



Dragon Suite

Software Documentation

Inhaltsverzeichnis

1 Copyright	5
2 Dragon Suite Change Log	6
3 Documentation History	18
4 Introduction	19
4.1 Symbols	19
4.2 Liability and Warranty Exclusion	19
5 Installation	20
5.1 Supported Hardware	20
5.2 Prerequisites	20
5.2.1 System Requirements	20
5.2.2 Hardware Installation	20
5.3 Software Installation	20
5.4 Update Manager	21
6 Using the GUI	23
6.1 Menu Bar	23
6.2 Toolbar	24
6.3 Interface Tree	25
6.4 Main Frame	27
6.5 Message Box	27
7 Setting up the Frame Generator	29
7.1 General	30
7.2 Signal Levels	31
7.3 Signal Routing	32
7.4 SerDes Config	33
7.5 External Board	35
7.6 Sideband Settings	37
7.6.1 UART	38
7.6.2 I ² C	38
7.6.3 SPI	39
7.7 LVDS Channels	40
7.7.1 Open LDI Mode	41
7.7.2 RxTx Loop	42
7.8 IO Routing	42
7.9 Ethernet	42
7.9.1 Example	44
7.10 LVDS Info	45
8 Frame Generator Dialog Window	47
8.1 Files on the Device	48
8.2 Display color	48
8.3 Display Direct	49
8.4 Video Preview	50
8.5 Pattern Generator	50
8.5.1 Advanced Pattern Generator	52
9 Setting up the Frame Grabber	53
9.1 General	54
9.2 Signal Levels	55
9.3 Signal Routing	56
9.4 SerDes Config	57
9.5 External Board	57
9.6 Sideband Settings	57
9.7 LVDS Channels	57
9.7.1 Open LDI Mode	60
9.7.2 RxTx Loop	60

9.8 IO Routing	62
9.9 Ethernet	62
9.10 LVDS Info	62
9.10.1 LVDS Information of basicCON 4121	63
9.10.2 LVDS Information of Video Dragon 6222	63
10 Frame Grabber Dialog Window	65
10.1 Tool Box	66
10.2 Capture Settings	66
10.3 Compare Settings	67
10.4 Avi Settings	68
10.5 Raw Data Recording	68
10.6 Frame Area	69
11 Sideband Communication	70
11.1 Configuration of Sideband Communication via G-API	71
11.1.1 Setting the Data Mode	71
11.1.2 Setting the Parameters	71
11.2 I ² C Master Mode	72
11.2.1 Communication	72
11.2.1.1 I ² C Transfer on the Bus	72
11.2.1.2 I ² C Transfer by Command	73
11.2.1.3 I ² C Protocol	74
11.3 I ² C Slave Mode	74
11.3.1 Communication	74
11.3.1.1 I ² C Slave Definition by command	75
11.4 SPI Mode	75
11.5 SPI Master Mode	75
11.5.1 Communication	76
11.5.1.1 SPI Transfer on the Bus	76
11.5.1.2 SPI Transfer by Command	76
11.6 SPI Slave Mode	77
11.6.1 Communication	77
11.7 SPI Dual Mode	77
11.7.1 Communication	78
11.7.1.1 SPI Transfer by Command	78
11.8 UART Mode	78
11.8.1 Communication of basicCON 4121	78
11.8.1.1 UART Transfer by Command	78
11.8.2 Communication of Video Dragon 6222	79
11.9 Sideband Communication Tool	80
11.9.1 UART	81
11.9.2 I ² C	82
11.9.3 SPI	83
11.9.4 Indigo	84
12 File Manager Tool	85
13 IO Tool	86
13.1 Digital IO	86
13.2 Trigger	87
13.2.1 SerDes GPIO	90
13.2.2 Examples	91
14 CAN Tool	93
14.1 CAN Node	93
14.2 CAN - UART Gateway	94
14.2.1 Example	94
15 Sequence Interface	97
16 Command Line Interface	98
17 Dragon Suite Advanced	99

17.1 Script Interface	100
17.2 Raw Data Recording	103
17.3 Monitor Dialog	105
17.3.1 CAN Monitor	105
17.3.2 SPI Monitor/ SPI Analyzer	106
17.3.2.1 Example for SPI Monitor	109
17.3.3 UART Monitor/ UART Analyzer	111
17.3.3.1 Example for UART Monitor	111
17.4 MiMfp Config Tab	113
18 Additional Features	115
18.1 TCP Remote Control	115
18.1.1 Server Settings	115
18.1.2 Client Settings	116
18.1.3 Remote control	116
18.1.4 Tray mode	119
19 First Steps	120
19.1 System Structure	120
19.2 Registration	120
19.3 Configuration	121
19.4 Capturing	123
20 Common Error Messages	124
21 Service and Support	125
21.1 Spare Parts and Accessories	125
21.2 Warranty and Repair	125
21.2.1 Conditions	125
21.2.2 Identification	125
22 Disposal	126
22.1 Disposal of used Electrical / Electronic Equipment	126
22.2 Disposal of used Disposable / Rechargeable Batteries	126

1 Copyright

Copyright © 2024 *GÖPEL electronic* GmbH. All rights reserved.

The software described in this manual as well as the manual itself are supplied under license and may be used or copied only in accordance with the terms of the license. The customer may make one copy of the software for safety purposes.

The contents of the manual is subject to change without prior notice and is supplied for information only. Hardware and software might be modified also without prior notice due to technical progress. In case of inaccuracies or errors appearing in this manual, *GÖPEL electronic* GmbH assumes no liability or responsibility.

Without the prior written permission of *GÖPEL electronic* GmbH, no part of this documentation may be transmitted, reproduced or stored in a retrieval system in any form or by any means as well as translated into other languages (except as permitted by the license).

GÖPEL electronic GmbH is neither liable for direct damages nor consequential damages from the company's product applications.

Version: 2.0.12

Printed: 2/26/2024

All product and company names appearing in this manual are trade names or registered trade names of their respective owners.

2 Dragon Suite Change Log

New Release 1.994

New features:

- added LockInfo for DS90UB913 to LVDS_Info_Dialog
- added colorformat YUV for PixelMode RGB888
- added DT - Compressed Data Stream to channel settings Dialog for Mipi DSI SerDes to settings dialog
- added LVDS Info - added Pixelmode to all interfaces

Advanced:

- added capture from URL (UDP, RTP)

ScriptInterface:

- added bool Ethernet_Dlt_Config(u32_t txFifoid, u8_t spiA_nodId, u8_t spi_slaveSelect, u16_t readIntervalInMs, u8_t dltSourceType, u8_t dltSendMode);
- added bool Ethernet_Dlt_Start();
- added bool Ethernet_Dlt_Stop();
- added Ethernet_Dlt_Property_GetById
- added Ethernet_Dlt_Property_SetById
- added optional portHandle to DLT functions
- added SPI Monitor to script
- added APIX Lockstate command with and without portHandle
- added Ethernet_Dlt_SendMessage

Bugfix:

- memory leak displayBufferAdv at Bayer colormode
- Pixelmode settings for VD1
- pixel mode handling in grabber dialog

New Release 1.964

New features:

- IO-routing dialog: added Trigger sources/targets for I2C slave
- Settingsdialog: added I2C Slave settings, added to config file (DS only, API later)
- I2C side band dialog: update G_Lvds_DataRegisterReadWithSTOP also for 16bit register address width
- Side band dialog - added DataMode selection
- Settings dialog: added apply GMSL2 Register settings to apply (SerDesConfig), DS only - API later

Advanced:

- generator settings: added timing adjustment for internal mode, later planned for advanced

ScriptInterface:

- added Lvds_Common_Data_I2cTransfer(QStringList TxData, u8_t SendStartMask, u8_t NumberOfRxBytes)

Bugfix:

- I2C side band dialog, writing more than 1 byte.
- script: DisplayLocalFile_Adv crash if show=true

New Release 1.942

New features:

- Framgrabbersettings: added TransferMode to channelsettings
- RxTx Loop config parameter in gen/grab config dialog
- Lvs Info - added PLL enable Info for DS90UB988
- Lvs Info - Transfermode for generator
- update to API 2.2.9855 RxTx Loop in config file
- Framegrabber -> DMA Multichannel support with automatic channel selection
- IP Info Dialog: added I2C Slave capability

Advanced:

- TCP automation - added tray icon for hidden server mode

ScriptInterface:

- added Lvds_FrameGenerator_RxTxLoop_Config
- added G.Lvds_FrameGrabber_Oldi_PLL_enable_Get() + G.Lvds_FrameGrabber_Oldi_PLL_enable_Set(bool enable)
- added Lvds_FrameGenerator_MIPI_CONT_CLK_Get/Set for CSI and DSI generator
- added Script command to clearScriptOutput

Bugfix:

- Framegenerator -> Bugfix generate images with odd resolutions
- I2C Dialog - ReadModWrite at 8 bit address width

New Release 1.922

New features:

- FramegeneratorSettings -> added start button to start the generator without setting the display parameter
- FramegeneratorSettings -> added SerdesConfig for GMSL2 chips - import csv config from MAXIM GUI (not saved to config file and applied yet)
- FramegeneratorSettings -> gmsl Registertable -> export to I2C config file (new version with mask)
- SidbandDialog->I2C added Mask+Comment, readModifyWrite possible (8 bit reg value)
- SidbandDialog->I2C -> combine successive I2C registers for writing (increase performance also for scripting)
- new gconf since this version (Version 3)!!!

Advanced:

- FrameGeneratorDialog-> added Colorformat BAYER_GB, BAYER_GR to DisplayBufferAdv and Dialog

ScriptInterface:

- added Lvds_FrameGenerator_CSI2_Cfg_Num_Lanes_Get + Set
- added u32_t Common_ShellExecuteEx(QString path,u16_t timeout,u8_t windowMode, bool asAdmin) - waits until the application finished or timeout
- added Lvds_SerDes_Interface_CfgNumLanes_Get/Set generic for Gen/Grab and CSI/DSI/OLDI
- added Lvds_Apix_SerDes_State_Get() to get Apix State Pin 1+2

Bugfix:

- FrameGeneratorSettingsDialog Videosettings VD1
- IO Dialog-> select trigger connection, show connection details
- delay in Script idle CPU load
- lineTime_Get
- LVDS Info for 4xMAX9295
- captureDialog resolution not divisible by 4
- settingsdialog, generator dialog minimum size adjustments

New Release 1.902

New features:



- added IP Info Dialog for VD2's

Skriptinterface:

- added Can_CyclicMsgs_Ramp_Define + Can_CyclicMsgs_Ramp_Delete
- added Lvds_Apix3_Phy_State_Save
- added Lvds_Apix3_Phy_Registers_Set for fast startup APIX3 config

Bugfix:

- CAN Monitor sort
- Import old generator config file (V1.0)
- Script: G.delay -> waits µs

Based on G-API 2.2.9519

New Release 1.892

New features:

- LvdsInfo: added - GMSL2 Linkspeed for MAX9295/96
- LvdsInfo: added - OLDI_speed for DS90UB947
- LvdsInfo: added status LED's for Video Tx Pixelclock also for pipe Y U H (MAX9295 only)
- Trigger Settings -> added control to set SerDesGpio_out
- added Serializer 96793
- GrabberDialog - added loading DAT files

Advanced:

- GrabberDialog : dialog/script add color tolerance for compare function similar to 4121
- GeneratorDialog: added companding for color format BAYER_BG_20To12Bit_Comp(20), BAYER_RG_20To12Bit_Comp(20)
- GeneratorDialog-> added color format YUV_YUYV+ YUV_UYVY @16 bit pixel mode
- GeneratorDialog-> added YUV_8bit for pixel mode raw12 and raw8
- GeneratorDialog-> added color format Bayer_RG, Grey_8, Grey_12
- GrabberDialog -> added new color format raw10 -> grey10

Skriptinterface:

- added G.Lvds_DisplayColorAdv(u32_t width, u32_t height, u8_t colorFormat, u8_t pixelmode, QList<u8_t> color) to use displayColor also for Pixelmode raw 8/12/16 in Bayer_BG format
- added G.sendJsonStringToClient(QString JSONString) - sends a JSON Sting as SIGNAL. e.g. to forward to TCP connection or other application .
- added G.Lvds_FrameGrabber_Capture_Get_FrameCounter(u8_t portHandle, u8_t channel);
- added G.LLvds_FrameGrabber_Capture_Get_FrameCounter(u8_t channel);
- added G.LLvds_FrameGrabber_Capture_Get_FrameCounter(QString InterfaceName, u8_t channel);
- added G.Lvds_SetCompandingPoints([x,y,x1,...xn,yn])

Bugfix:

- GeneratorDialogr: optimized generator 12 bit BayerPattern (normalized to 12 bit)
- GeneratorSettingsDialog Bugfix oldi generaror
- settings: skip reading/writing MiMfp+SerDesGpio if not advanced
- GrabberDialog -> Speichern von Bilder mit BGR Swap nach Single shot
- GrabberDialog -> CaptureDialog > show in new window BGR swap
- GrabberDialog -> remember last selected color format
- Bugfix UART Monitor 2nd Instance

New Release 1.849

New features:

- Framegenerator Dialog: Increased resolution for display file and display color to 8192x8192
- Framegenerator settings: enabled set/get for pixelmode

Advanced:

- added G.Lvds_APIX3_BstBwModeSet(speed)) ;/* set bwSpeed 1=1,5 ; 2=3Gbit/2 ; 3=6Gbit/s*/ on FPGA Level interim
- G.Lvds_APIX3_BstBwModeGet()); //reads Deserializer Register
- Framegenerator: generate Bayer_BG for pixelmode Raw12 and raw16 on VD2

Bugfix:

- Bugfix Select png in display local file filedialog

Release 1.840

New features:

- new Help files with change log

Advanced:

- Script addedCommon_ShellExecute(QString path,u8_t windowMode, bool asAdmin); to execute an external application

Bugfix:

- Lvds_FrameGenerator_CSI_Dt_Get disabled for VD1
- generator settings Bugfix Host DeviceType while loading
- load generator settings from file
- description for overloaded functions
- ScriptInterface - Lvds_Common_Data_Register_Write

Release 1.826

New features:

- Settings: Generator configuration
- Generator Settings -> added channels settings DT/pixelmode settings for CSI, DSI and OLDI limited to one channel for now
- LVDS Info -> added Generator Framecounter total
- added Type G_LVDS__COMMON__SER_DES__DS90UB971 + 981
- Framegenerator dialog -> disabled Display File for VD2, will not be implemented to firmware
- added Sequence dialog - basic functions on right mouse menu
- added UserCode dialog - basic functions on right mouse menu
- added Ethernet dialog with MIMUX Gui

Advanced:

- Script: added Lvds_FrameGenerator_CSI_Dt_Get/Set
- G.Lvds_FrameGenerator_Simulation_Event_Control_Event1_enable_lowlevel(bool); //FPGA based
- G.Lvds_FrameGenerator_Simulation_Frame_Counter_Control_FC_reset_lowlevel(); //FPGA based
- G.Lvds_FrameGenerator_Simulation_Frame_Counter_Control_FC_enable_lowlevel(bool); //FPGA based
- G.Lvds_Common_VideoOutputAutoReSync_Request_lowlevel(). //FPGA based
- G.Sequence_Replay_AutoPlayByInput_FileName_Set(u8_t index, QString filename);
- G.Sequence_Replay_AutoPlayByInput_FileName_Get(u8_t index);
- G.Sequence_Replay_AutoStartSequenceHandleGet(u32_t *seqHandle);

- Monitor dialog: added menu to select a type (CAN;SPI, UART)

Bugfix:

- Guiless: bugfix Syncscan
- Framegrabber: bugfix Compare 30bit images
- removed auto set datamode on combobox change
- Settings dialog: bugfix APIX mode
- FrameGenerator configuration -> added Lvds_Generator_ChGenEnReq(0,false); to stop the generator pipe for channel 0
- Generator Settings: apply HFrontPorch
- Settings -> bugfix load/save/show apix register comments

Release 1.801

New features:

- support save 30 bit image as 48 bit Tiff or png
- GrabberConfig -> re-added OLDI DT settings
- CAN Dialog: added AnalyzerMode/blinking Flag to Init
- CAN Dialog: added VBat enable
- CAN Dialog: added BusTermination enable

Advanced:

- UART Monitor
- allow 2 Framegrabber windows

Bugfix:

- Lvds_Capture in Script and dll crashed
- Framegrabber BugFix save image as .dat also for 30 bit

Release 1.789

New features:

- DragonSuite Plugin Support V1
- FramegrabberDialog added auto configure LVDS channel box
- Dragon Suite Plugin Template V1

Advanced:

- SPI MonitorDialog -> added SSIdleTime to settings to Gui

Script:

- ScriptInterface G.GetFeatures("00000000-00000000-00000000-00000000") to decode the HW Featurecode
- added check LED, use for led recognition at an captured image
- added: G.Lvds_Gen_MipiDsi_Loop_Config_Set(u32_t Flags /* int Enable 0x1 ;ScanEnable 0x2*/, VcId, Dt, SyncMode,HResInPixels, VResInPixels, LineTimeInNs, HSyncWidthInNs, HBackPorchInNs, VFrontPorchInLines, VSyncWidthInLines, VBackPorchInLines, StartWaitTimeInNs)
G.Lvds_Gen_MipiDsi_Loop_Config_Get()
G.Lvds_Gen_MipiDsi_Loop_StartStop(true); //start
G.Lvds_Gen_MipiDsi_Loop_ScanData_Get();
G.Lvds_Gen_MipiDsi_Loop_ScanData_Reset();
G.Lvds_Gen_MipiDsi_Loop_StartStop(false) //stop
G.Lvds_Gen_MipiDsi_Loop_Reset();

Bugfix:

- adjustment readGMSL2 states for MAX96714
- LVDS Info adjustments for different channel configurations
- Bugfix channelsettings grabber config

Release 1.771

New features:

- LVDSInfo: Now checks Datamode before reading SerDes status register
- Pattgen removed auto initialisation
- Pattgen added status red for not initialized
- Pattgen add LineCmdEnable
- added -GlobalI2CDelay set/get to console

Advanced:

- SPI Monitor -> Monitor two channels in parallel (one log)
- Grabber -> new colorformat-pixelmode combo: PIXEL_MODE__RAW8 as grey, RAW12_MIPI_CSI2 as 8 bit grey, RAW16_MIPI_CSI2 as 8 bit grey
- Grabber -> new colorformat-pixelmode combo: PIXEL_MODE__RAW16 as 24bit BG0

Script:

- added bool Lvds_ShowLastCapturedImage();
- added Lvds_ShowLastComparedImage();
- added Lvds_SaveLastCapturedImage(QString path);
- added Lvds_SaveLastComparedImage(QString path);
- added -GlobalI2CDelay set/get to script
- AdvancedDll:*
- Adv_LVDS_SaveLastCapturedImage(u32_t instance, const char * const filePath); //extension bmp,png,jpg or pgm
- Adv_Lvds_Capture(u32_t instance, u16_t width, u16_t height, u8_t colorFormat); // optional with color conversion
- Adv_Lvds_CompareCapturedImage(u32_t instance, u16_t compareArea, // [x,y,width,height], u32_t *nmbOffDifferentPixels);

Bugfix:

- PattGen MinValue height.

