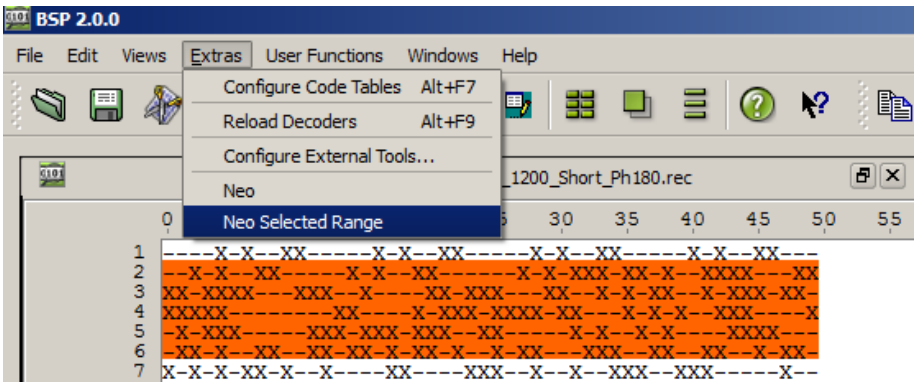


Figure 99: Calling External Functions

go2ANALYSE Components

Main Window

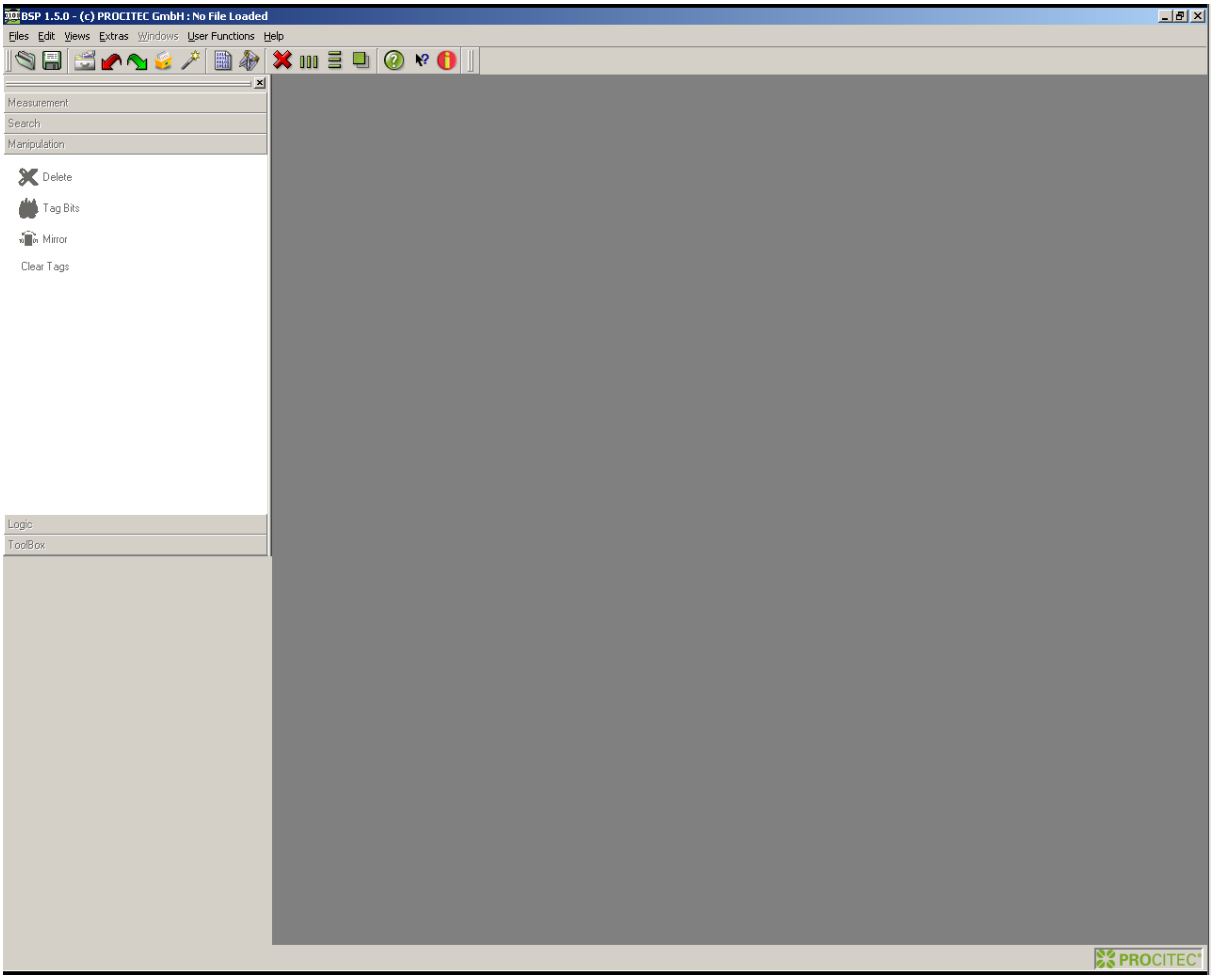


Figure 100: go2ANALYSE Main Window

The main window consists of the menu bar and the toolbar, the workspace and the Function Kit. It provides docking areas to dock windows such as the property sheets of the various displays, the Function Kit, the dialogue box of the go2ANALYSE function and the toolbar.

The buttons of the Function Kit are disabled if no bitstream is loaded. They are enabled when opening a bitstream. They are disabled again when the current bitstream is closed.

Menu Bar

The menu bar provides quick access to important functions.

Menu	Menu Item	Icon	Shortcut	Description
Files				This menu contains all actions with regard to files
	Open		Ctrl + O	Open bitstream in text format
	Save		Ctrl + S	Save current state of bitstream
	Open Workflow History			Open workflow history
	Save Workflow History			Save current workflow history
	Bitstream Cut			Cut oversized .rec file into partial bitstream
	Attributes & Statistics		Alt + T	Open file characteristics and statistics dialogue
	Close File		Alt + C	Close current bitstream and all open displays
Edit				This menu contains undo/redo actions and a list of recent actions
	Redo		CTRL-Y	Redo last action undone
	Undo		CTRL-Z	Undo last action affecting the bitstream
Views				This menu contains actions to open new displays
	Bit Display		F5	Open a blank bit display
	Dock Windows			Show list of available dock windows and open/close the dock windows
Extras				
	Configure Code Tables		Alt + F7	Open the Configure Code Table dialogue box to manage code tables