Release 1.758

New features:

- added "wait:" option for each line to communication dialog in UART section
- added writeUartFromFile to CommandLine (LVDS and IO)
- Pattgen Dialog-> increased max for width and height to 21000

Advanced:

- added "G.Lvds_FrameGrabber_SyncInfo()" to scriptinterface ; Returns a QStringList of sync values for different kinds of ifTypes
- added writeUartFromFile to Script
- GrabberDialog -> Compare Support for VD2 (without color tolerance)
- AdvancedDll: * added loadReferenceFromFile, * captureAndCompare * SaveLastComparedImage

Bugfix:

- bugfix Capture Area Settings VD1
- bugfix DRS over config by struct

Release 1.752

New features:

- GrabberDialog -> Support for RGB101010
- added DT comboBox for OLDI IfType
- IO Trigger Dialog added status box for MiMfPs
- IO Triggert Dialog -> SerDes GPIO 0-4 Get values

Advanced:

- added PIN_CONFIG__FLAG__SAVE_PIN to MimfpConfig Config Dialog
- Script: updatet Can_SendSingleCANMessage for CAN-FD
- added Lvds_Common_DSI_Loop_Config_lowlevel();
- added Lvds_Common_DSI_Loop_StartStop_lowlevel to script
- added ReSync for DSI Loop

Bugfix:

- Lvds_Get/Set_BUS_WIDTH_SELECTPropValue

Release 1.735

New features:

- added apply DRS Button to config Dialog
- enabled pattgen/generator switch in framegeneratordialog for VD2
- ResetDevice -> Do not switch to Bootloader if USB connection ! (interim until new USB Bootloader)
- Grabber Settings channel config for up to 4 lvds channels

Advanced:

- added Can_Node_BusTermination_Enable to ScriptInterface
- added Lvds_Apix3_Phy_Reset();
- diag some commands to ScriptInterface

Bugfix:

- Read APIX3 Register Page8

G-API 2.1.8844 required

Release 1.720

New features:

- added channel enable for OLDI grabber
- added Maxim DRS Settings to config dialog and Script
- added UART_RX/TX to TriggerConfig basicCON4121
- added io_uart_fifo read/write to side band dialog
- changed decoding for YUV at RAW16 and YUV8 pixelmode YUYV <->UYVY
- Added some information to LVDS Info for generators

Advanced:

- SerDesGpio + MiMfP widget added new col to activate the saving for this config
- added Lvds_Frame_Generator_If_InterfaceEnable_Set(true)
- added Lvds_Frame_Generator_If_InterfaceEnable_Set(u8_t enable);
- Lvds_Frame_Generator_If_InterfaceEnable_Get to SkriptInterface
- added io_uart_fifo functions to script
- add function to apply PattgenConfg after executing configbyFileAdv

Bugfix:

- Bux fix Display in new window single shot BGR swap
- Bugfix Patterngenerator Display settings
- Bugfix LVDS Info SyncWidthV for VD1 SyncScn always on
- added Lvds_FrameGenerator_OpenLDI_Loop_Set true if muxSource pass through && OLDI

Release 1,674

New features:

- 947/948, 988 Support for VD2
- added pixelclock to OLDI LVDSInfo
- LVDS Info added SyncScan values for openLDI If Type
- ConfigureByBinary (gconf) //Version 2-> checksum over Serials + I2C sideband

Advanced:

- added get CoreTemp to Scripting
- added YUV422_UYVY + YUV422_10_UYVY to colorformats
- Script Interface MII Erweiterung (UDP. TxRx Fifo)
- added convertRawVideoToAvi to Scripinterface and Grabberdialog
- added autoinc filename to raw data recording
- added AShell read/write to communication dialog

Bugfix:

- removed Sleep(0) for I2C writing
- bgr swap in external grabber window
- Settingsdialog Bugfix load MiMfPConfig from File
- Settings Dialog Fullscreen fixed

Release 1.629

New features:

- SettingsDialog -> SerDes -> Import für APIX3 Config aus Inova Tool + Speicherung in Config.xml
- SettingsDialog -> Framegeneratorquelle -> pass Through -> Generator_If_RemoveBlanking auto aktiviert
- SettingsDialog -> APIX Mode Umschaltung für VD2
- APIX Mode in LVDS Info
- speichern/laden von MII-mux Settings in Konfigfile
- Flash Firmware über Dragon Suite (im Device Baum)
- IO Trigger Settings starts counting at ZERO !

Advanced:

- added r/w Indigo Tab in Seitenband Dialog

- Skriptbefehl (get/set) für Generator_If_HsBurstEnable + Generator_If_RemoveBlanking

Bugfixes:

- recording in DMA Mode
- Erzeugung Skriptkommando für Mi_Mfp_Config

Release 1.611

New features:

- LvdMuxSourceSwitch in FramegeneratorSettings und Script.
- LvdMuxSourceSwitch Settings im ConfigFile
- DSI SyncInfo in LVDS Info

Advanced:

- Framegrabber -> Avi recording für YUV 10 bit, Grey 12 bit
- ScriptInterface -> CSI/DSI InterfaceScan
- ScriptInterface öffnen von 2 Instanzen möglich

Bugfixes:

- Avi recording

Release 1.601

New features:

- Framegrabbersettings -> weitere CSI Datentypen einstellbar (raw6,7,14)
- Framegrabbersettings -> Handling für DSI Dt's
- Communication Dialog vergrößerbar

Advanced:

- Framegrabber -> Bayer RG support

Bugfixes:

- Bugfix GetCapturearea
- ignore wheelEvent für comboBox in mfp+serdesGPIO Dialog

Release 1.596

New features:

- unterstützt neue Pixelmodes RAW10_MIPI_CSI2 und YUV422_10_MIPI_CSI2
- FramegrabberDialog: decoding Bayerpattern im Pixelmode RAW10_MIPI_CSI2
- FramegrabberDialog: decoding YUV422 im Pixelmode 24 bit und YUV422_10_MIPI_CSI2
- Mainwindow: Untermenüpunkt zum Abschalten der Fehlerausgabe
- Skript und Kommandozeile: neuer CustomType zum Auslesen der Teilenummer der SVC220

G-API 1.4.8236 recommended und enthalten.

Advanced:

- FramegrabberDialog: neuer Farbformatyp YUV 10bit -> angezeigt wird aber immer RGB888 basierend auf 8 bit Decoding.
- unterstützt für Pixelmode 24 bit und YUV422_10_MIPI_CSI2

Bugfixes:

- unknown iftype im Devicescan

- Mainwindow: Fehlermeldungsanzeige

Release 1.587

New features:

- Patterngeneratordialog in Framegenerator Dialog
- added LVDS Info menuitem to Interfacelist menu for LVDS interfaces
- Side band Dialog -> Read I2C registers allow to read up to 0xFFFF number of registers instead of 0xFF
- G-API 1.4.7787 recommended

Advanced:

- Patterngeneratordialog advanced load/save reference template
- Scriptng: Adv_Script_EnableMessageLogging & Adv_Script_SaveLoggedMessagesToFile
- Added some sample scripts

New Advanced-dll * added Scriptexecution

Bugfixes:

Release 1.565

New features:

- Goepel splashScreen
- LVDSInfo new features for MAX9295-96
- LVDSInfo: APIX3 link state (by lowlevel FPGA read)
- save/load SerDesGpioConfig/Mi-Mfp-Config to/from config file
- Mii multiplexer config in settings dialog
- PXI6222/GCAR6222 support
- G-API 1.4.7760 recommended

Advanced:

- Script: pattern generator config
- Script: Apix3 config register read/write
- Script: Lowlevel SerDes_Reset command for Scripting
- Script: Mii multiplexer commands
- Script: added subscript execution in main thread
- Script: added local paths for config and subscripts, relative to scriptpath

Bugfixes:

- some adaptations for Linux
- IO Dialog crash in not advanced mode

Release 1.539

New features:

- Io-trigger-dialog -> get/set dio + software in/out
- Settingsdialog -> configuration history on LoadButton
- save/load Io-Trigger configuration to/from config-xml-file
- Change OpenCV to 348
- added digital IO's to IO Dialog (get/set digital io & relais)

Advanced:

- context help for script dialog
- unlock ADV by hardware lock code
- 30 day's demo license possible
- SPI monitor
- CAN monitor

Bugfixes:

- (Advanced) Raw Data Recording -> Rawfile Header changed to 64 bit aligned
- Framgrabbersettings -> read SerDes Vector
- suppress some Errors with latest Firmware and unsupported properties

Release 1.495

New features:

- FrameGrabberDialog: Umstellung DMA capturing direkt
- Bugfixes LVDS Info
- Bugfix YUV->RGB in RAW16
- CustomReadSerialNumber für Skript und Konsole zu auslesen der Seriennummer der SVC220 Kamera (offiziell nicht dokumentiert)

Advanced:

- Skript Dialog permanent in GUI, zur Ausführung Freischaltung erforderlich
- FrameGrabberDialog, capturing & replay von Rohdaten von 2 LVDS Kanälen (gleiches Interface)

Release 1.485

New features:

- Capturen über DMA möglich, stabil bis 4K@34fps Videos getestet
- LVDS Info Erweiterung für MIPI-CSI2 Grabber
- Settingsdialog: Applyknopf für Seitenbandsettings
- DS90UB940 Erkennung
- CaptureToFile für Kommandozeile inkl. Farbkonvertierung
- bugfixes

Advanced:

Release 1.459

New features:

- 32 bit und 64 bit Version verfügbar
- Upgrade auf QT 5.12.3 und OPENCV 4.1 (64 bit)
- Speicherung der Channel Informationen und "UART passthrough switch" im XML File
- bugfixes

Advanced:

- raw Dekodierung eines Graubildes + Videoaufnahme
- Lvsd_Common_Data_Register_Write für Skriptmodul

Release 1.440

New features:

- IO Dialog for Trigger and SerDes GPIO Configuration
- CAN Dialog for CAN-UART configuration
- Lvds_FrameGrabber_CaptureToFile -> changed mode to MODE__BMP__BOTTOM_UP
- startCapturing command added to commandline

Release 1.406

New features:

- LVDS channel settings (MIPI CSI2)
- side band pass through switch for MAX9276
- save single shot image as *.dat (raw data)

3 Documentation History

Date	Editor	Rev.	Comment
2022-01-18	E.Richter	2.0.0	Document created in HelpNDoc
2022-01-20	E.Richter	2.0.1	Changed image sizes
2022-01-21	E.Richter	2.0.2	Changed image sizes Added Change Log
2022-01-31	E.Richter	2.0.3	Added Sequence Interface Added LVDS Channels for Frame Generator
2022-02-03	E.Richter	2.0.4	Layout changes Added MiMfp Config Tab
2022-03-03	E.Richter	2.0.5	Added Open LDI Mode Changed Frame Generator "Files on Device" to "only for basicCON 4121" Added chapter SerDes GPIO in IO Trigger
2022-04-26	E.Richter	2.0.6	Added Release Notes in Change Log Added chapter Additional Features
2022-12-05	E. Richter	2.0.7	Added some changes for Sequence Interface
2023-02-15	E. Richter	2.0.8	Added info box for 16 bit register in chapter I ² C (chapter 11.9.2) Added maximum bytes for transmitting/ receiving I ² C (chapter 11.2.1.2) Added RxTx Loop for Gen/Grb LVDS Channels Added Color Format to Frame Generator Dialog Window: Display Direct Added Transfer Mode to Generator LVDS Info Added Tray mode for TCP Automation Added Start / Stop buttons to Generator Settings Added import APIX & GMSL register import to SerDes Config Added Mask and Command to I ² C write table; added Group Mode to I ² C command
2023-04-26	E. Richter	2.0.9	Added I ² C Slave configuration Added the Media Interface Boards, for which the Sideband Pass Through mode is now available.
2023-06-01	E. Richter	2.0.10	Added I ² C transfer command
2023-10-09	E. Richter	2.0.11	Removed chapter DMA Configuration
2024-02-26	E. Richter	2.0.12	Added missing line in header structure in chapter Raw Data Recording Added new Pixel Modes Documentary changes in Compare Settings for Frame Grabber Dialow Window

4 Introduction

The *Dragon Suite* software provides a complete tool to configure the *Video Dragon* .




The *Dragon Suite* software consists of the following features:

- Configure the *Frame Grabber* and capture images and videos.
- Configure the *Frame Generator* and generate images and videos.
- *Sideband* communication, to communicate with devices, connected to the *LVDS* network.
- *File System* functions to use the *Video Dragon* file system.
- IO functions to use the *Video Dragon* IO interface and IO trigger matrix.
- CAN functions to use the *Video Dragon* CAN interface.
- The ability to use the **Command Line** Interface or write whole **execution scripts** to simplify the handling.

This documentation gives a short overview of the *Video Dragon* features. For hardware operation, please refer to the respective user manual.

4.1 Symbols

This guide highlights some important comments as follows:

Symbol	Description
	Warning that indicates risk situations and dangers. Disregard can lead to life-threatening situations or destruction of components.
	Information that indicates certain aspects or is important for a particular topic or goal.
	Tip that gives useful hints or recommendations.

4.2 Liability and Warranty Exclusion

This software is designed to simplify the use of our API in conjunction with our Video Dragon hardware. We do not guarantee stability, security and usability, especially when used in manufacturing processes (e.g. end-of-line). In no event shall *GÖPEL electronic* be responsible for any direct, indirect, incidental, special, exemplary, or consequential damages (including but not limited to the purchase of replacement goods or services, loss of use, loss of data or profit, breakdowns, injury, or potential death) in any way in the case of improper use of the *Dragon Suite*.

5 Installation

5.1 Supported Hardware

Supported hardware devices from *GÖPEL electronic* are:

- basicCON 4121
- Video Dragon 6222

5.2 Prerequisites

5.2.1 System Requirements

Dragon Suite is a software for Microsoft Windows operation systems. Your system must comply with the following requirements:

- CPU with at least 4 cores, 8 cores recommended at 2,9 GHz
- Windows 7 or later (**only 64 bit**)
- At least 200MB of free disk space
- At least 8GB RAM
- Installed *G-API* version 1.4.6325 or higher (see *G-API Manual* for reference)

5.2.2 Hardware Installation

For hardware installation please follow the steps in the hardware manual of the corresponding LVDS device.

5.3 Software Installation

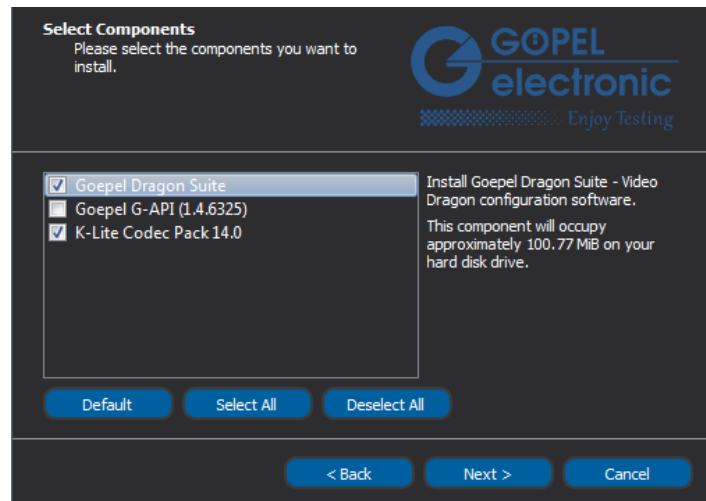
The *Dragon Suite* comes with a setup wizard that guides you through the installation procedure. Make sure that your system meets the [system requirements](#).

Download the Windows installer and start the execution file. Validate the integrity of the file if necessary and run the installer by following the instructions in the installation program.



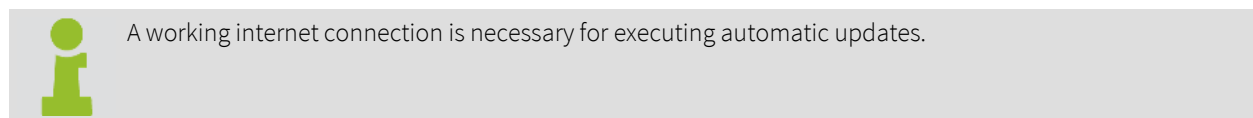
While installation you need to select the components you want to install.


- *Dragon Suite* always needs to be selected.
- It is indispensable having installed the G-API for using the *Dragon Suite*. If it is not installed yet, select this component.
- Select the **K-Lite Codec Pack** if playing videos in the *Dragon Suite* is desired.



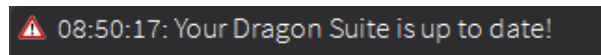
5.4 Update Manager

The software can independently search for updates online at the *GÖPEL electronic* service homepage <https://genesis.goepel.com>.

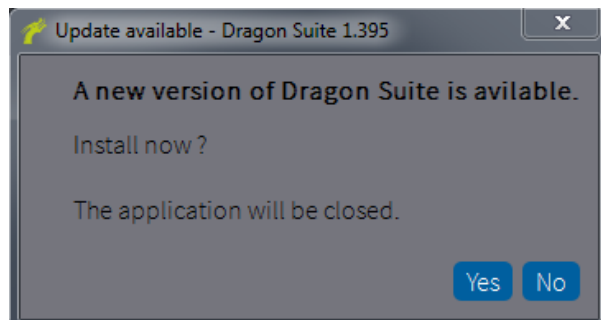


To check for updates, use the [Help](#) menu in the software's menu bar. Use the option **Update** () and the software checks if an update is available.

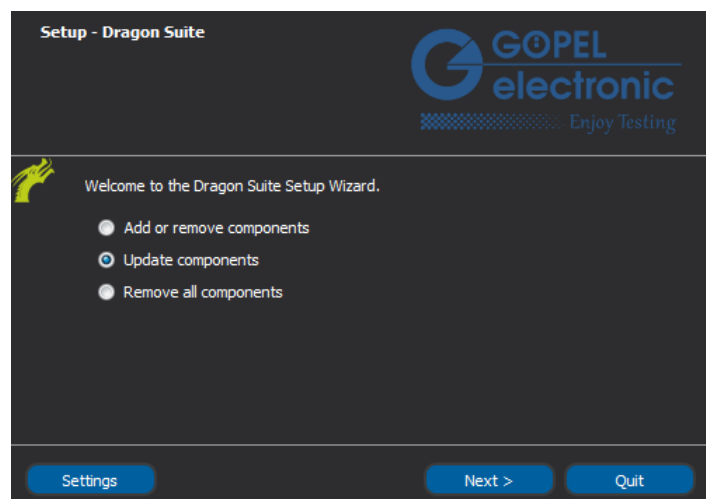
In case no update is available, a message appears in the [Message Box](#).



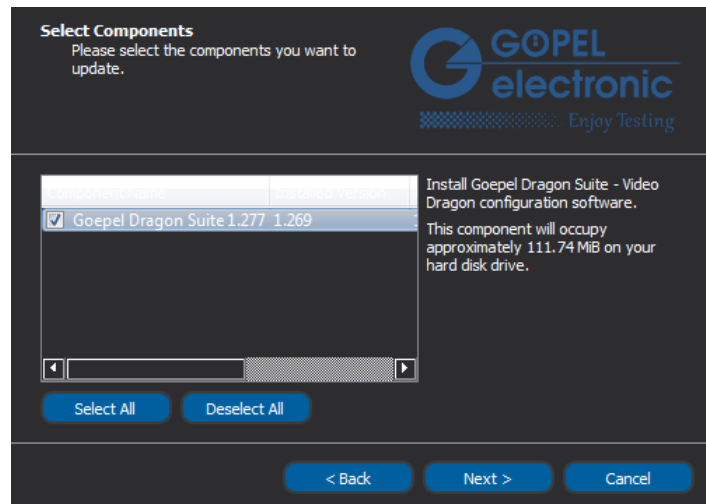
When a new *Dragon Suite* version is available, a small window appears asking if you want to update now or not. Click **Yes** to close the application and start the update. Select **No** if you do not want to close the application, and then run the update later.



When you start the installation, a window opens with options for adding or removing components and for updating components. Select the option **update components** and click the **Next**-button.

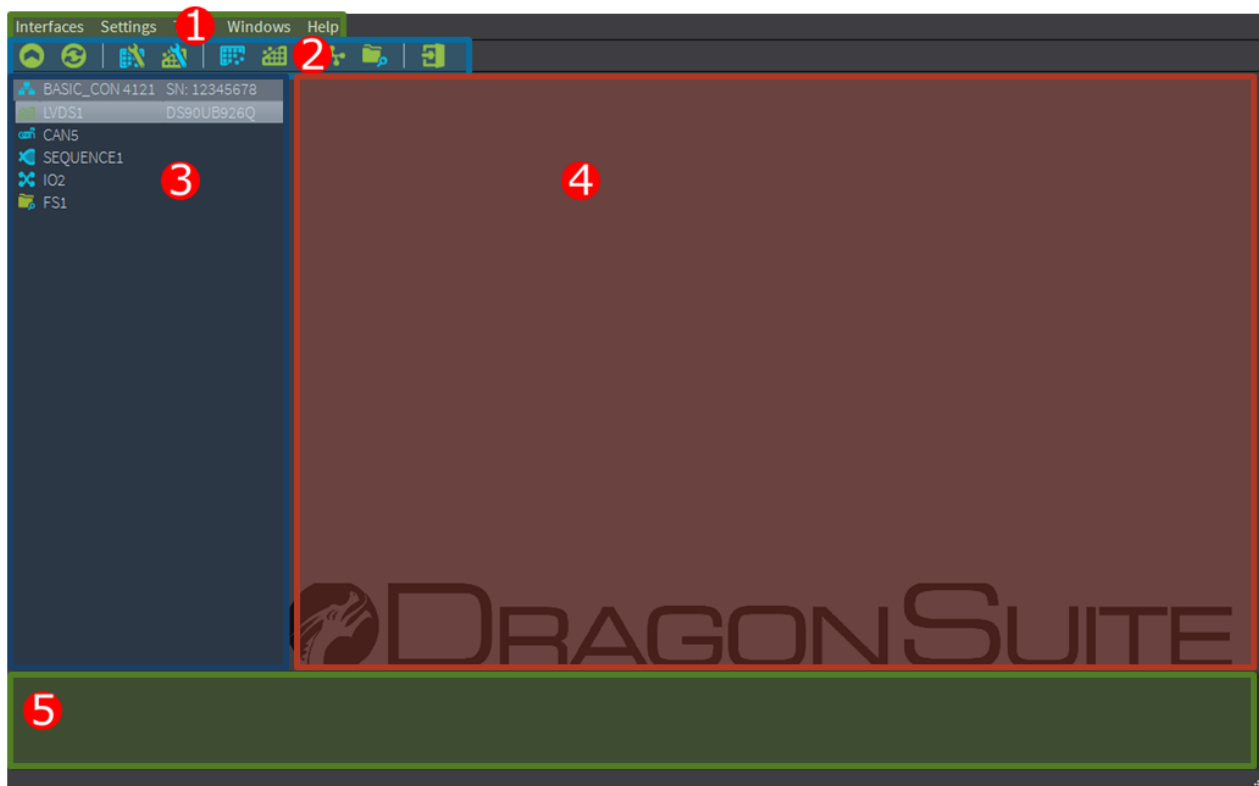


In case an update is available, a checklist will be displayed to choose the components that could be updated. Select all components to be updated.
Click the button **Next** to execute the update. After the update, click **Finish** to exit the wizard.