	Reload Decoders		Alt + F9	Reload decoders in User Functions
Windows				This menu contains actions to manage open windows
	Arrange All			Arrange all open windows side by side
	Tile Horizontally			Arrange all open windows one underneath the other. All windows have the same height.
	Cascade			Cascade all open windows
	Close			Close active window
	Close All		Alt + C	Close all open displays
User Functions				This menu contains decoders definable by the user
Help				This menu contains actions for user help
	Online Help go2ANALYSE		F1	Open the present Operating Manual by means of Adobe® Reader
	Info			Open the About dialogue with Copyright information
	What's this		Shift + F1	The cursor changes into the What's this cursor. The next click on a GUI element will supply online help, if any

Table 36: Menu Bar items

Toolbar

The toolbar displays the major functions of the menu bar in icons. These icons are activated by clicking the respective icon with the left mouse button.

The functions are described in the order in which they appear on the toolbar:

Icon	Function
	Load bitstream
	Save bitstream
	Open attributes and statistics dialogue
	Undo last action affecting the bitstream
	Redo last action made to the bitstream
	Show command history for current bitstream
	Replay history
	Open bit display
	Call bitstream cut function
	Close current file and all windows
	Arrange all open windows vertically
	Arrange all open windows horizontally
	Cascade all open windows
	Open the present Operating Manual by means of Adobe® Reader
	What's This? help
	Show About dialogue

Table 37: Toolbar Icons

Function Kit

The function kit provides quick access to the go2ANALYSE functions.