6 Using the GUI

After starting the *Dragon Suite* the main window appears. The main window remains open during the entire runtime of the software.


















- ① The [Menu Bar](#)
- ② The [Control Bar](#)
- ③ The [Interface Tree](#)
- ④ The [Main Frame](#)
- ⑤ The [Message Box](#)

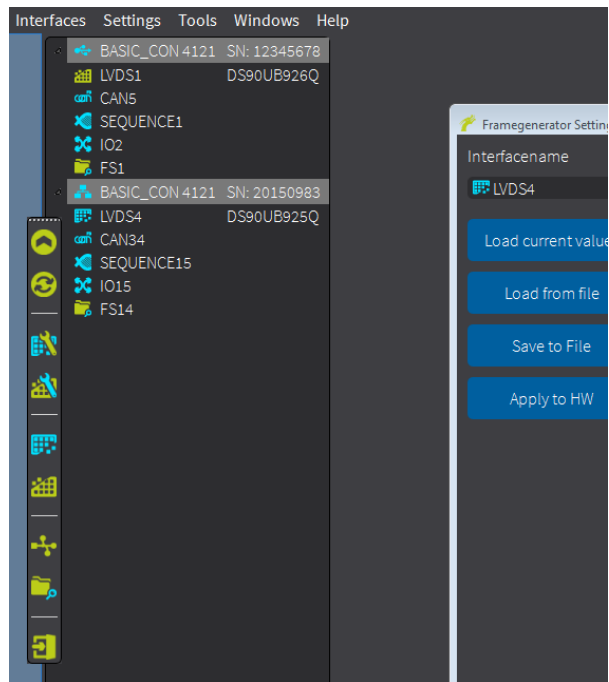
6.1 Menu Bar

Interfaces	In this menu, you can update the interface list displayed in the Interface Tree . With Exit the software will be closed.
Settings	The Settings menu opens the Frame Generator or Frame Grabber settings window where various hard- and software settings can be made.
Tools	Use this menu to open the Frame Generator or Frame Grabber dialog window, the Sideband communication window, the File System manager window, the IO dialog , the CAN dialog , the Sequence Interface or the Monitor Interface .
Windows	Hide or show the Interface Tree and the Message Box in this menu. Additionally use this menu to switch between already opened <i>Frame Generator</i> or <i>Frame Grabber</i> windows in the Main Frame .
Help	In the Help menu, use the option Help (F1) to open the <i>Dragon Suite</i> help window. The help window contains the <i>Dragon Suite</i> manual with a simplified search function. There you can search for Contents or by keywords in the Search tab. The option About (F12) opens a dialog box with information about the <i>Dragon Suite</i> and how to contact <i>GOEPEL electronics</i> for support. Use the Update option to check if an update is available.

6.2 Toolbar

Icon	Description
	Hide the Interface Tree (Ctrl + H).
	Show the Interface Tree (Ctrl + H).
	Refresh the Interface List shown in the Interface Tree (Ctrl + R).
	Open the Frame Generator settings window (F2).
	Open the Frame Grabber settings window (F3).
	Open the Frame Generator dialog window (F4).
	Open the Frame Grabber dialog window (F5).
	Open the Sideband communication window (F6).
	Open the File System manager window (F7).
	Open the IO dialog window (F8).
	Open the CAN dialog window.
	Open the Sequence dialog window.
	Open the Script Interface window.
	Open the Monitor dialog window.
	Close the <i>Dragon Suite</i> (Ctrl + Q).

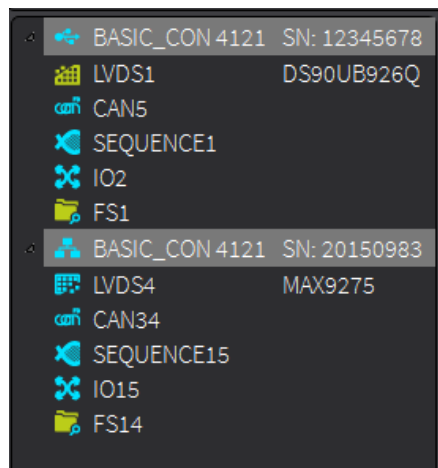
Arrange the Toolbar elsewhere in the Main Window by left-clicking the white dotted line and dragging it to the desired position.



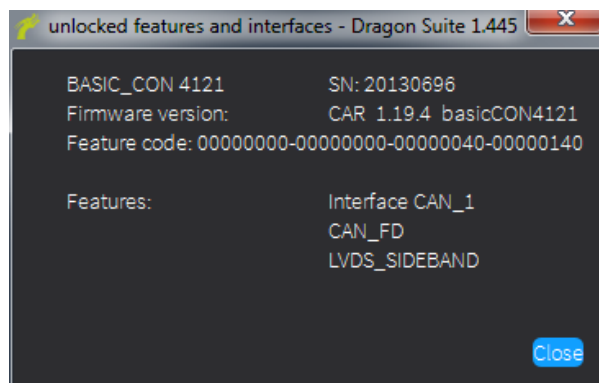
6.3 Interface Tree

In the Interface Tree all available interfaces of the *GÖPEL electronic* devices are represented. Although you can see all existing interfaces and devices, not all are supported in the *Dragon Suite*. The interface names are assigned through the **Hardware Explorer** installed with the *G-API*. If a connected device does not appear in the Interface Tree, first update the **Hardware Explorer**. If this does not help, check the **Hardware Explorer** preferences.

To hide the Interface Tree use the [Menu Bar](#) or the [Toolbar](#). Single devices can be hidden by clicking on the small triangle symbol on the left side of the device name.



Right-clicking on the LVDS device opens the submenu with options for resetting the entire device or displaying the **unlocked features** of the device.

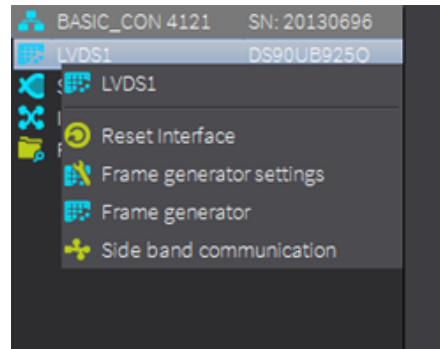


Additionally, by right-clicking on the device and selecting **Show IP Features**, the details of the device and its interfaces can be viewed.

Device	
Device Type	GCAR6222 - Video Dragon II
Firmware Version	CAR_1.22.3 GCAR6222_VAR3
Hardware Version	1.01
Serial Number	20221103
VHDL version	1.36

Interface #1	
Type: RX	
Version: 1.00	
Lane CFG: 0	
MIPI: OpenLDI	
DPHY Type: Master	
Lane Rate: 76.85MHz	

Right-clicking on the LVDS interface opens a small submenu and offers four different options. The first one is to reset the interface. Also you can open the settings window or dialog window for either *Frame Generator* or *Frame Grabber*, depending on the device. Additionally there is the possibility to open the *Sideband* communication window. Any other interface can be reset by right-clicking on the interface in the Interface Tree.



6.4 Main Frame

The settings and dialog windows are displayed in the Main Frame. This windows can be arranged arbitrarily. To work with multiple windows, enlarge the Main Frame by left-clicking on the right edge of the Main Window and dragging it to the right.

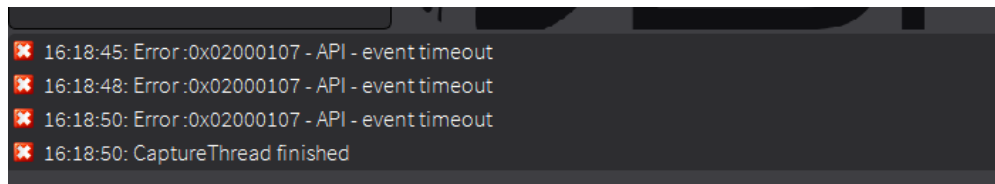


The windows can be minimized in the lower left corner of the Main Frame or maximized to fill the entire frame.

6.5 Message Box

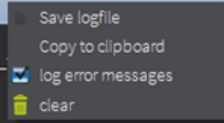
The Message Box is used to display the **log file**, for example capturing information or [error messages](#). If you right-click in the message box, you can save or delete the log file.

Hide the Message Box using the [Menu Bar](#).



By right clicking in the message box a small menu selection appears. The messages can be saved in a log file or copied to the clipboard. Furthermore the logging of error messages can be switched off (uncheck the box). With **Clear** all entries are deleted.

```
port not initialized
helper.cpp function:u32_t QG_api_helper::Lvds_PattGen_Property_GetById(u8_t, u32_t)
port not initialized
helper.cpp function:bool QG_api_helper::Lvds_PattGen_Pattern1Def_Get(u8_t, u8_t*, u8_t*, u16_t)
port not initialized
helper.cpp function:u32_t QG_api_helper::Lvds_PattGen_Property_GetById(u8_t, u32_t)
port not initialized
helper.cpp function:QStringList QG_api_helper::Lvds_PattGen_Pattern1Def_Get(u8_t)
port not initialized
helper.cpp function:bool QG_api_helper::Lvds_PattGen_Init(u8_t, u8_t)
```



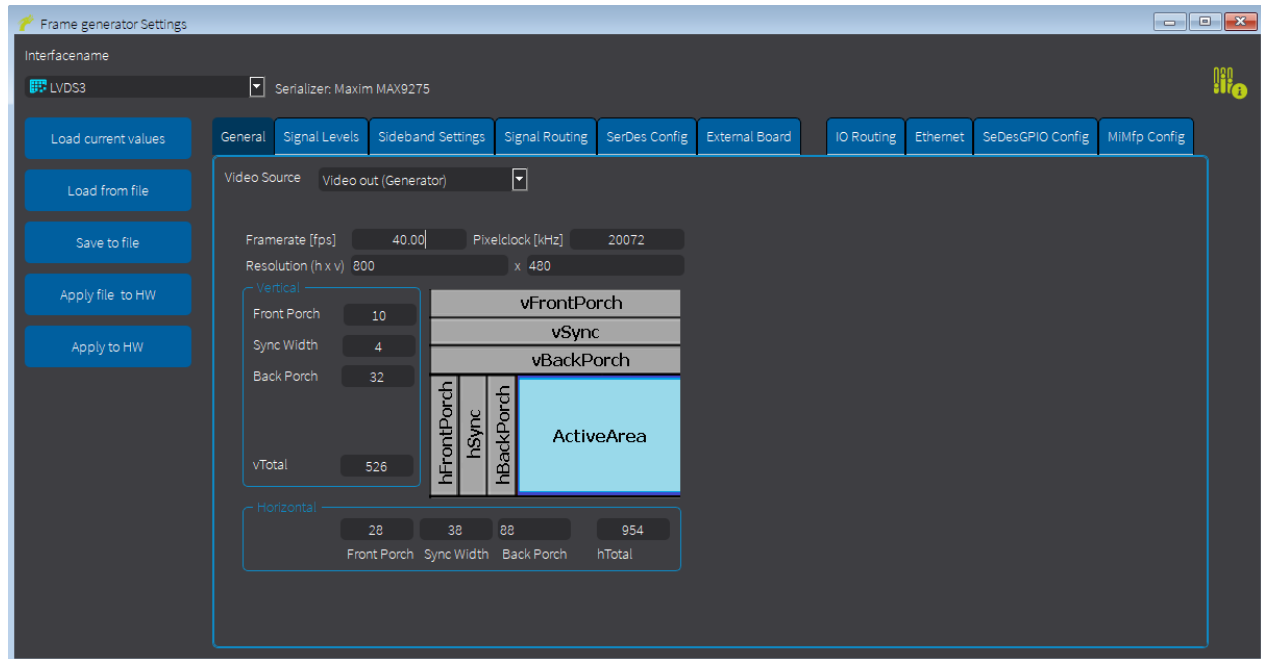
An overview of common mistakes can be found in chapter [Common Error Messages](#).

7 Setting up the Frame Generator

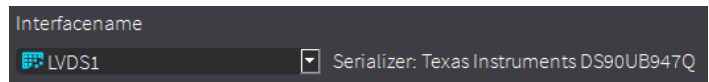
Open the *Frame Generator* settings window by using one of the alternatives illustrated in chapter [Using the GUI](#). The settings window shows all supported device settings. Most of the functionalities of the *Dragon Suite* depend on valid settings of the interface. Setting up the device should therefore be the first step after starting the software.



Generally parallel usage of several interfaces with the *G-API* is possible. But the *Dragon Suite* supports the usage of only one *Frame Generator* interface at a time.



All available *Frame Generator* interfaces are listed in the drop down menu. The currently selected interface is shown in the text field of the drop down menu.

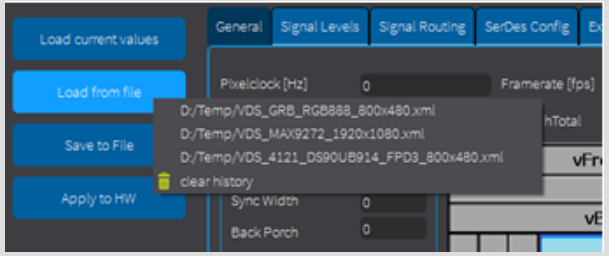


On the left of the settings window are the following buttons:

Button	Description
Load current values	Load the current setting values of the selected interface.
Load from file	Load the values for the selected interface by importing an external XML file. Importing a settings file does not overwrites the current settings on the device. To overwrite the device settings, use the Apply to HW button.
Save to File	Save the current settings of the selected interface by exporting them to an external XML file.
Apply file to HW	Overwrite the current settings directly from the configuration file.
Apply to HW	Overwrite all current settings on the device with the settings displayed on the tabs of the window.

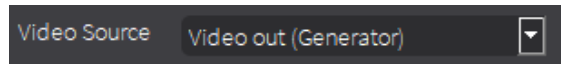
TIP

A right click on the **Load from file** button opens a selection of the last opened files. This facilitates the search for frequently used configurations. The history can be cleared with **clear history**.

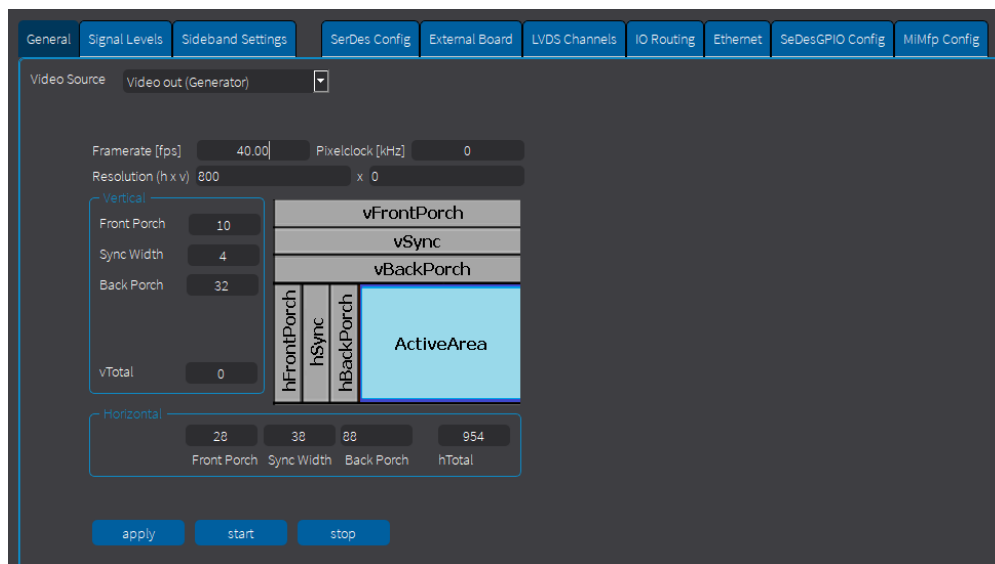


7.1 General

Depending on the device used, a stored image can be generated or the hardware itself generates an individual pattern. Use the dropdown menu to select the video source.



The general serializer settings for **Video Source = Video Out** are pixel clock, frame rate and image format parameters. Image format parameters are described by the horizontal and vertical synchronization signals, indicated in clocks. Horizontal synchronization means that the end of a single line is reached, whereas vertical synchronization indicates the end of a complete frame.



Pixel clock	Determines, how many pixels need to be sent per second.
Frame rate	Sets the maximum rate for capturing frames in continuous mode.
Active area	Size of desired file.
Front Porch	Describes the duration between the frame end information and the signal pulse.
Sync Width	Signal pulse. The vertical Sync Width unit is lines whereas the horizontal Sync Width unit is pixels.
Back Porch	Interval between Sync Width and the beginning of frame information.

There is a dependency between pixel clock, frame rate and synchronization signal parameters, as follows:

$$\text{Frame Rate [fps]} = \text{Pixel Clock [Hz]} * 1000 / (\text{hTotal} * \text{vTotal})$$

When changing one of the parameters, frame rate or pixel clock adapt oneself automatically in *Dragon Suite*.

Below the parameters are the following buttons:

Button	Description
Apply	Apply the current settings.
Start	Start generating.
Stop	Stop generating.



Changed values of pixel clock and frame rate are taken directly from the hardware, without having to use the [Apply to HW](#) button.

Select **Video Source = Video in** to switch to Passthrough mode. The frame and sideband data from the input interface are passed through to the output.

For **Video Source = Pattern Generator Out** please go to [Pattern Generator](#) chapter.

7.2 Signal Levels

The valid signal levels and edges are defined in the Signal Levels tab.



This parameters are only necessary for *basicCON 4121*.

Polarity	Defines which signal level indicates vertical or horizontal synchronization.
Data Enable	Specifies at which level pixel data is being transmitted.
Pixel Clock Polarity	Describes at which edge of the pixel clock signal (rising or falling) the values of the signals are to be sampled.
Lock Output Enable	When this parameter is activated, LED 4 lights up for a successful lock.
Lock Polarity	Determines whether the signal is high or low when a lock is detected.

7.3 Signal Routing



This functionality is only available for *basicCON 4121*.

Since there is no common standard for mapping the video signals to the 32 serialized bits in the LVDS video stream, this assignment must be defined for each device. This can be done in the Signal Routing tab.

The screenshot shows the 'Signal Routing' tab with four columns of bit mappings. Each bit is represented by a number and a dropdown menu.

Video bit 0-7	Video bit 8-15	Video bit 16-23	Video bit 24-31
0 NOT USED	8 RED 3	16 GREEN 3	24 BLUE 3
1 NOT USED	9 RED 4	17 GREEN 4	25 BLUE 4
2 HSYNC	10 RED 5	18 GREEN 5	26 BLUE 5
3 VSYNC	11 RED 6	19 GREEN 6	27 BLUE 6
4 DATA ENABLE	12 RED 7	20 GREEN 7	28 BLUE 7
5 RED 0	13 GREEN 0	21 BLUE 0	29 NOT USED
6 RED 1	14 GREEN 1	22 BLUE 1	30 NOT USED
7 RED 2	15 GREEN 2	23 BLUE 2	31 NOT USED

On the Signal Routing tab, you can specify the mapping of each serialized bit of the LVDS stream. Therefore the combo boxes of the tab provide the option to set the color bits. Since the resulting RGB frame has a color depth of 24 bits, each color (red, green and blue) has a maximum depth of 8 bits. The selection of the corresponding bit in the video stream is made by selecting the correct value from the possibilities given by the combo box. All 32 bits of the stream and the value **not used** can be selected. **Not used** means that the signal has no correspondent in the video stream. This should be done, for example, if a video stream contains frames whose color depth is less than 24. The significance of the color bits is ascending. For example, this means for an 8-bit color value of red, **R0** is the least significant bit, and **R7** is the most significant bit.