Function Category	Function	Icon	Description
Measurement			
	Autocorrelation		Analyze bitstream by means of autocorrelation
	Run Analysis		Analyze length of the runs of ones and zeros in the file
	Frame Statistics		Analyze ratio of zeros in ones of frames with user-defined length
	Parity / Weight		Analyze parity and weight of frames with user-defined length
Search			This category contains search functions and functions for verification
	Pattern		Search for specific bit sequence in the stream
	Search LFSR		Search for bit sequence generated by an LFSR with specified

Function Category	Function	Icon	Description
			maximum linear complexity
	Polynomial		Verify if bitstream was generated by LFSR entered
Manipulation			
	Delete		Delete sequence of bits from stream
	Tag Bits		Tag bit sequence with a specific color
	Mirror		Mirror bit sequence at specified position
	Clear Tags		Clear all tags or only tags with a specific color
Logic			
	AND		Logical AND of a specified bit sequence with the bitstream
	OR		Logical OR of a specified bit sequence with the bitstream
	NOT		Invert bitstream at a specified position
	XOR		Logical XOR of a specified bit sequence with the bitstream
Toolbox			This category contains all functions not belonging to other categories
	Descramble		Descramble bitstream with a given LFSR
	Recover		Recover channel-coded bitstream
	Deinterleave		Deinterleave bits from an interleaving pattern
	Demultiplex		Demultiplex bits from the bitstream into several bitstreams
	Map Bits to Text		Map bits manually to text

Table 38: Function Kit Categories

Service

Support

In the event of further questions or problems during the test stage, please do not hesitate to contact:

PLATH AG
Stauffacherstrasse 65
CH-3014 Bern
Switzerland

Phone: +41 31 311 6446
www.go2signals.ch
info@go2signals.ch

After the test stage, i.e. during regular use of the product, we shall provide support within the scope of the Service and Support Agreements concluded on the acquisition of this product.

Training

As a supplement to this documentation, we offer comprehensive training e.g. in decoder creation, manual analysis and the enhancement of automated production.

For additional information on the above training options, please contact your local sales representative.

Requests and Suggestions

Any requests and suggestions about our products will be highly appreciated. We would be glad to receive your information via the support contacts stated above.

Appendix

Keyboard Shortcuts

Function	Shortcut
Open	Ctrl + O
Save	Ctrl + S
Copy	Ctrl + C
Insert	Ctrl + V
Paste	Ctrl + P
Undo	Ctrl + Z
Redo	Ctrl + Y
Close Current File	Alt + C
Exit	Alt + F4
What's This Help	Shift + F1
Open Bit Display	F5
Open Measurement Display	F6
Open Text Display	Alt + F5
Open Attributes and Statistics	Alt + T

Table 39: Keyboard Shortcuts

Built-In Code Tables

Baudot Code Table

Bit-Code	Decimal	Hex	Letter	Figure
00000	0	00	Ignore (@)	
00001	1	01	T	5
00010	2	02	Carriage Return (CR)	Carriage Return (CR)
00011	3	03	O	9
00100	4	04	Space	Space
00101	5	05	H	Space
00110	6	06	N	,

Bit-Code	Decimal	Hex	Letter	Figure
00111	7	07	M	.
01000	8	08	Line feed (LF)	Line feed (LF)
01001	9	09	L	)
01010	10	0a	R	4
01011	11	0b	G	&
01100	12	0c	I	8
01101	13	0d	P	0
01110	14	0e	C	:
01111	15	0f	V	=
10000	16	10	E	3
10001	17	11	Z	+
10010	18	12	D	$
10011	19	13	B	?
10100	20	14	S	\
10101	21	15	Y	6
10110	22	16	F	!
10111	23	17	X	/
11000	24	18	A	-
11001	25	19	W	2
11010	26	1a	J	~
11011	27	1b	Figure Shift >	Figure Shift >
11100	28	1c	U	7
11101	29	1d	Q	1
11110	30	1e	K	(
11111	31	1f	Letter Shift <	Letter Shift <