The combo boxes are also used to assign the control signals. The vertical sync signal **VSynC** notifies the end of transmission of a complete frame, whereas the horizontal sync signal **HSynC** indicates the end of transmission of a line. The third control signal is the **Data Enable** signal. In a continuous stream, the video signal may contain porches in each single line and between the end and the beginning of a new frame. The data enable signal indicates whether pixel data is currently being transmitted.

7.4 SerDes Config

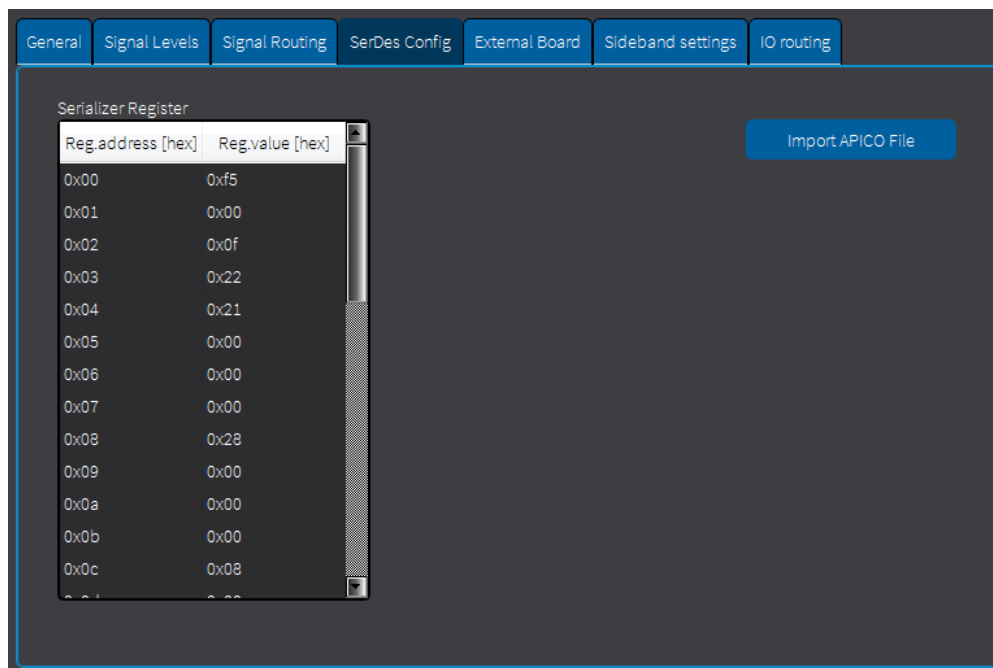


This functionality is only available for all Media Interfaces of *basicCON 4121* and *APIX* Media Interfaces of *Video Dragon 6222*.

Dragon Suite gives the opportunity to configure the serializer / deserializer by manipulating the bus register.

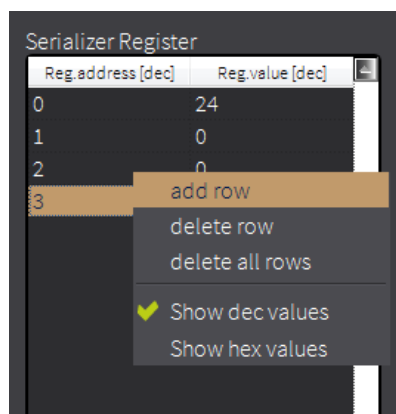


Manual manipulating of the configuration data list needs an extreme good knowledge of the meaning of each register value. This can only be obtained from the data sheets of the serializer / deserializer. Even a single wrong setting of only one register may render the complete configuration invalid and the serializer / deserializer inoperative.



To set up the serializer configuration, you must understand the structure of the configuration data: Each serializer / deserializer stores its configuration in a set of one-byte register values that can be read from or written to the serializer / deserializer. Some serializer / deserializer types group subsets of their registers to different internal devices, others have only one internal device. The value can be presented as a decimal or hexadecimal value, which is selected by right-clicking in the list.

To add a new row to the register list, right-click in the list and select **add row** from the sub menu. A blank row appears at the bottom of the list. To delete a single row, right-click in the relevant row and select **delete row**. It is also possible to delete all rows.



To read a register, double-click an existing register address item and change its value to the requested register address. [Loading the current values](#) updates the register data from the specified address.

The register data can be changed by double-clicking in the corresponding entry and changing the value. The [Apply to HW](#) button immediately overwrites the current value in the serializer / deserializer.

Reg.address [dec]	Reg.value [dec]
0	24
1	0
2	0
3	210
4	21

For Media Interface with APIX2 or APIX3 (*Video Dragon 6222*) the register configuration can be loaded from a *.csv file (generated with *APICO* tool). To do this, right-click in the table and select **Import APIX config**. Right-click and select **Apply Register config** to write the registers without reconfiguring the entire Media Interface.

Page[hex]	Register[hex]	Value[hex]	Comment
0x00	0x8b	0xf8	
0x00	0x8c	0xef	
0x00	0x12	0x0c	
0x02	0x00	0x03	
0x02	0x01	0x30	
0x03	0x0c	0x00	
0x03	0x0d	0x00	
0x00	0x0c	0x00	
0x00	0x0d	0x00	
0x03	0x07	0xf8	
0x03	0x08	0xff	
0x03	0x09	0x00	

For Media Interface with GMSL2 or GMSL3 (*Video Dragon 6222*) the register configuration can be loaded from a *.csv file (see also chapter [I²C](#)). To do this, right-click in the table and select **Import GMSL register config**. Right-click and select **Apply Register config** to write the registers without reconfiguring the entire Media Interface.

Device[hex]	Register[hex]	Value[hex]	Mask[hex]	Comment
0x00	0x2be	0xc3	0xff	GPIO# 0 RX/TX TX ID 0
0x00	0x2bf	0x40	0xff	GPIO# 0 RX/TX TX ID 0
0x00	0x2c0	0xc0	0xff	GPIO# 0 RX/TX TX ID 0
0x00	0x2be	0xc3	0xff	GPIO# 0 RX/TX TX ID 3
0x00	0x2bf	0x43	0xff	GPIO# 0 RX/TX TX ID 3
0x00	0x2c0	0xc0	0xff	GPIO# 0 RX/TX TX ID 3
0x00	0x30	0x48	0xff	GPIO# 0 RX/TX TX ID 3
0x00	0x31	0x88	0xff	GPIO# 0 RX/TX TX ID 3



Devices with an *APIX Media Interface* Board require a special configuration via an Apico file. This file can be loaded via the **Import Apico File** button. The button can only be used with the appropriate Media Interfaces and *basicCON 4121*.

7.5 External Board



This parameters are only necessary for *basicCON 4121*.

Different modes can be set for the various supported serializers. Depending on the currently selected LVDS interface, only the corresponding setting options are available. Further information on the individual setting options can always be found in the data sheet of the respective serializer.

If you set **undefined** in one of the settings, the device configuration is not taken into account.

EDID Loop Through	<p>Extended Display Identification Data (EDID) is a 128-byte data format for displays that describes their capabilities, such as manufacturer, date of manufacturing or display size. This parameter is only supported by ADV7611 (HDMI).</p> <ul style="list-style-type: none"> No loop through (default): the EDID data of serdes extension board are used. Loop through enabled: the EDID data of output device is looped to input.
Maxim bus width select	<p>This parameter is only supported by MAX9260, MAX9272, MAX9276, MAX9259, MAX9271 and MAX9275. It describes the data format of the device.</p> <ul style="list-style-type: none"> 24 bit bus mode (default): the first 21 bits contain video data. 32 bit bus mode: the first 29 bits contain video data. High bandwidth mode (27bit; only supported by MAX9276 and MAX9275): the first 24 bits contain video data or special control signal packets. <p>The last 3 bits always are the embedded audio channel bit, the forward control channel bit and the parity bit of the serial word.</p>
Maxim mode select	<p>This parameter is only supported by MAX9260, MAX9272, MAX9276, MAX9259, MAX9271 and MAX9275. Two modes of control-channel operation are available for this device:</p> <ul style="list-style-type: none"> Base mode (default): use either I²C (half-duplex) or GMSL UART protocol (full-duplex). Bypass bus mode: use a custom UART protocol.

Maxim DRS select	<p>This parameter is only supported by MAX9276 and MAX9275. There are two modes of operation:</p> <ul style="list-style-type: none"> • DRS = low rate • DRS = high rate (default)
APIX mode	<p>This parameter is only supported by INAP375T and INAP375R. The <i>APIX</i> functionality provides a high-speed Gigabit video link in combination with full-duplex communication over two wire pairs. There are two modes of operation:</p> <ul style="list-style-type: none"> • APIX1 mode • APIX2 mode (default)
TI low frequency mode	<p>This parameter is only supported by DS90UB947 and DS90UB948.</p> <ul style="list-style-type: none"> • Normal mode (default) • Low frequency mode enabled
TI LVDS link data mapping	<p>This parameter is only supported by DS90UB947 and DS90UB948. The device can be configured to accept 24-bit color with two different mapping schemes:</p> <ul style="list-style-type: none"> • Mapping mode 0 (default): SPWG mapping. • Mapping mode 1: OpenLDI mapping.
Transceiver power down	<p>This parameter is only supported by MAX9260, MAX9272, MAX9276 and DS90UB914. The devices have a power-down mode which reduces power consumption:</p> <ul style="list-style-type: none"> • Transceiver is powered up (default). • Transceiver is powered down.
Sideband pass through mode	<p>This parameter is supported only depending on the serializer used. The devices pass a sideband command through to further sideband subscribers:</p> <ul style="list-style-type: none"> • disabled (default) • I²C pass through (only for MAX9295/9296, DS90UB947/948 and DS90UB953/954) • UART pass through (only for MAX9276 v1.1 and MAX9295/9296)

7.6 Sideband Settings

Different sideband communication modes can be set for the various supported serializers and deserializers. The *Video Dragon* supports SPI, I²C and UART. Depending on the currently selected LVDS interface, *Dragon Suite* will provide only the appropriate data mode options.

Detailed information about sideband can be found in the chapter [Sideband Communication](#).



Data Modes are only available with activated sideband features.

General | Signal Levels | Signal Routing | SerDes Config | External Board | Sideband settings | IO routing

Data Mode: DATA MODE I2C MASTER TO SERIALIZER

UART | I²C | SPI

I2C master baud rate: 100 kHz

I2C master clockstretching: enabled

I2C master stop no acknowledge: Do not send ACK after last byte

I2C slave baud rate: 100 kHz

apply



The sideband settings can be set independently of all other parameters. Use the **Apply** button in the lower right corner of the Settings window.

The following Data Modes are possible:

- I²C master to deserializer: *Video Dragon* is I²C master at deserializer IC.
- I²C master to serializer: *Video Dragon* is I²C master at serializer IC.
- SPI master to deserializer: *Video Dragon* is SPI master at deserializer IC.
- SPI master to serializer: *Video Dragon* is SPI master at serializer IC.
- SPI passthrough dual: *Video Dragon* is connected to transmitter and receiver in "dual mode".
- SPI receiver dual: *Video Dragon* is connected to receiver in "dual mode".
- UART to deserializer: *Video Dragon* is connected to UART interface of deserializer IC.
- UART to serializer: *Video Dragon* is connected to UART interface of serializer IC.
- SPI transmitter dual: *Video Dragon* is connected to transmitter in "dual mode".
- SPI slave to deserializer: *Video Dragon* is SPI slave at deserializer IC.
- SPI slave to serializer: *Video Dragon* is SPI slave at serializer IC.
- I²C slave to deserializer: *Video Dragon* is I²C slave at deserializer IC.
- I²C slave to serializer: *Video Dragon* is I²C slave at serializer IC.

Depending on the selected Data Mode, the sub-tab for the corresponding sideband is automatically opened.

7.6.1 UART

For UART mode the following parameters can be adapted:

- Baud rate (default value: 115200 baud): Sets the earliest possible value to this parameter.
- Parity (default value: no parity): Sets, if a parity will be build and transferred.

7.6.2 I²C

For I²C mode the following parameters can be adapted:

- I²C master baud rate (default value: 400kHz): Baud rate of I²C master can be 100kHz or 400kHz.
- I²C master clockstretching (default value: enable): Enables or disables the master response on the deferment of the clock, organized by the slave.
- I²C master stop no acknowledge (default value: Do not send ACK after last byte): I²C master sends an acknowledge or not after the last received byte from the slave.
- I²C slave baud rate (default value: 100kHz): Baud rate of I²C slave can be 100kHz or 400kHz.

I²C Slave

It is possible to simulate the response of an I²C slave to a master request. The I²C master first sends a request, for example that it wants to read the data from address 0x21 of the slave. Then the master gives a clock and fetches the response from the slave. See also the sections in chapter "Sideband Communication".

If the Data Mode I²C slave to Serializer or I²C slave to Deserializer is set, another input mask appears below.

The screenshot shows the 'SerDes Config' tab in the Dragon Suite software. The 'Data Mode' dropdown is set to 'DATA MODE I2C SLAVE TO DESERIALIZER'. The 'I2C' sub-tab is selected. The configuration parameters are as follows:

- I2C master baud rate: 100 kHz
- I2C master clockstretching: enabled
- I2C master stop no acknowledge: Do not send ACK after last byte
- I2C slave baud rate: 400 kHz

A table for mapping memory offsets to data is displayed:

mapCode	memory offset	data length	data [csv]
0x20	0x00	0x05	0xca,0xfe,0x00,0xbe,0xef
0x21	0x05	0x06	0xca,0xfe,0x00,0xbe,0xef,0x01
0xabcd	0x0b	0x07	0xca,0xfe,0x00,0xbe,0xef,0x01,0x02

Below the table are buttons for '+ add row', 'delete all rows', and 'Load mapping from device'. An 'apply' button is located at the bottom right of the configuration area.

In this mask the freely selectable **Slave Device Address** must be specified. In addition, the **Register Address Width** must be set to 1 byte or 2 bytes.

If the **Mapping Mode** is not set, a confirmation is sent for each I²C address (MapCode). In this case **Acknowledge all** and the mapping table are not relevant.

If **Mapping Mode** is set, the slave addresses can be defined individually in the table below. The FPGA then only responds to the addresses defined in the mapping (MapCode). If the master requests an address that is not in the MapCode and **Acknowledge all** is not set, "no Acknowledge" is returned.

In the mapping table the **MapCode** must be defined first. This can be for example a register address.

By right-clicking in the table you can add or delete rows.

A maximum of 32 MapCodes may be configured.

Memory Offset defines from which memory location should be read.

In **Data** you define which data (starting from Memory Offset) should be written into the FPGA memory. The length of the data must also be added in the **Data Length** column.



The **Slave Device Address** and the **Register Address width** can only be initialized and reset. It is not yet possible to get these parameters.

However, with the correct Slave Device Address the set mapping table can be read back. This is done by right-clicking in the table and selecting **Load Mapping from Device**.

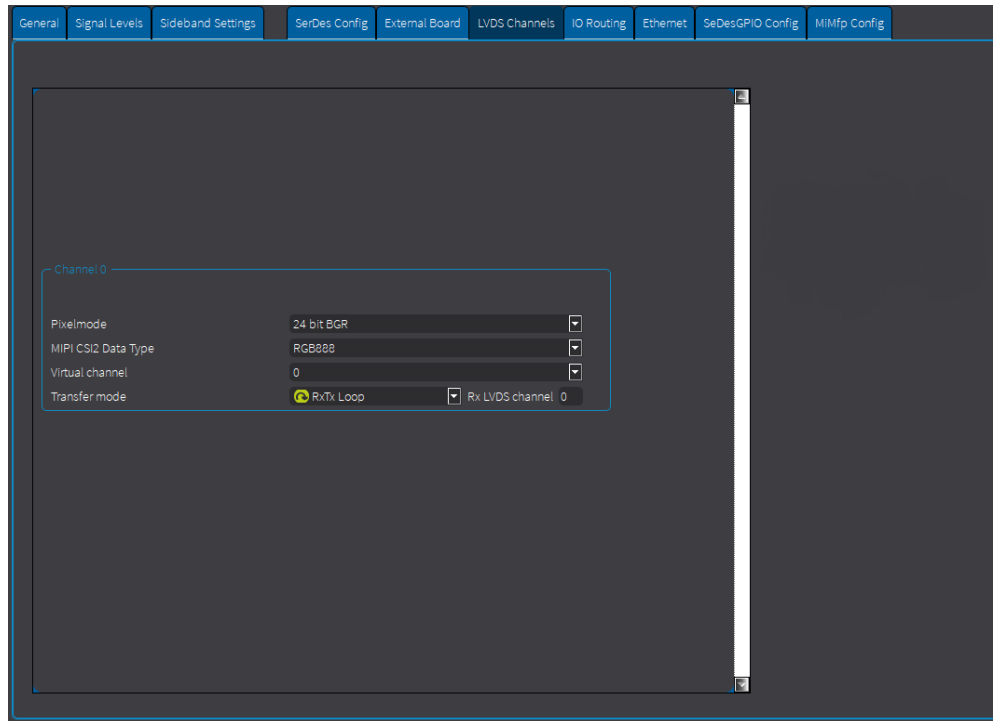
7.6.3 SPI

For SPI mode the following parameters can be adapted:

- SPI master CPHA (default value: data is valid at first clock edge after chip select): SPI clock phase - Valid data at first respectively second clock edge after chip select.
- SPI master CPOL (default value: clock polarity in idle state is low): SPI clock polarity in idle state.
- SPI master CS_mode (default value: chip select remains low (active) between consecutive bytes): SPI chip select behavior between consecutive bytes of the same transfer.
- SPI master clock divider (default value: 0): determines SPI clock frequency based on freq. 25MHz.
 - 0: divider is 1; clock frequency is 25MHz
 - 1: divider is 2; clock frequency is 12.5MHz
 - 2: divider is 3; clock frequency is 8.33MHz
 - ...
 - 255: divider is 256; clock frequency is 0.1953125MHz
- SPI slave CPHA (default value: data is valid at first clock edge after chip select): SPI clock phase - Valid data at first respectively second clock edge after chip select.
- SPI slave CS idle [ns] (default value: 20): SPI chip select idle minimum in 20ns steps.
 - 0: $127 * 20\text{ns} = 2.54\mu\text{s}$ chip select idle after transfer
 - 1: $1 * 20\text{ns} = 20\text{ns}$
 - 2: $2 * 20\text{ns} = 40\text{ns}$
 - ...
 - 255: $255 * 20\text{ns} = 5.1\mu\text{s}$

7.7 LVDS Channels

Use this tab to configure the capture parameters for the individual channels of the *Video Dragon*. Currently the settings for one channel (Channel 0) can be adjusted.



The following properties are only supported for *Video Dragon 6222* with MIPI CSI-2 and OLDI virtual channel extension. Not all Media Interface Boards support all parameters.

From *Video Dragon 6222*, several physical channels (Channel 0, Channel 1, etc.) lead to the PC. The number of virtual channels depends on the *Media Interface* module used. Depending on the configuration, the virtual channels can go through one of the physical channels.

Enable or disable the physical channel by checking the **enable** box. You can select the **Pixel Mode** and the **Data Type** for each physical channel.

For the **MIPI CSI-2 Data Type** the following data formats are possible:

- RGB888
- RAW6
- RAW7
- RAW8
- RAW10
- RAW12
- RAW14
- YUV422 8bit
- YUV422 10 bit
- embedded data (raw data packet, not necessarily just video data)

For the **OLDI Type** the following data formats are possible:

- RGB 18 bit

- RGB 24 bit
- RGB 30 bit

For the **Pixel Mode** the following modes are possible:

Mode	Description
24 bit BGR	24 bit BGR format with 3 bytes per pixel. Per byte one color channel is transmitted. Byte 0: Blue (bit 7..0) Byte 1: Green (bit 7..0) Byte 2: Red (bit 7..0)

From the **Virtual Channel** drop-down menu, you can select the virtual channel to be routed to the appropriate physical channel.

For the **Transfer Mode** the following data formats are possible:

- Generate
- RxTx Loop

Write the displayed settings in the window by clicking the **Apply** button. When you use this button, the settings of the other tabs are not written to the device.

7.7.1 Open LDI Mode



This functionality is only supported for *Video Dragon 6222*.

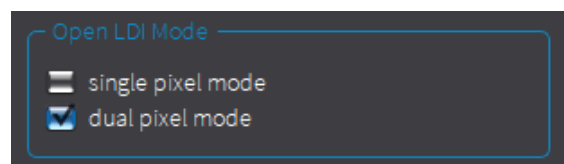
For *Media Interface* boards with Open LDI serializers (e.g. DS90UB947 from TI) and deserializers there is the possibility to work in single or dual pixel mode.