Table 40: Baudot Code

ITA2P, ITA3, CCIR476 Code Tables

No.	Letter	Figure	ITA2P ARQ1A	ITA3	CCIR476 SITOR
1	A	-	0110001	0011010	0001110
2	B	?	0100110	0011001	1011000
3	C	:	0011100	1001100	0100011
4	D	$	0100101	0011100	0011010
5	E	3	0100000	0111000	1001010
6	F	%	0101100	0010011	0010011
7	G	&	0010110	1100001	0101001
8	H	#	0001011	1010010	0110100
9	I	8	0011001	1110000	0100110
10	J	@	0110100	0100011	0001011
11	K	(	0111101	0001011	1000011
12	L	)	0010011	1100010	0101100
13	M	.	0001110	1010001	0110001

No.	Letter	Figure	ITA2P ARQ1A	ITA3	CCIR476 SITOR
14	N	-	0001101	1010100	0110010
15	O	9	0000111	1000110	0111000
16	P	0	0011010	1001010	0100101
17	Q	1	0111011	0001101	1000101
18	R	4	0010101	1100100	0101010
19	S	~	0101001	0101010	0010110
20	T	5	0000010	1000101	1101000
21	U	7	0111000	0110010	1000110
22	V	=	0011111	1001001	1100001
23	W	2	0110010	0100101	0001101
24	X	/	0101111	0010110	1010001
25	Y	6	0101010	0010101	0010101
26	Z	+	0100011	0110001	0011100
27	Carriage return (cr)	cr	0000100	1000011	1110000
28	Line feed (lf)	lf	0010000	1011000	1100100
29	Letter shift >	>	0111110	0001110	1010010
30	Figure shift <	<	0110111	0100110	1001001
31	Space (sp)	sp	0001000	1101000	1100010
32	Idle α	α	0000001	0000111	1010100
	RQ, Ph 2 β	β	1110000	0110100	1001100
	Idle, beta θ	θ	1001001	0101001	0011001
	Idle, alpha, Ph 1 π	π	1000110	0101100	0000111
	CS1 γ(gamma)	γ			0101100
	CS2 δ	δ			1010100
	CS3 ε	ε			0110010
	CS4 ζ	ζ			0101001
	CS5 η	η			0110100
	Not defined (§)	§	All others	All others	All others

Table 41: ITA2P, ITA3 and CCIR476 Codes

Code Table File Format

The code tables used by the *Text Display* are stored in files with an extension **.ctb* using the following conventions:

- Each code table starts with this qualifier: ALPHA_DEF(N)
- N represents the table identification number, an arbitrary decimal number

Note: Only the first code table of each file will be read by go2ANALYSE. It ends with this qualifier: **END_ALPHA_DEF**

Table Attributes and Table Switches

The following notations are used to store the parameters of the code tables to a file (N always representing a decimal value in full).

BitNo = N length of input code in bits (<= 32)

N = 0 indicates that the length is variable (e.g. Huffman code)
NoLevels = N number of symbol levels (e.g. letters/figures)
ErrSymb = # defines the output-symbol in case of a non-defined input code

List of Table Values

Table values require the following structure of annotation:
Example: User defined code table

```
TABLE
; Input Letter Figure
0 ,                         <> ,<> ; here, comments are allowed
1 , T , 5
2 ,<CR> ,<CR>
3 , O , 9
4 ,<_> ,<_>
5 , H ,<>
6 , <IDLE> ,<44>
7 , M , .
8 ,<LF> ,<LF>
9 , L , )
10 , R , 4
11 , G , &
12 , I , 8
13 , P , 0
13 , P , 0
14 , C , :
15 , V , =
16 , Number16 , 3
17 , Z , +
18 , D ,<WRU>
19 , B , ?
20 , S , '
21 , Y , 6
22 , F ,<>
23 , X , /
24 , A , -
25 , W , 2
26 , J ,<BEL>
27 ,<L2> ,<L2> ; figure shift
28 , U , 7
29 , Q , 1
30 , K , (
31 ,<L1> ,<L1> ; letter shift
ENDTABLE
```

The table input values are placed in the first column either in decimal, hexadecimal, octal or binary annotation.

The number of additional output columns must be first predefined by means of *NoLevels* in the table attribute section. Output values may consist of up to 50 characters.

Any comments in a **.ctb* file will be ignored by go2ANALYSE.