The image to be generated is broken down into individual pixels in the FPGA. From there, the data is sent to the serializer. Since the serializer has two inputs for the data from the FPGA, the pixel stream from the FPGA is routed to two data channels (dual pixel mode). These two data streams are then also routed to two channels at the output of the serializer and accordingly received and processed by the counterpart (deserializer).

If the remote station has only one channel, the serializer must operate in single mode. Otherwise, some of the pixels will be lost and there will be an incorrect image or no lock at all.

To generate in single mode, two steps must be performed:

1. The FPGA must be configured to give the pixels only to one Open LDI input of the serializer. This is done via the configuration and can be set in the Dragon Suite by selecting the Open LDI Mode.



2. The serializer must also be set to single mode. This is done by setting the register via the [I²C communication](#). Which register has to be set depends on the used circuit.

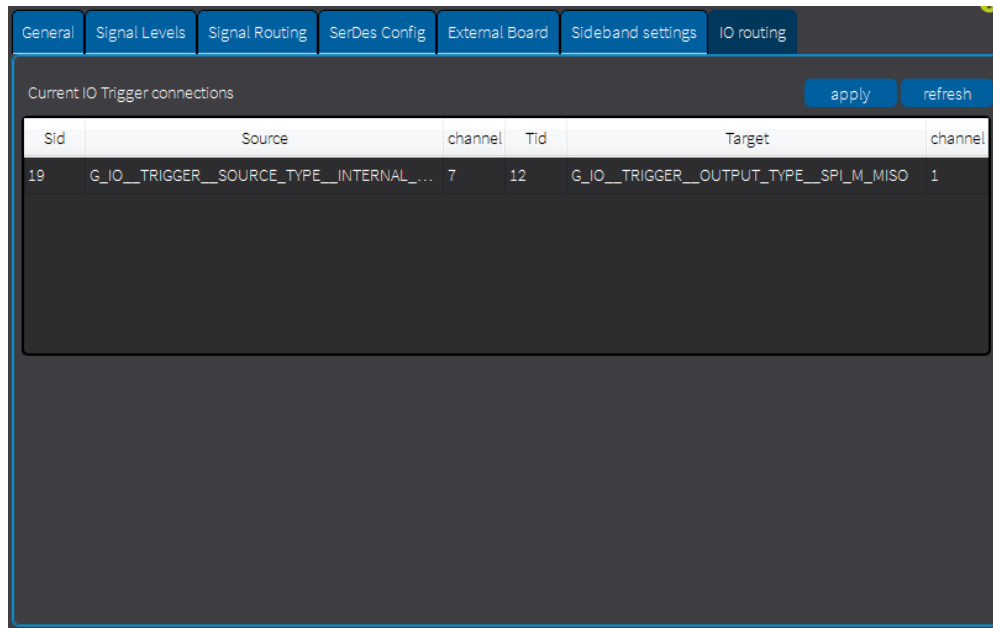
In the DS90UB947 circuit, for example, register 0x4F is responsible for the Open LDI mode.

7.7.2 RxTx Loop

This segment is described in the [Frame Grabber settings](#).

7.8 IO Routing

The *GÖPEL electronic Video Dragon* has an IO trigger function with which certain events can be routed to a trigger event. The routing can be defined in the [IO Trigger Dialog](#) window. The defined connections can also be found in the IO Routing tab of the Settings window for an easier overview.



Sid	Source	channel	Tid	Target	channel
19	G_IO_TRIGGER_SOURCE_TYPE_INTERNAL_...	7	12	G_IO_TRIGGER_OUTPUT_TYPE_SPI_M_MISO	1

The channel entries of the connection table can be changed manually by clicking on the desired field and editing the value. Use the **Apply** button to set the values. Use the **Refresh** button to load the actual settings.

7.9 Ethernet

The Ethernet tab supports the **MII Multiplexer** functionality (Media Independent Interface Multiplexer). With the MII Multiplexer you can establish connections between different communication sources and destinations directly on the device.

Depending on the selected interface the corresponding sources and targets are available. Which ones are available can be seen in the graphic, which is automatically displayed in the tab.



Regarding LVDS devices, this feature is only supported for *Media Interface* boards with *APIX* MII function (Currently these are the boards INAP562T and INAP562R).

General

Signal Levels

Signal Routing

SerDes Config

External Board

Sideband Settings

LVDS Channels

IO Routing

Ethernet

Source

PHY RX

2

Instance(APIX-RX 0)

Ethernet MII Multiplexer

LVDS4

Get

Set

reset own

Target

SWITCH IN 0

1

Instance

PhyRx/Tx

Instance 1

Phy/ G Patch

Tx

Rx

Instance 2

Apix Rx A0

Tx

Rx

Instance 3

Apix Tx A0

Tx

Rx

Instance 4

Apix Tx A1

Tx

Rx

MII Multiplexer

Switch Instance 1

In 0

In 1

In 2

Out 0

Switch Instance 2

In 0

In 1

In 2

Out 0

MAC (firmware)

Rx

Tx

Current connections

apply

refresh

Sid	Source	channel	Tid	Target	channel
5	SWITCH_OUT	1	0	MAC_RX	1
1	MAC_TX	1	1	PHY_TX	1
0	PHY_RX	3	1	PHY_TX	2
0	PHY_RX	2	1	PHY_TX	3
0	PHY_RX	2	5	SWITCH_IN_0	1
0	PHY_RX	3	6	SWITCH_IN_1	1

Using the switches of the MII multiplexer, data can be routed from *APIX* Rx to *APIX* Tx (Phy) and/or to the MAC of the firmware for instance. The *APIX* IC has two **A-Shell** channels. In the INAP562T, both **A-Shell** channels are routed out as MII (Tx). With the INAP562R only one (Rx) is routed out. All MIIs are bidirectional.

All possible PHY Rx and MAC Tx instances can be routed to an input of the switch and all PHY Tx and MAC Rx instances can be routed to the output. The possible sources and targets are listed in the drop-down lists. With the help of the graphic the correct instance can be specified in the input field below the dropdown list. All incoming packets of the switch are routed to the switch output. The priority is indicated by the number of the input. So Switch Input0 has the highest priority and Switch Input2 the lowest.



This priority distribution is only valid for the *APIX* interface. This is not the case for other *GÖPEL electronic* devices with MII Multiplexer.

In the middle of the dialog window there are three buttons:

Button	Description
Get	Load the current MII Multiplexer setting values of the selected interface.
Set	Overwrite all current MII Multiplexer settings on the device with the settings displayed on the tabs of the window.
reset own	Reset all MII Multiplexer settings of the selected interface.

The defined connections can be found in the connection table below the graphic. The channel entries of the connection table can be changed manually by clicking on the desired field and editing the value. Use the **Apply** button to set the values. Use the **Refresh** button to load the actual settings.

7.9.1 Example

This is a short example of pass-through of data from *APIX Rx-A0* to *APIX Tx-A0*. Instance 2 of *Phy_Rx* needs to be routed to Instance 3 of *Phy_Tx*. The other way around Instance 3 of *Phy_Rx* needs to be routed to Instance 2 of *Phy_Tx*.

Deserializer: INAP562R

General | Signal Levels | Signal Routing | SerDes Config | External Board | Sideband Settings | LVDS Channels | IO Routing | Ethernet

Source: PHY RX, 3 instance(APIX-TX 0)

Target: PHY TX, 2 instance(APIX-RX 0)

Ethernet Mii Multiplexer LVDS4

Get Set reset own

PhyRx/Tx

Instance 1 Phy/G Patch, Instance 2 Apix Rx A0, Instance 3 Apix Tx A0, Instance 4 Apix Tx A1

Mii Multiplexer

Switch Instance 1, Switch Instance 2

MAC (firmware)

Current connections

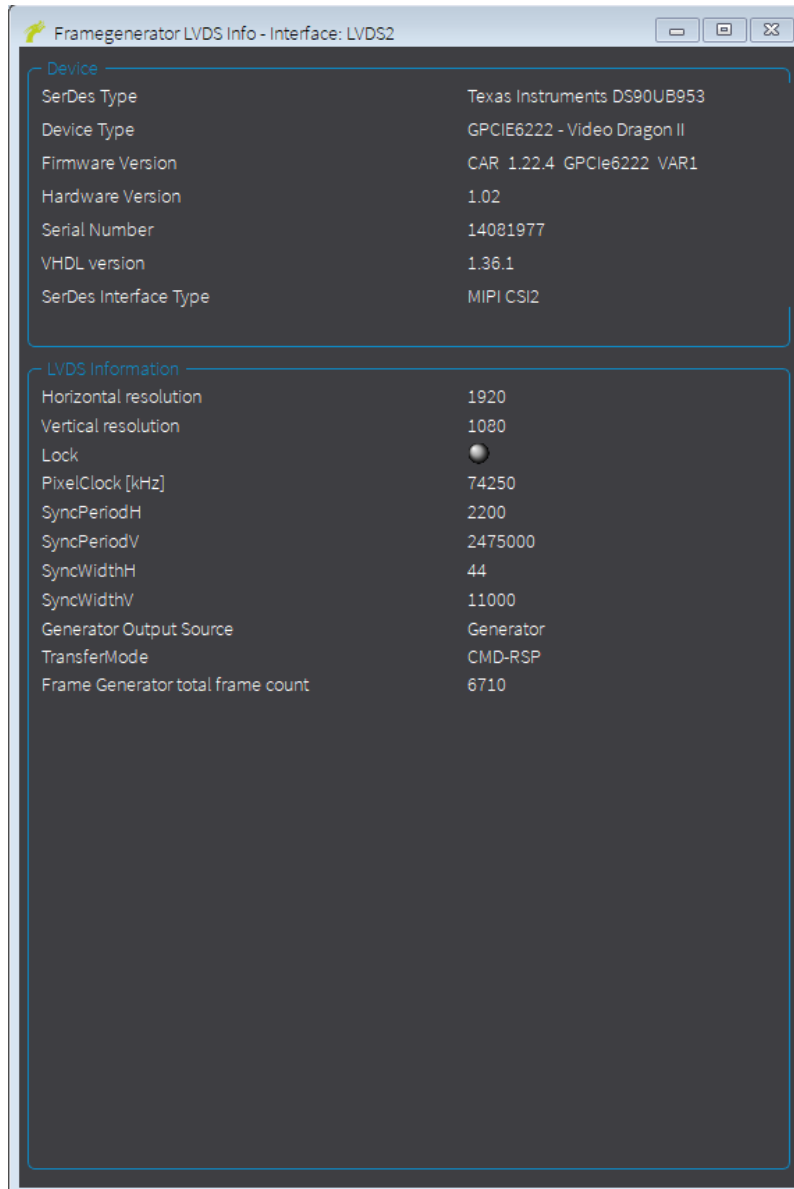
Sid	Source	channel	Tid	Target	channel
0	PHY_RX	1	0	MAC_RX	1
1	MAC_TX	1	1	PHY_TX	1
0	PHY_RX	3	1	PHY_TX	2
0	PHY_RX	2	1	PHY_TX	3

apply refresh

7.10 LVDS Info



The *Frame Generator* LVDS Info window for the selected interface can be opened with the icon ()(ALT + 6).



The device information contains:

- SerDes Type: The type of the serializer mounted on the currently installed extension board of the LVDS device.
- Device Type: The type of the LVDS device.
- Firmware Version: The firmware version running on the device.
- Hardware Version: The hardware revision number.
- Serial Number: The serial number of the LVDS device.
- VHDL Version: The version number of the VHDL design of the FPGA device.
- SerDes Interface Type: Serializer interface used

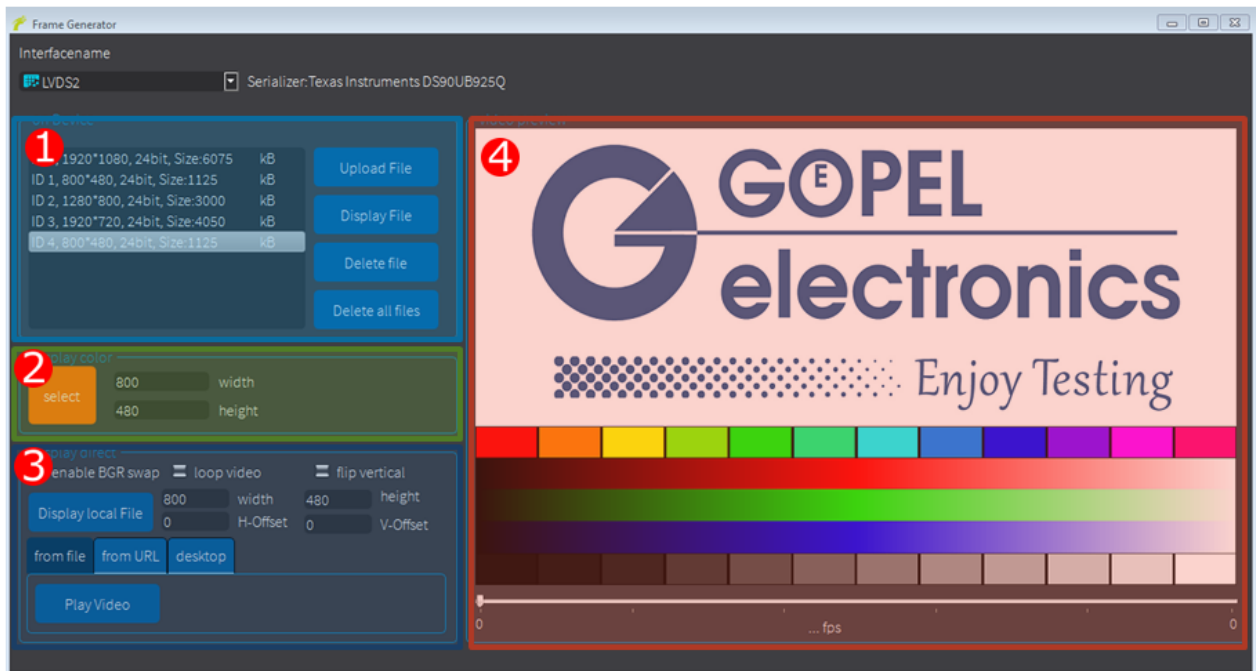
The LVDS information contains information about the current video stream:

- LVDS Channel: Select the LVDS Channel. (only supported for *Video Dragon 6222*).

- Horizontal Resolution: Number of horizontal pixel clock cycles where the Data Enable signal is active. The value corresponds to the horizontal resolution of the image source.
- Vertical Resolution: Number of vertical pixel clock cycles where the Data Enable signal is active. The value corresponds to the vertical resolution of the image source.
- Lock: Indicates whether the deserializer is synchronous to the LVDS bit stream. (only for supporting Media Interface Boards).
- PixelClock: Frequency of pixel clock in kHz.
- SyncPeriodH: Absolute number of pixel clock cycles between two horizontal synchronization edges.
- SyncPeriodV: Absolute number of pixel clock cycles between two vertical synchronization edges.
- SyncWidthH: Number of pixel clock cycles during the horizontal synchronization is active. The value corresponds to the [Horizontal Sync Width](#) of the image source.
- SyncWidthV: Number of pixel clock cycles during the vertical synchronization is active. The value corresponds to the product of the total horizontal active area [hTotal](#) and the [Vertical Sync Width](#) of the image source.
- Generator Output Source: Video Source set
- Transfer mode: Mode, how the data is transferred (Possible modes: Command-Response mode, DMA mode)
- Frame Generator total frame count: Number of frames sent since last initialization

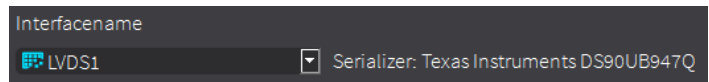
8 Frame Generator Dialog Window

Open the *Frame Generator* dialog window with one of the alternatives illustrated in chapter [Using the GUI](#). The dialog window has four segments that allow you to manage the files on the device and view the frames.



- 1 The [Files on Device](#) overview
- 2 The [Display Color](#) box
- 3 The [Display Direct](#) box
- 4 The [Video Preview](#)

All available *Frame Generator* interfaces are listed in the drop down menu. The currently selected interface is shown in the text field of the drop down menu.



When working with *Video Dragon 6222* use the button  to switch between [Files on Device](#) and [Pattern Generator](#).

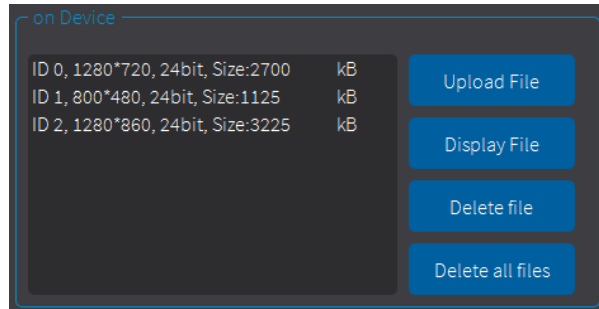
8.1 Files on the Device

This dialog segment displays the files stored on the device and has several buttons to manage them.



This functionality is only available for *basicCON 4121*.

The table lists all files stored in the internal *Frame Generator* memory. Each file has a file number (ID) that starts with 0. When you upload a new file, it gets the lowest vacant ID. Next to the ID, the table displays the width and height of the stored images in pixels, the bit depth in bits and the file size in kilobytes. To select a file, left-click in the list. The file is highlighted in the list and the image is also displayed in the preview segment.



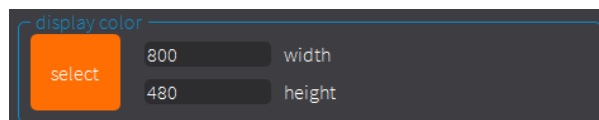
On the right side of the segment are four buttons with the following functions:

Button	Description
Upload File	Open a window to select a path to the desired file. The file must be in 24-bit bitmap format. The selected file is saved in the internal memory of the <i>Frame Generator</i> and displayed in the file list on the left.
Display File	Display the file selected in the file list. The selected file width or height can not be larger than configured in the <i>Frame Generator</i> settings .
Delete file	Delete the selected file from the internal memory of the <i>Frame Generator</i> immediately.
Delete all files	Delete all files from the internal memory of the <i>Frame Generator</i> immediately and reset the internal file system to a clean state. Depending on the number of files stored in the memory, this action may take several minutes.

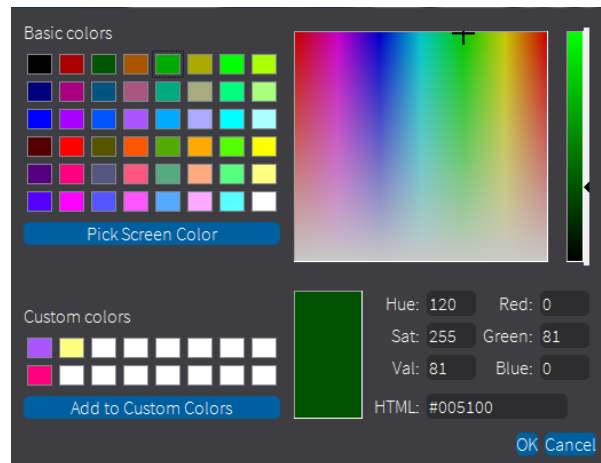
8.2 Display color

The display color segment provides the ability to select a color to be displayed with the *Frame Generator* as a frame.

The width and height of the frame are arbitrary. Any value from 1 to the maximum width or height configured in the *Frame Generator* [settings](#) is valid. Clicking in the colored **select** field opens an additional window for configuring the displayed color.



After selecting a color and confirming with **OK**, the color is displayed with the *Frame Generator*.



8.3 Display Direct

This segment allows a file or video to be displayed from the PC with the *Frame Generator* without having to upload the file to the device.

Select the **Color Format** to generate from the drop-down list. Additionally select **enable BGR swap** to switch between RGB and BGR.

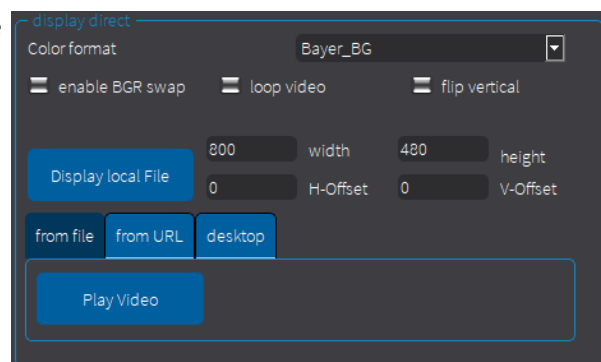
Click **loop video** to generate the video in infinite loop. Select **flip vertical** to mirror the image vertically.