The following reserved character sequences have predefined functions:

Reserved Character	Defined Function
<>	No output
<_>	Space character
<LF>	Line feed
<CR>	Carriage return
<L1>	Switch to output level 1
<L2>	Switch to output level 2
<LN>	Switch to output level 3
{value}	ASCII value of symbol (0-255)
{44}	comma
{59}	semicolon

Table 42: Code Table Reserved Characters

For reasons of general syntax, the characters “,” (comma) and “;” (semicolon) must be edited with their ASCII annotation {44} and {59} exclusively.

Interfaces

Format for Input/Output Bitstreams

The format for input/output bitstreams is the *.txt* format with characters according to the following conversion table:

Character	Converted Bit
0	0
- (Dash)	0
_ (Underline)	0
L	0
. (Dot)	0
1	1
X	1
x	1
H	1

Table 43: Bitstream Conversion Table

Any other characters in the text file will be ignored!

Glossary of Terms

AF

Audio **F**requency 40 … 20000 Hz

Audio **F**requency is the range of acoustic waves which the human can perceive, in contrast to ultrasonic waves, which humans cannot hear.

ALE

Automatic Link Establishment, commonly known as ALE, is the worldwide de facto standard for digitally initiating and sustaining HF radio communications. ALE is a feature in an HF communications radio transceiver system that enables the radio station to make contact, or initiate a link between itself and another HF radio station or network of stations. The purpose is to provide a reliable rapid method of calling and connecting during constantly changing HF ionospheric propagation, reception interference, and shared spectrum use of busy or congested HF channels.

ARQ

The **Automatic Repeat reQuest** protocol **ARQ** is a method to increase the reliability of data-transfer. The data to transfer is split into smaller packets, each packet is extended by a packet-number and a checksum. On the receiving side a checksum is generated on the received data-part of the packet and compared to the checksum that was sent. If they do not concur, the receiving station sends a message to the originator of the message, reporting which packet failed. This message prompts the originator to send the indicated packet another time.

ASCII

The **American Standard Code for Information Interchange** commonly known as **ASCII** is a character-encoding scheme originally based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that use text.

ASCII evolved from telegraphic codes. Its first commercial use was as a seven-bit code for teleprinting promoted by Bell data services.
ASCII includes definitions for 128 characters: 33 are non-printing control characters (many now obsolete) for formatting and 95 printable characters, both upper and lower case.

Signals consist of 1 start bit, 7 or 8 data bits, 1 or 2 stop bit and optionally a parity bit, thus each character consists of a total of 9, 10 or 11 bits.

BCH

In coding theory the **BCH codes** form a class of parameterized error-correcting codes. BCH codes were invented in 1959 by Hocquenghem, and independently in 1960 by Bose and Ray-Chaudhuri. The acronym BCH comprises the initials of these inventors' names.

Reed–Solomon codes, which are BCH codes, are used in applications such as satellite communications, compact disc players, DVDs, disk drives, and two-dimensional bar codes.

In technical terms a BCH code is a multilevel cyclic variable-length digital error-correcting code used to correct multiple random error patterns. BCH codes may also be used with multilevel phase-shift keying whenever the number of levels is a prime number or a power of a prime number. A BCH code in 11 levels has been used to represent the 10 decimal digits plus a sign digit.

Context menu

A context menu (also called contextual, shortcut, popup or pop-up menu) is a menu in a graphical user interface (GUI) that appears upon user interaction, such as a right-click mouse operation. A context menu offers a limited set of choices that are available in the current state, or context, of the operating system or application. Usually the available choices are actions related to the selected object.

dB

Decibel (Symbol: dB) is a logarithmic unit that indicates ratio or gain. Decibel is used to indicate the level of acoustic or electromagnetic waves or electronic signals. The logarithmic scale can characterize very big or very small numbers with short notation. The dB level can be viewed as relative gain or attenuation of one level vs. a second, or absolute logarithmic scale level for well known reference levels.

Decibel is a dimensionless unit.

The ratio in Bel is the base 10 logarithm of the ratio of P1 to P0:

Ratio (dB) = $10*\log_{10}(P1 / P0)$

DDC

In digital signal processing, a **D**igital **D**own-**C**onverter converts a digitized real signal centered at an intermediate frequency to a baseband complex signal centered at zero frequency. In addition to down-conversion, DDC's typically decimate signals to a lower sampling rate.