If the file size is larger than the defined active area, the file is trimmed into the vertical and horizontal active area, beginning at the top left pixel of the file. The size can also be set in this segment using the width and height edit fields. Furthermore, an offset can be defined so as not to start the frame at the upper left pixel but at the desired one.



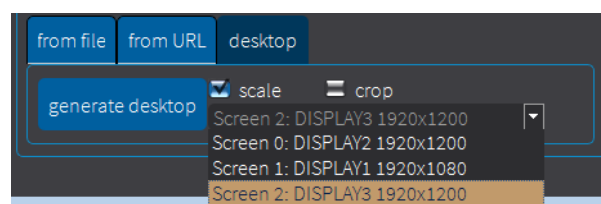
The specified offset plus file size must not be larger than defined in *Frame Generator* settings.

The **Display local File** button selects an image file to be displayed with the *Frame Generator*. The file type must be a prevalent image file type, such as JPEG or bitmap. To display a video, go to the **from file** tab and use the **Play Video** button. Then a window opens for selecting a file. After selecting a video file, the *Frame Generator* displays the video prompt.



Use the **from URL** tab to display a stream from a URL, such as a webcam.

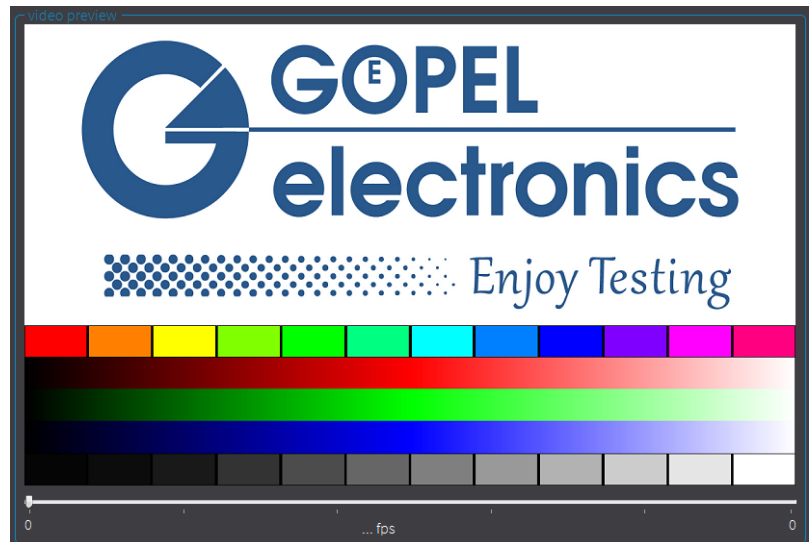
The **desktop** tab is used to display the current desktop screen. When using several screens, the desired screen can be selected using the drop down menu. The *Frame Generator* can display the entire screen by downscaling the overall frame to the size of the active area. Therefore use the flag **scale**. To display only a cutout of the overall frame, use the flag **crop**.



For direct video presentation, there are some functions that are additionally selectable with the appropriate flag. You can enable the BGR swap, loop the video in a loop by restarting it at the end or flip the image vertically.

8.4 Video Preview

This segment displays the preview image from a file selected in the [device file list](#). Additionally, selected videos played in segment [Display Direct](#) are displayed. The scale at the bottom of the segment shows the video timeline and its frame rate in frames per second. While displaying, you can use this timeline to rewind and play the video.

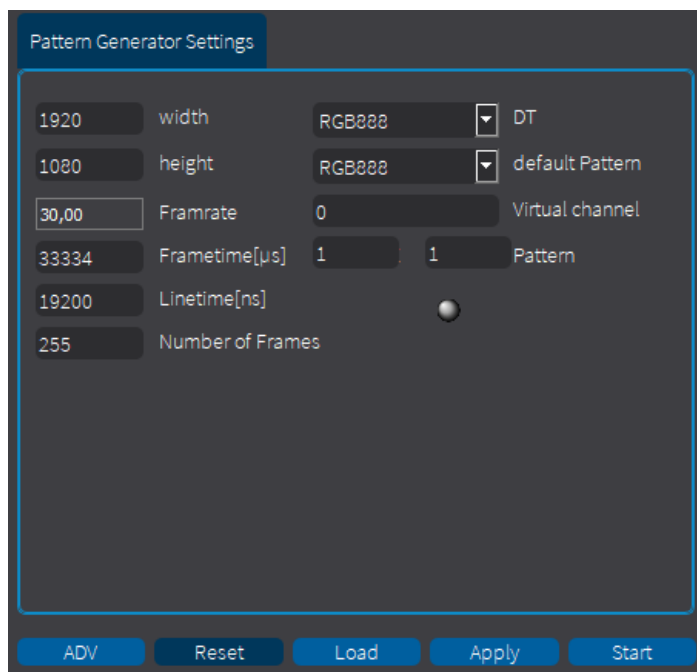


8.5 Pattern Generator



This feature is only supported by *Video Dragon 6222*.

The *Video Dragon 6222* provides the ability to generate LVDS image data internally. Therefore, an external source is not necessary.



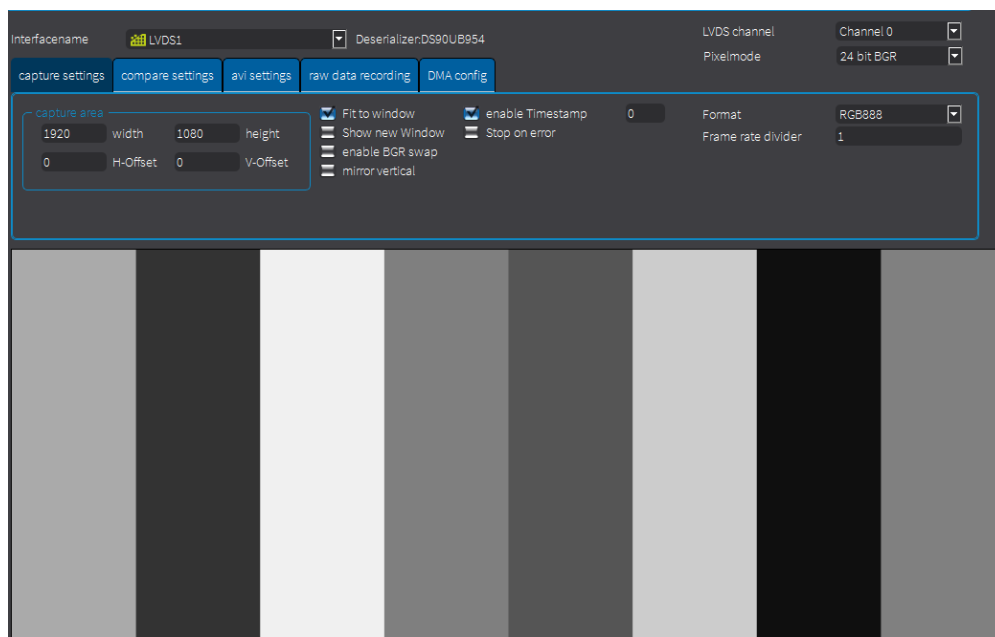
In this segment the buttons have the following functions:

Button	Description
Init	Start initialisation and unlock the settings options.
Apply	Apply the current settings.
Start	Start the Pattern Generator.
Stop	Stop the Pattern Generator.
Reset	Reset all settings to default.
ADV	Open Advanced Settings Window for detailed color setting.

When the Init button is clicked, the Pattern Generator is initialized to the default values. Now the Settings parameters can be changed as desired and set with Apply. The following parameters are available:

Parameter	Description
Width/Height	Width and Height of the image.
Framerate	Framerate of the image.
Frametime	Frametime of the image in microseconds (read only).
Linetime	Linetime of the image in nanoseconds (read only).
Number of Frames	Number of frames that are displayed from 1 to 254. 255 means infinite displaying.
DT	Picture format of the pattern (RGB888 and YUV 4:2:2 8 Bit mode are possible).
Virtual Channel	Virtual channel ID for MIPI CSI-2.
Pattern	Number of individual pattern segments in horizontal and vertical dimension from 1 to 8.

The Pattern Generator can only be started after initialization. As long as an image is being generated, the indicator light in the Pattern Generator Settings window lights up green.



8.5.1 Advanced Pattern Generator



This feature is only available when [Dragon Suite Advanced](#) is activated.

In the advanced Pattern Generator window, the 8 reference pixels of the individual patterns can also be set. Define the number of vertical and horizontal patterns (8 at maximum) and the table below is enlarged or reduced accordingly. Click into the table cells and change the values. The values result from 2 rows of 4 pixels each as RGB hex values.

Advanced pattern generator config LVDS2 LVDS2 - Dragon Suite 1.733

Pattern Generator Settings

1920	width	RGB000	DT
1080	height	RGB000	default Pattern
30,00	Framrate	0	Virtual channel
33333	Frametime[μs]	0	Pattern
9000	Linetime[ns]		
255	Number of Frames		

Reset Load Apply Start

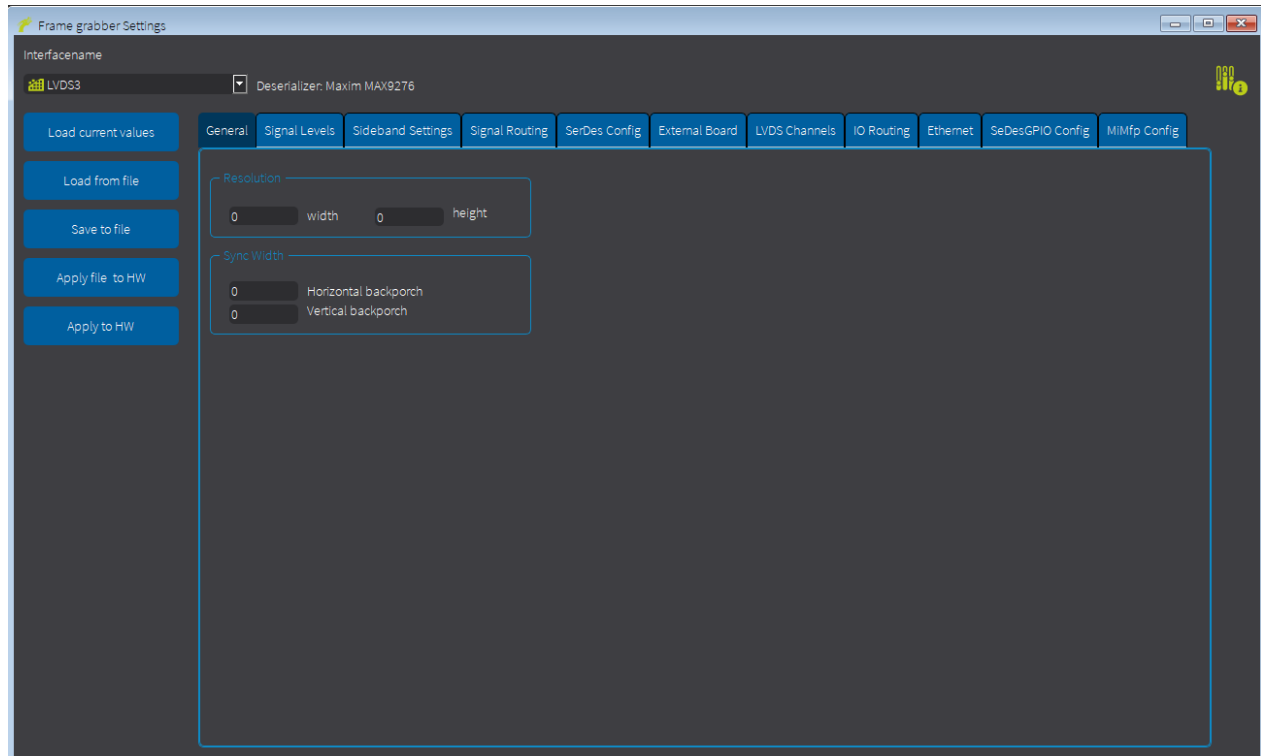
#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678
#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678
#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678
#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678	#12345678

9 Setting up the Frame Grabber

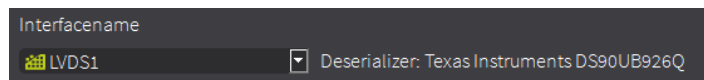
Open the *Frame Grabber* settings window by using one of the alternatives illustrated in chapter [Using the GUI](#). The settings window shows all supported device settings. Most of the functionalities of the *Dragon Suite* depend on valid settings of the interface. So setting up the device should be the first step after starting the software.



Generally parallel usage of several interfaces with the *G-API* is possible. But the *Dragon Suite* supports the usage of only one *Frame Grabber* interface at a time.



All available *Frame Generator* interfaces are listed in the drop down menu. The currently selected interface is shown in the text field of the drop down menu.

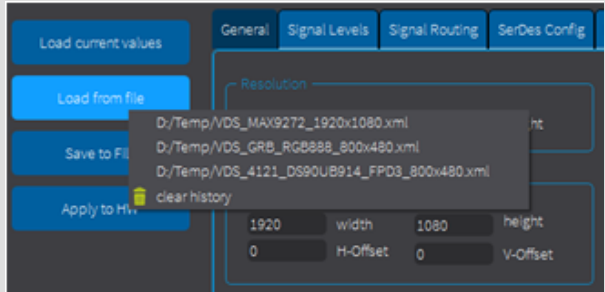


On the left of the settings window are the following buttons:

Button	Description
Load current values	Load the current setting values of the selected interface.
Load from file	Load the values for the selected interface by importing an external XML file. Importing a settings file does not overwrites the current settings on the device. To overwrite the device settings, use the Apply to HW button.
Save to File	Save the current settings of the selected interface by exporting them to an external XML file.
Apply file to HW	Overwrite the current settings directly from the configuration file.
Apply to HW	Overwrite all current settings on the device with the settings displayed on the tabs of the window.

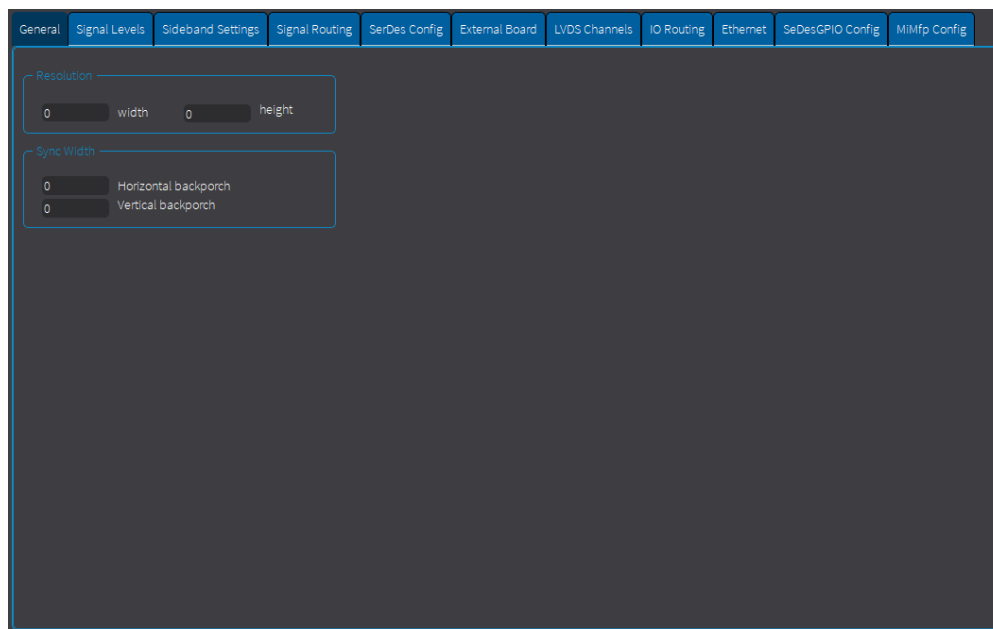
TIP

A right click on the **Load from file** button opens a selection of the last opened files. This facilitates the search for frequently used configurations. The history can be cleared with **clear history**.



9.1 General

The general deserializer settings are resolution and sync width parameters for *basicCON 4121* and resolution for *Video Dragon 6222*.



The **Resolution** is specified by the number of columns (width) and the number of rows (height) of a bitmap graphic. The maximum resolution of the frame that can be captured is displayed in width and height. These parameters are purely informative for the user and do not need to be modified.

In exceptional cases, no bit should be assigned to the data enable signal in the signal routing settings (see section [Signal Routing](#)). If no data enable signal is being transmitted within the LVDS stream, the vertical and horizontal back porches must be set to allow the LVDS device to determine the beginning of the valid pixel data within the stream. These back porch settings in the General Settings tab are called **Sync Width**. The **Vertical back porch** must be set to the number of lines transmitted after a vertical sync signal until a line of valid pixel data begins. The **Horizontal back porch** must be set to the number of columns (pixels) that are transmitted within one line after a horizontal sync signal until valid pixel data begins. When a data enable signal is transmitted and correctly assigned in signal routing, the values of the porches (sync widths) are not important and will not be evaluated. Likewise, the value of the signal level setting of the data enable signal is not evaluated if no data enable signal is transmitted or assigned in the signal routing.

9.2 Signal Levels

The valid signal levels and edges are defined in the Signal Levels tab.



This parameters are only necessary for *basicCON 4121*.

Polarity	Defines which signal level indicates vertical or horizontal synchronization.
Data Enable	Specifies at which level pixel data is being transmitted.
Pixel Clock Polarity	Describes at which edge of the pixel clock signal (rising or falling) the values of the signals are to be sampled.

9.3 Signal Routing



This functionality is only available for *basicCON 4121*.

Since there is no common standard for mapping the video signals to the 32 serialized bits in the LVDS video stream, this mapping must be defined for each device. This could be done on the Signal Routing tab.

The screenshot shows the 'Signal Routing' tab with four main sections: Red, Green, Blue, and Control Signals. Each section contains a list of signals and their corresponding video bit assignments.

Signal	Assignment
R0	Video bit 0
R1	Video bit 1
R2	Video bit 2
R3	Video bit 3
R4	Video bit 4
R5	Video bit 5
R6	Video bit 6
R7	Video bit 7
G0	disabled
G1	disabled
G2	disabled
G3	disabled
G4	disabled
G5	disabled
G6	disabled
G7	disabled
B0	Video bit 0
B1	Video bit 1
B2	Video bit 2
B3	Video bit 3
B4	Video bit 4
B5	Video bit 5
B6	Video bit 6
B7	Video bit 7
HSync	Video bit 25
VSync	Video bit 24
DE	Video bit 26

To deserialize the data within the LVDS data stream, the deserializer needs to know which bit of the data stream should be deserialized to which data. Within the signal routing tab it is possible to define the assignment of each serialized bit of the LVDS data stream to its meaning in the non-serialized video data. Therefore, the combo boxes of the tab provide the ability to set the color bits. Since the resulting RGB frame has a color depth of 24 bit, each color (red, green and blue) has a maximum depth of 8 bits. The selection of the corresponding bit in the video stream is made by selecting the correct from the possibilities of the combo box. All 32 bits of the stream and the value **disabled** can be selected. **Disabled** means that the signal has no correspondent in the video stream. This should be done, for example, if a video stream contains frames with a color depth of less than 24 bits. The significance of the color bits is ascending. For example, this means for an 8-bit color value of red, **R0** is the least significant bit, and **R7** is the most significant bit.

The control signal boxes include the vertical sync signal **VSync**, which indicates the end of the transmission of a complete frame, while the horizontal sync signal **HSync** indicates the end of the transmission of a line. The third control signal is the **Data Enable** signal. In a continuous stream, the video signal may contain porches in each single line and between the end and the beginning of a new frame. The data enable signal indicates whether pixel data is currently being transmitted.

The figure above shows a possible example of video routing. The video stream contains 24-bit RGB frames, with the first 8 bits in the stream representing the 8-bit blue value in ascending order (bit 0 is the least significant, bit 7 is the most significant bit). The following 16 bits represent the green and red values, but without any order. The horizontal sync signal is bit 24, followed by the vertical sync signal and the data enabled signal. Bits 27 to 31 have no assigned values.



There is no common standard for serializing the video data into a 32-bit LVDS stream. The assignment of the several bits to their respective meaning within the deserialization process therefore requires exact knowledge of the settings of the serializer used. The use of wrong settings will most likely lead to incorrect

frame data (i.e. wrong colors) or no valid frame data at all. The LVDS device can not determine if the user has entered valid settings and can report capture errors due to these settings, even though the transmitted frame data is correct.

9.4 SerDes Config

This segment is described in the [Framer Generator Settings](#).

9.5 External Board

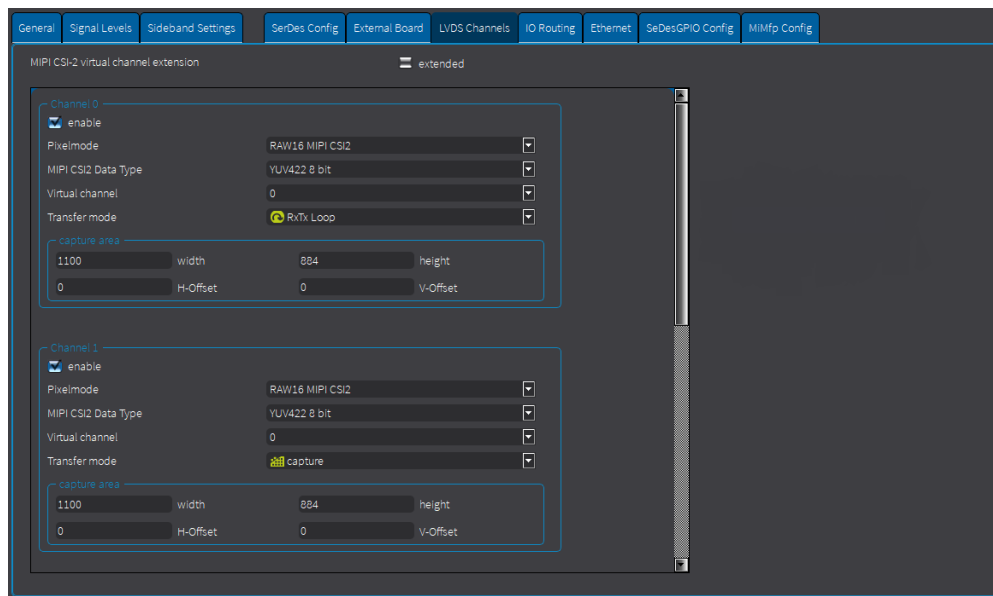
This segment is described in the [Frame Generator settings](#).

9.6 Sideband Settings

This segment is described in the [Frame Generator settings](#).

9.7 LVDS Channels

Use this tab to configure the capture parameters for the individual channels of the *Video Dragon*. The Capture Area parameter is available for *basicCON 4121* and *Video Dragon 6222*.



Capture Area defines the resolution of the frame to be captured, starting with the upper left pixel of the transmitted frame. Any value from 1 to the real maximum width or height of the transmitted video frames is valid. If the values are below the transmitted frame, the captured frame is part of the transmitted frame, starting with the upper left pixel. Larger values than the transmitted frame will result in an [error message](#) as soon as a frame has to be captured. To capture a frame with a defined offset, use H-Offset to ignore columns on the left side of the frame, or V-Offset to disregard rows at the top of the frame.



The following properties are only supported for *Video Dragon 6222* with MIPI CSI-2 and OLDI virtual channel extension. Not all Media Interface Boards support all parameters.

From *Video Dragon 6222*, several physical channels (Channel 0, Channel 1, etc.) lead to the PC. The number of virtual channels depends on the *Media Interface* module used. Depending on the configuration, the virtual channels can go through one of the physical channels.

Enable or disable the physical channel by checking the **enable** box. You can select the **Pixel Mode** and the **Data Type** for each physical channel.

For the **MIPI CSI-2 Data Type** the following data formats are possible:

- RGB888
- RAW6
- RAW7
- RAW8
- RAW10
- RAW12
- RAW14
- YUV422 8bit
- YUV422 10 bit
- embedded data (raw data packet, not necessarily just video data)

For the **OLDI Type** the following data formats are possible:

- RGB 18 bit
- RGB 24 bit
- RGB 30 bit

For the **Pixel Mode** the following modes are possible:

Mode	Description
24 bit BGR	24 bit BGR format with 3 bytes per pixel. Per byte one color channel is transmitted. Byte 0: Blue (bit 7..0) Byte 1: Green (bit 7..0) Byte 2: Red (bit 7..0)
12 bit	RAW12 format. Here two pixels split into three bytes (little endian). Byte 0: Pixel 1 (bit 7..0) Byte 1: Pixel 2 (bit 3..0) und Pixel 1 (bit 11..8) Byte 2: Pixel 2 (bit 11..4)
RAW 8 (only Video Dragon 2)	RAW8 format. Per byte one pixel is transmitted. Byte 0: Pixel 1 (bit 7..0) Byte 1: Pixel 2 (bit 7..0) Byte 2: Pixel 3 (bit 7..0) Byte 3: Pixel 4 (bit 7..0)
RAW 10 MIPI CSI2 (only Video Dragon 2)	RAW10 format. Here four pixels split into five bytes. Recommended memory storage format. Byte 0: Pixel 1 (bit 9..2) Byte 1: Pixel 2 (bit 9..2) Byte 2: Pixel 3 (bit 9..2) Byte 3: Pixel 4 (bit 9..2) Byte 4: Pixel 4 (bit 1..0) and Pixel 3 (bit 1..0) and Pixel 2 (bit 1..0) and Pixel 1 (bit 1..0)
RAW 12 MIPI CSI2 (only Video Dragon 2)	RAW12 Format. Here two pixels split into three bytes (according to MIPI-CSI-2 specification). Byte 0: Pixel 1 (bit 11..4) Byte 1: Pixel 2 (bit 11..4) Byte 2: Pixel 2 (bit 3..0) and Pixel 1 (bit 3..0)
RAW 16 MIPI CSI2 (only Video Dragon 2)	RAW16 format. One pixel is set to two bytes (according to MIPI-CSI-2 specification). Byte 0: Pixel 1 (bit 15..8) Byte 1: Pixel 1 (bit 7..0)
RAW 12 16 LE (only Video Dragon 2)	RAW12 format. One pixel is set to two bytes, with byte 0 shifted 4 bits to the left (little endian). Byte 0: Pixel 1 (bit 3..0) 0b0000 Byte 1: Pixel 1 (bit 11..4)
RAW 16 LE (only Video Dragon 2)	RAW16 format. One pixel is set to two bytes (little endian). Byte 0: Pixel 1 (bit 7..0) Byte 1: Pixel 1 (bit 15..8)
YUV 422 8 MIPI CSI (only Video Dragon 2)	Byte 0: Y2 Byte 1: V1 Byte 2: Y1 Byte 3: U1
YUV 422 8 Y1U1Y2U2 (only Video Dragon 2)	Byte 0: Y1 Byte 1: U1 Byte 2: Y2 Byte 3: V1
YUV 422 10 MIPI CSI2 (only Video Dragon 2)	YUV422 10 bit MIPI CSI-2. Here YUV data were split into five bytes. Recommended memory storage format. Byte 0: U1 (bit 9..2) Byte 1: Y1 (bit 9..2) Byte 2: V1 (bit 9..2) Byte 3: Y2 (bit 9..2) Byte 4: Y2 (bit 1..0) and V1 (bit 1..0) and Y1 (bit 1..0) and U1 (bit 1..0)
BGR 888 24 (only Video Dragon 2)	Byte 0: B1[7..0] Byte 1: G1[7..0] Byte 2: R1[7..0]

Mode	Description
RGB 888 24 (only Video Dragon 2)	Byte 0: R1[7..0] Byte 1: G1[7..0] Byte 2: B1[7..0]
RGB 101010 30 (only Video Dragon 2)	30 bit RGB (10 bit red, 10 bit green, 10 bit blue) packed pixel stream supported IfTypes: OPEN_LDI Byte 0: R1[7..0] R1.7 R1.6 R1.5 R1.4 R1.3 R1.2 R1.1 R1.0 Byte 1: G1[5..0] R1[9..8] G1.5 G1.4 G1.3 G1.2 G1.1 G1.0 R1.9 R1.8 Byte 2: B1[3..0] G1[9..6] B1.3 B1.2 B1.1 B1.0 G1.9 G1.8 G1.7 G1.6 Byte 3: R2[1..0] B1[9..4] R2.1 R2.0 B1.9 B1.8 B1.7 B1.6 B1.5 B1.4 Byte 4: R2[9..2] R2.9 R2.8 R2.7 R2.6 R2.5 R2.4 R2.3 R2.2 Byte 5: G2[7..0] G2.7 G2.6 G2.5 G2.4 G2.3 G2.2 G2.1 G2.0 Byte 6: B2[5..0] G2[9..8] B2.5 B2.4 B2.3 B2.2 B2.1 B2.0 G2.9 G2.8 Byte 7: R3[3..0] B2[9..6] R3.3 R3.2 R3.1 R3.0 B2.9 B2.8 B2.7 B2.6 Byte 8: G3[1..0] R3[9..4] G3.1 G3.0 R3.9 R3.8 R3.7 R3.6 R3.5 R3.4 Byte 9: G3[9..2] G3.9 G3.8 G3.7 G3.6 G3.5 G3.4 G3.3 G3.2 Byte 10: B3[7..0] B3.7 B3.6 B3.5 B3.4 B3.3 B3.2 B3.1 B3.0 Byte 11: R4[5..0] B3[9..8] R4.5 R4.4 R4.3 R4.2 R4.1 R4.0 B3.9 B3.8 Byte 12: G4[3..0] R4[9..6] G4.3 G4.2 G4.1 G4.0 R4.9 R4.8 R4.7 R4.6 Byte 13: B4[1..0] G4[9..4] B4.1 B4.0 G4.9 G4.8 G4.7 G4.6 G4.5 G4.4 Byte 14: B4[9..2] B4.9 B4.8 B4.7 B4.6 B4.5 B4.4 B4.3 B4.2

From the **Virtual Channel** drop-down menu, you can select the virtual channel to be routed to the appropriate physical channel. Depending on whether the extended box is checked or not, you can choose between 4 or 16 (extended) virtual channels. The parameter applies globally to all physical channels.

For the **Transfer Mode** the following data formats are possible:

- Generate
- RxTx Loop

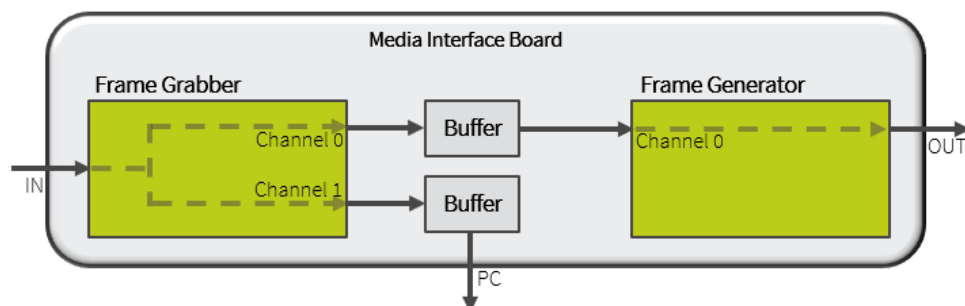
Write the displayed settings in the window by clicking the **Apply** button. When you use this button, the settings of the other tabs are not written to the device.

9.7.1 Open LDI Mode

This segment is described in the [Frame Generator settings](#).

9.7.2 RxTx Loop

RxTx Loop is used to pass incoming LVDS data internally from the *Frame Grabber* to the *Frame Generator*.



For this, the loop must first be configured on the *Frame Grabber* side. This will create the internal buffer.

Channel 0

☒ enable

Pixelmode RAW16 MIPI CSI2

MIPI CSI2 Data Type YUV422 8 bit

Virtual channel 0

Transfer mode RxTx Loop

capture area

1100 width 884 height

0 H-Offset 0 V-Offset

After that, the frame generator must be configured accordingly. For example, if channel 0 of the frame grabber is set to RxTxLoop, the Rx channel must be set to 0 on the frame generator. Thus, the previously created buffer is then grabbed.

Channel 0

Pixelmode 24 bit BGR

MIPI CSI2 Data Type RGB888

Virtual channel 0

Transfer mode RxTx Loop ☒ Rx LVDS channel 0


9.8 IO Routing

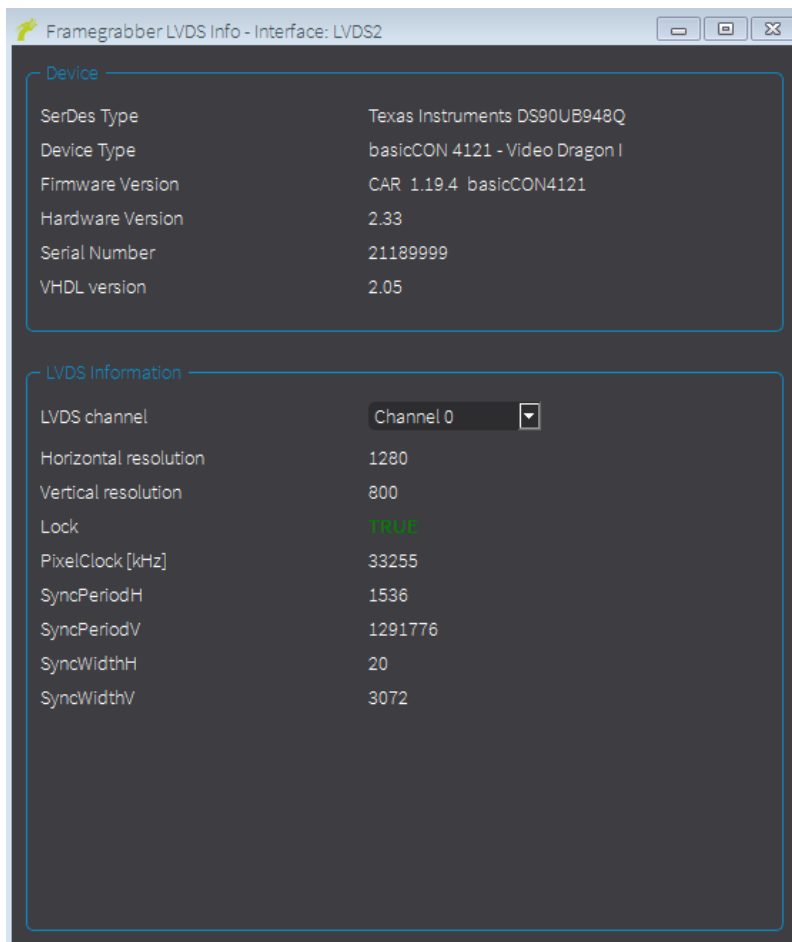
This segment is described in the [Frame Generator settings](#).

9.9 Ethernet

This segment is described in the [Frame Generator settings](#).

9.10 LVDS Info

The *Frame Grabber* LVDS Info window for the selected interface can be opened with the icon ()(ALT + 6). The window consists of two parts: the Device Information and the LVDS Information. For *basicCON 4121* and *Video Dragon 6222* the device information is the same. However, the LVDS information differs between the two devices.



The device information contains:

- SerDes Type: The type of the deserializer mounted on the currently installed extension board of the LVDS device.
- Device Type: The type of the LVDS device.
- Firmware Version: The firmware version running on the device.
- Hardware Version: The hardware revision number.
- Serial Number: The serial number of the LVDS device.
- VHDL Version: The version number of the VHDL design of the FPGA device.

9.10.1 LVDS Information of basicCON 4121

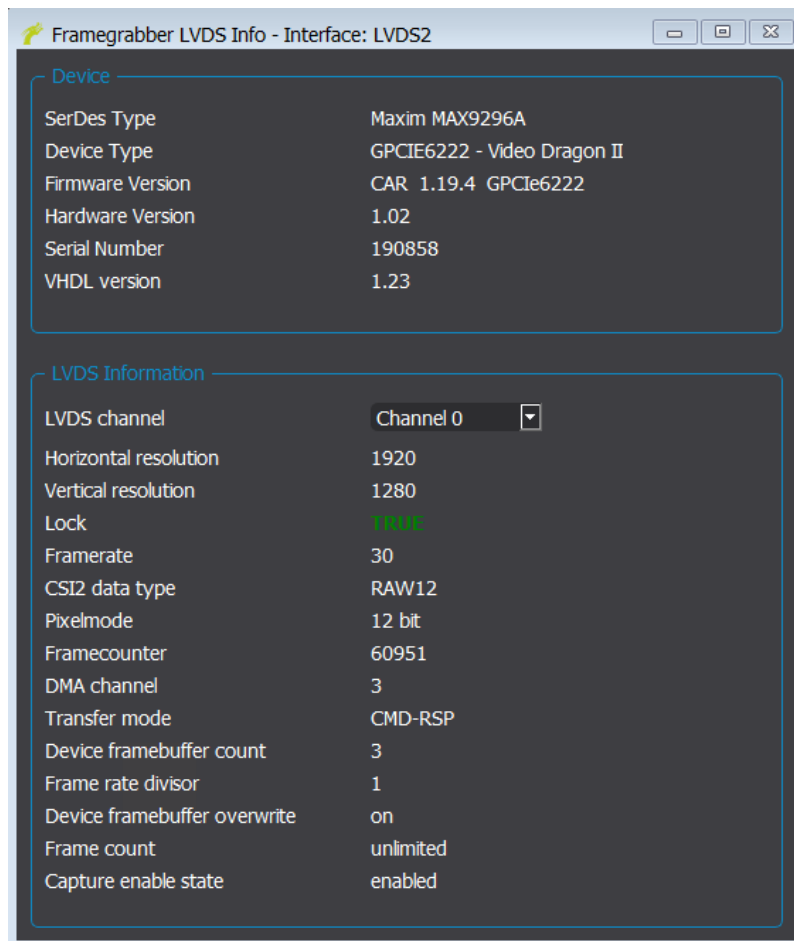
The LVDS information contains information about the current *basicCON 4121* video stream:

- LVDS Channel: Select the LVDS Channel. (for *basicCON 4121* only one channel is supported).
- Horizontal Resolution: Number of horizontal pixel clock cycles where the Data Enable signal is active. The value corresponds to the horizontal resolution of the image source.
- Vertical Resolution: Number of vertical pixel clock cycles where the Data Enable signal is active. The value corresponds to the vertical resolution of the image source.
- Lock: Indicates whether the deserializer is synchronous to the LVDS bit stream.
- PixelClock: Frequency of pixel clock in kHz.
- SyncPeriodH: Absolute number of pixel clock cycles between two horizontal synchronization edges.
- SyncPeriodV: Absolute number of pixel clock cycles between two vertical synchronization edges.
- SyncWidthH: Number of pixel clock cycles during the horizontal synchronization is active. The value corresponds to the [Horizontal Sync Width](#) of the image source.
- SyncWidthV: Number of pixel clock cycles during the vertical synchronization is active. The value corresponds to the product of the total horizontal active area [hTotal](#) and the [Vertical Sync Width](#) of the image source.