DDL

The **D**ecoder **D**escription **L**anguage is a programming language developed by Procitec for the easy implementation of modems. A compiler converts the source-code into binary intermediate code, which is interpreted by the application.

DHCP

The Dynamic Host Configuration Protocol (DHCP) is a network protocol used to configure devices that are connected to a network (known as hosts) so they can communicate on that network using the Internet Protocol (IP). It involves clients and a server operating in a client-server model.

FEC

The **F**orward **E**rror-**C**orrecting code is a method to increase the reliability of data-exchange. Additional data is appended to the original data which can be used to correct data if they are partly corrupted. This technique is applied in cases where there is no channel for back-reporting, e.g. in a broadcast situation. It is used as well in situations where the switch-over and retransmission time by far exceeds the time to generate, transfer and evaluate the correction code (deep space communication).

FFT

The **F**ast **F**ourier **T**ransformation is a variant of the Fourier transformation. This is a method to convert data between time- and frequency-domain. Data are sampled in the time-domain, in many applications they are transformed into the frequency-domain for further processing.

The DFT is the discrete variant of the Fourier transformation. It works with every integer number N of samples and requires N^2 operations. The FFT is a special variant, where N is 2^m, m being an integer. In this case only $N*\log_N$ operations are required, accelerating processing significantly for larger N.

HF

High **F**requency 3 … 30 MHz

This is the frequency range for world-wide information-transfer over radio with low bandwidth. Propagation in this range is marked by reflections of the waves in the ionosphere, a layer which encloses planet earth at a high of about 60 to 600 kilometers. This way almost every 2 points on earth can exchange information sometimes within 24 hours, either by ground wave or via reflected waves. The ionization depends heavily on the solar radiation, so the available propagation-paths are a function of the time of the day.

I/Q

I/Q data are signals represented in the complex plane by their **I**nphase and **Q**uadrature parts. While the sole amplitude information of a signal is ambiguous regarding the phase, the combination of I and Q data identifies the phase positively. The magnitude of an I/Q signal is the square-root of $(I^2 + Q^2)$, the phase is arcsin(I) plus the quadrant information derived from Q.

LF

Low **F**requency 30 … 500 kHz

This is the frequency range for medium-range information-transfer over radio with low bandwidth. Propagation in this case is restricted to ground-waves, so the coverage is limited to a few 100 kilometers.

Modem

Modem is an abbreviation for **mo**dulator / **dem**odulator. It characterizes a device which is used to transfer information over radio, telephone- or fiberglass-line. The information which is to be sent is adapted to the channel so that it can be retrieived as reliable as possible, given the characteristics of the channel.

RCM

Receiver Control Modul (receiver.exe)

SLEW

Link11 is a NATO standard exchange of for tactical data over radio. **S**ingle Tone **L**ink **E**leven **W**aveform is a variant with extended data protection by interleaving and convolutional block coding.

SNR

Signal-to-noise ratio (often abbreviated SNR or S/N) is a measure used in science and engineering that compares the level of a desired signal to the level of background noise. It is defined as the ratio of signal power to the noise power. A ratio higher than 1:1 indicates more signal than noise. Normally the values are indicated in dB.

UHF

Ultra **H**igh **F**requency 0.3 … 3 GHz

This is the frequency band for information-transfer with high bandwidth. Due to the quasi-optical wave propagation the range is limited to about 10 kilometers for omnidirectional antenna systems, and to line-of-sight links in case of directional antennas.

VHF

Very **H**igh **F**requency 30 … 300 MHz

This is the frequency range for information-transfer with medium bandwidth. Due to the more or less quasi-optical wave propagation the range is limited to some 10 kilometers for omnidirectional antenna systems, and to close to line-of-sight links in case of directional antennas.

XSLT

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents, or other objects such as HTML for web pages etc.,

The original document is not changed; rather, a new document is created based on the content of an existing one.

List of Figures

List of Tables

Index

U

V

X

Z