9.10.2 LVDS Information of Video Dragon 6222

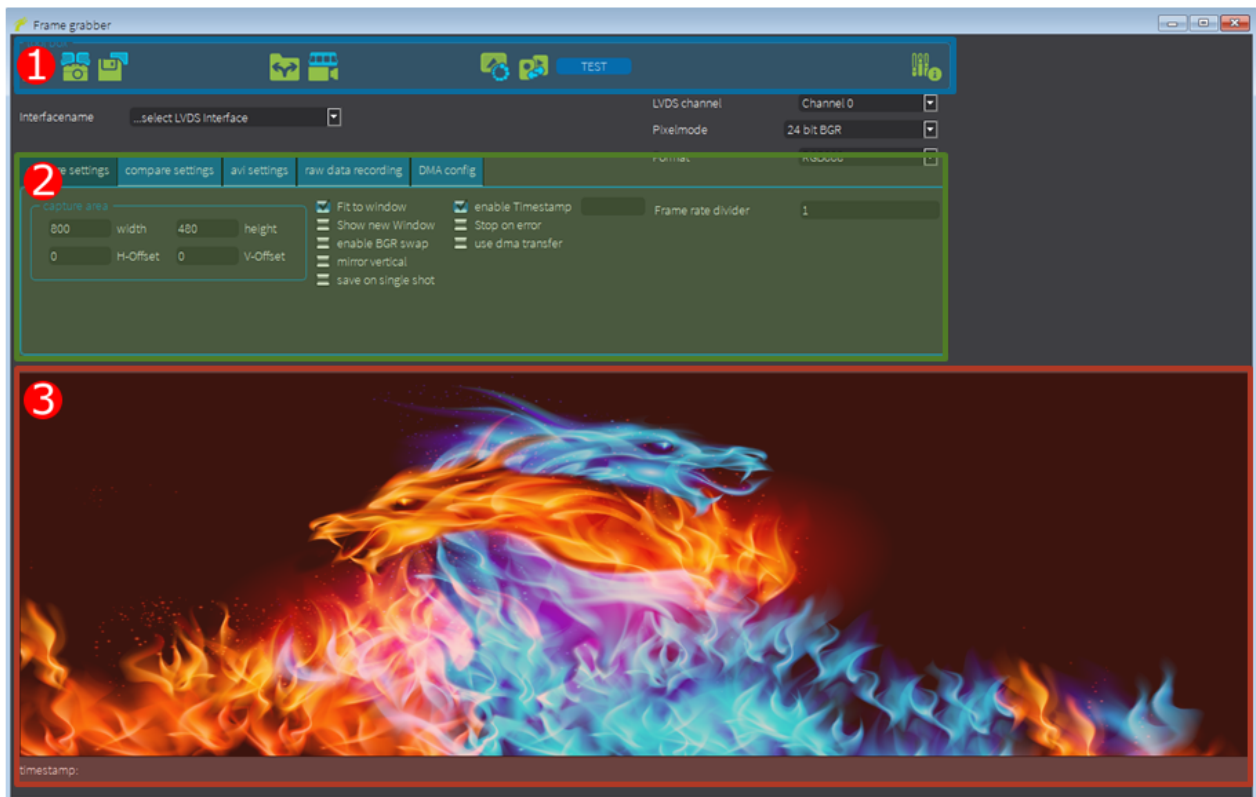
The LVDS information contains information about the current *Video Dragon 6222* video stream:

- LVDS Channel: Select the LVDS Channel.
- Horizontal Resolution: Number of horizontal pixel clock cycles where the Data Enable signal is active. The value corresponds to the horizontal resolution of the image source.
- Vertical Resolution: Number of vertical pixel clock cycles where the Data Enable signal is active. The value corresponds to the vertical resolution of the image source.
- Lock: Indicates whether the deserializer is synchronous to the LVDS bit stream.
- Frame rate: The speed at which the images are sent in FPS (Frames per second).
- CSI2 data type: MIPI CSI-2 type of the data.
- Pixel mode: Pixel Mode of the data.
- Frame counter: Number of captured frames.
- DMA channel: Used DMA channel.
- Transfer mode: Mode, how the data is transferred (Possible modes: Command-Response mode, DMA mode).
- Device framebuffer count: Number of frame buffers within the device.
- Frame rate divisor: Specifies whether all frames or every x-th frame should be captured.
- Device framebuffer overwrite: Specifies whether full frame buffers are overwritten when new frames receive.
- Frame count: Number of frames to capture. When number of frames to capture is reached, frame capturing is automatically disabled.
- Capture enable state: Specifies whether Capture is enabled or disabled.



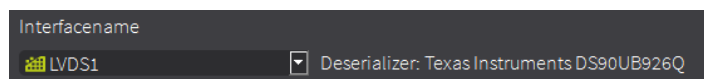
10 Frame Grabber Dialog Window

Open the *Frame Grabber* dialog window with one of the alternatives illustrated in chapter [Using the GUI](#). The dialog window has several segments to manage the capturing process.

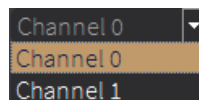


- 1 The [Tool Box](#)
- 2 Several tabs:
 - [Capture Settings](#)
 - [Compare Settings](#)
 - [Avi Settings](#)
 - [Raw Data Recording](#)
 - [DMA Config](#)
- 3 The [Frame Area](#)

All available *Frame Generator* interfaces are listed in the drop down menu. The currently selected interface is shown in the text field of the drop down menu.



The **LVDS Channel** setting in the dropdown menu on the right can be used for *Video Dragon 6222* only. Here you can choose between LVDS channels 0 and 1.



The **Color Data Format** and **Pixel Mode** of the captured frame can be changed in the dropdown menus on the right.

Depending on the *Media Interface* board, the possible values are displayed in the drop-down list. More detailed information can be found in the *G-API* manual.



For a more complete way to customize the LVDS channels for *Video Dragon 6222*, see the [LVDS Channels](#) tab in Frame Grabber Settings section.

10.1 Tool Box

The Tool Box segment provides several icons with the following functions:

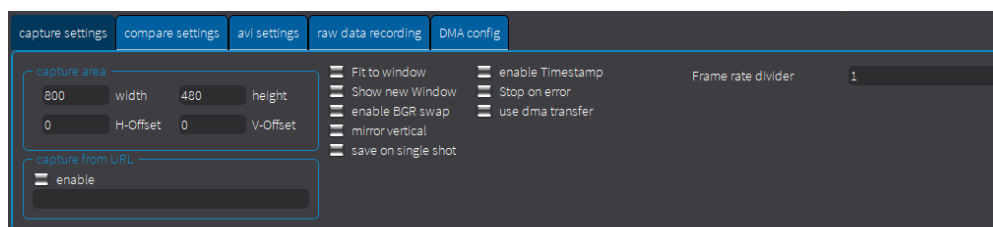
Icon	Description
	Capture a single frame from the selected device and display it in the frame area (ALT + 1).
	Capture images continuously and display them in the frame area. The capturing starts immediately after clicking and stops after clicking again. The icon will remain red while capturing is in progress (ALT + 2).
	Save the last captured frame to a file. The last captured frame can always be seen in the frame area (ALT + 3).
	Choose a path to the directory where video files created in Record mode will be stored (ALT + 4).
	Record continuously captured frames to an AVI video file. If no path is selected yet, a window opens in which you can select the directory. An existing file will be overwritten. The recording starts immediately after clicking and stops after clicking again. The icon will remain red while capturing is in progress (ALT + 5).
	Load a reference frame for compare function by selecting an image file (ALT + r).
	Compare a captured frame to the loaded reference frame (ALT + c).
	Open LVDS info window (ALT + 6).



When you save the captured frame (), you can also save the image as **.dat** -file. This will save the raw data directly from the buffer without changing it. However, this only works after recording a single frame!

10.2 Capture Settings

As in chapter [Framegrabber General Settings](#), the Capture Area defines the resolution of the frame to be captured, starting at the specified offset.



Additionally in this tab are several flags to configure the frame area:

Flag	Description
Fit to window	The captured frame is adapted to the entire frame area size. Changing the size of the dialog window also changes the size of the captured frame.
Show new window	The captured frame will be displayed in a new full-screen window. At the bottom of the new window, the window coordinates and RGB data of the pixel where the mouse is located are displayed. You can also use the mouse scroll wheel to zoom into the image until the single RGB pixel values are displayed.
Enable BGR swap	Change from RGB to BGR image format.
Mirror vertical	The captured frame is displayed upside down.
Save on single shot	Saves an image for every single shot.
Enable timestamp	The timestamp of the captured frame is displayed at the bottom of the frame area.
Stop on error	The capturing stops when an error is recognized.
Use dma transfer	Using DMA (Direct Memory Access), the frames can be stored directly in the main memory in order to achieve a faster data transfer and to relieve the processor. (Only supported for G PCle 6222 when using PCIe connection)
Frame rate divider	Divide the frame rate.



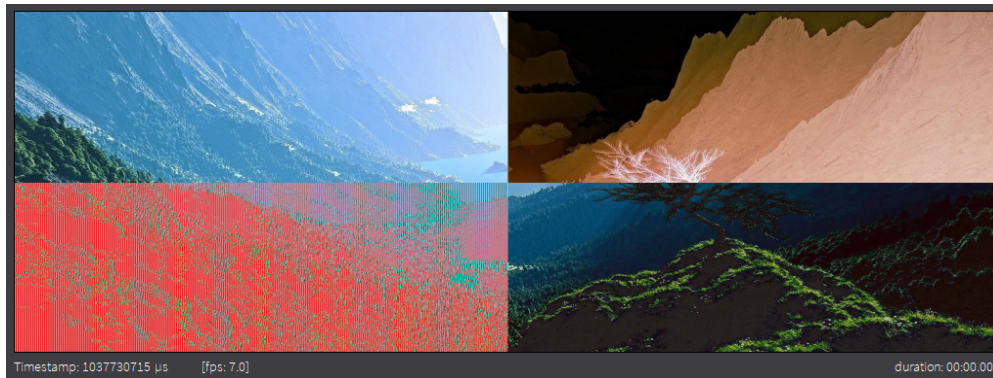
A [Dragon Suite Advanced license](#) is required for the following function.

To capture from a URL, set enable and add the URL name.

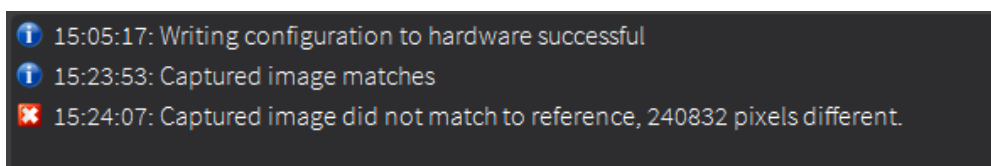
10.3 Compare Settings

The *Dragon Suite* provides the ability to compare a captured frame with a reference frame. This function compares the frames pixel by pixel to see if they match.

The **Compare Area** parameters define the resolution of the frame to be captured, starting at the specified offset. The allowable tolerance range for the RGB values can be set in section **Color Tolerance**. For each color - red, green and blue - a separate tolerance value between 0 and 255 can be set. This value specifies by how many pixels the corresponding color of the captured frame may deviate from the color value of the reference frame. For example, if the color value Red in the reference frame is 132 pixels and the tolerance value is 10 pixels, the pixel value of the captured frame may be between 122 and 142. Otherwise, this pixel is considered to be faulty.



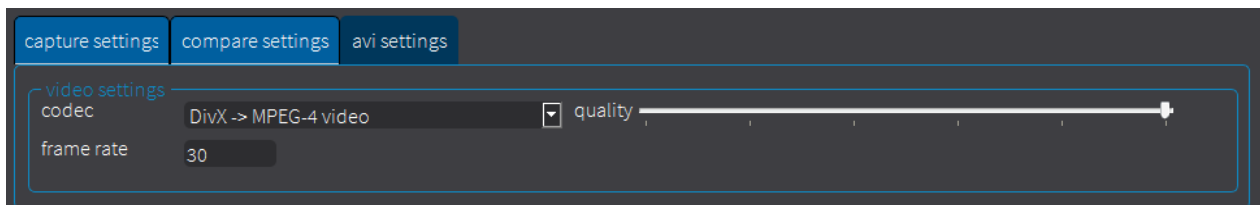
To use the comparison function, a reference frame must be loaded via the [Load Reference](#) button. To compare the frames, use the [Compare](#) button. The loaded reference frame is compared with the last captured frame, which is also displayed in the [Frame Area](#). If no frame has been captured before, the function returns an error. When the Compare command is executed, the differing pixels are displayed in the Frame Area:



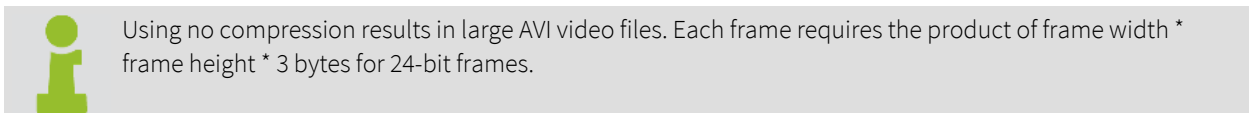
The comparison result is displayed in the [Message Box](#) below. In case the frames do not match, the number of different pixels is displayed.

10.4 Avi Settings

In this segment, the settings for recording videos can be defined.



The selectable **Codec** defines the compression type for the recording AVI video files.



The **Frame rate** sets the maximum rate for capturing frames in continuous mode. This affects both the capture mode and the recording of video files.

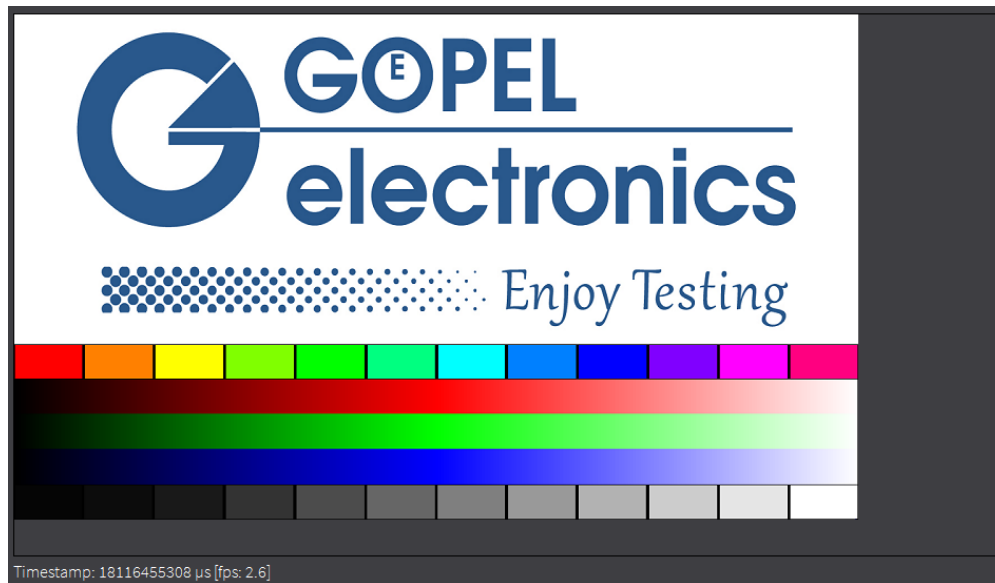
The **Image Quality** slider sets the compression parameter for the selected video compression codec. Depending on the selected codec, the value has different impacts. The lower the number, the higher the compression and the lower the image quality. For more information about the relationship between value and compression of the video, see the documentation for the corresponding codec.

10.5 Raw Data Recording

Recording raw data is only possible in *Dragon Suite Advanced*. This segment is described in chapter [Dragon Suite Advanced](#).

10.6 Frame Area

The frame area always displays the last captured frame.



Below is the timestamp, if desired. It also displays the frame rate at which the image is captured.

11 Sideband Communication

Parallel to the image transfer, the control data can be read in or out via sideband. Depending on the *Media Interface* module the *Video Dragon* supports I²C, SPI and UART. With appropriate commands, the registers of the ICs of both the *Video Dragon* as well as the remote station can be read or written. For some hardware, it is imperative that e.g. a sideband display must receive a "wake-up" command to start communication or receive and process image data.



The sideband communication is only available with enabled sideband activation. The activation can be obtained through GÖPEL electronic [sales department](#).

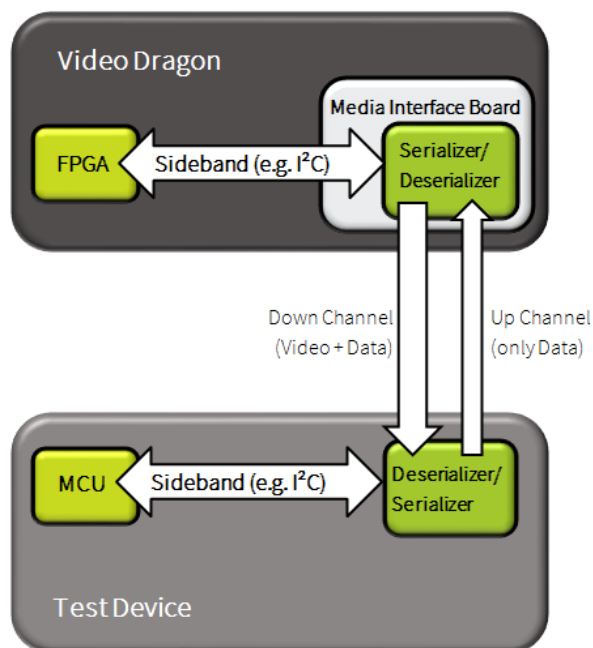


Manual manipulating of the configuration data list needs an extreme good knowledge of the meaning of each register value. This can only be obtained from the data sheets of the serializer and deserializer. Even a single wrong setting of only one register may render the complete configuration invalid and the device inoperative.



An incorrectly configured device can be reset to its default values by a restart.

The following figure shows the sideband communication between the serializer and the deserializer in a schematic way:



The data communication signal (I²C, UART or SPI) from the FPGA is routed to the IC and serialized there. The transmission then takes place via the LVDS data stream in parallel with the video signal. In the receiving IC, the data is deserialized and returned to the outside. The communication is usually bidirectional. In general, the up channel has a lower data rate than the down channel.

11.1 Configuration of Sideband Communication via G-API

To configure and use the sideband communication of the *Video Dragon* the `G_Lvds_Common_Data_*` commands of the [G-API](#) are used. The configuration is divided into setting the data mode and adjusting the parameters.



Please compare the *G-API* Documentation for more information.

11.1.1 Setting the Data Mode

The data mode is set within the command `G_Lvds_Common_Data_Property_SetById` by the property `G_LVDS__COMMON__DATA__PROPERTY_ID__DATA_MODE`. For these, the following options are currently available:

Data mode	Description
NONE	No Data Mode is set
I2C_MASTER_TO_DESERIALIZER	Video Dragon is I ² C Master on deserializer IC
I2C_MASTER_TO_SERIALIZER	Video Dragon is I ² C Master on serializer IC
I2C_SLAVE_TO_DESERIALIZER	Video Dragon is I ² C Slave on deserializer IC
I2C_SLAVE_TO_SERIALIZER	Video Dragon is I ² C Slave on serializer IC
SPI_MASTER_TO_DESERIALIZER	Video Dragon is SPI Master on deserializer IC
SPI_MASTER_TO_SERIALIZER	Video Dragon is SPI Master on serializer IC
SPI_PASSTHROUGH_DUAL	Video Dragon combines data from INAP375R and INAP375T in "dual-mode"
SPI_RECEIVER_DUAL	Video Dragon is connected to INAP375R in "Dual Mode"
SPI_TRANSMITTER_DUAL	Video Dragon is connected to INAP375T in "Dual Mode"
SPI_SLAVE_TO_DESERIALIZER	Video Dragon is SPI Slave on deserializer IC
SPI_SLAVE_TO_SERIALIZER	Video Dragon is SPI Slave on serializer IC
UART_TO_DESERIALIZER	Video Dragon is connected to the UART interface of the deserializer IC (only supported by basicCON 4121)
UART_TO_SERIALIZER	Video Dragon is connected to the UART interface of the serializer IC (only supported by basicCON 4121)

The data mode currently set on the IC can be queried by the command `G_Lvds_Common_Data_Property_GetById`.

11.1.2 Setting the Parameters

Depending on the Data Mode set, certain parameters for the sideband communication can be set. This also happens in the command `G_Lvds_Common_Data_Property_SetById`. The prefix of the property name `G_LVDS__COMMON__DATA__PROPERTY_ID__*DATAMODE*_` indicates for which data mode a property is valid (*DATAMODE* can be `I2C_MASTER`, `I2C_SLAVE`, `SPI_MASTER`, `SPI_SLAVE` or `UART`).

The following table explains the available parameters, with the default values highlighted in bold:

Parameter	Description
I2C_MASTER_BAUDRATE	Baud rate of the I ² C master: 100kHz (0) or 400kHz (1)
I2C_MASTER_CLOCK_STRETCHING	Clock stretching is the ability of the I ² C slave to delay the master-driven clock. For this, the master must read back the clock to respond accordingly. This parameter enables (1) or disables (0) the reaction to the delay of the master.
I2C_MASTER_STOP_NACK	The I ² C master sends an ACK (0) or no ACK (1) from the slave after the last received byte.
I2C_SLAVE_BAUDRATE	Baud rate of the I ² C slave: 100kHz (0) or 400kHz (1).
SPI_MASTER_CPHA	Sets the clock phase of the SPI master data: The data is valid on the first edge (0) after the ChipSelect or on the second edge (1).
SPI_MASTER_CPOL	Sets the clock polarity of the SPI signal: In idle state, the clock is Low (0) or High(1).
SPI_MASTER_CS_MODE	Sets the behavior of the ChipSelect signal during a transfer over several bytes: ChipSelect remains active between the individual bytes (0) or changes briefly to inactive (1).
SPI_MASTER_CLOCK_DIVIDER	Determines the SPI clock frequency based on a basic clock of 25MHz according to the following equation: $\text{Clock} = 25\text{MHz}/(\text{Divider} + 1)$ For example Divider = 0 results in a clock of 25MHz; Divider = 1 results in a clock of 12,5MHz etc.
SPI_SLAVE_CPHA	Sets the clock phase of the SPI slave data: The data is valid on the first edge (0) after the ChipSelect or on the second edge (1).
SPI_SLAVE_CS_IDLE	Sets the minimum time that the ChipSelect must be inactive for the slave to detect the completion of a transfer as a multiple of 20ns. The specified value sets the time to a multiple of 20ns, e.g.: 0 = 127*20ns = 2,54μs 1 = 1*20ns = 20ns 2 = 2*20ns = 40ns usw.
UART_BAUDRATE	The baud rate that is closest to the specified value is set (the real value can be read out). Default: 115200 baud (only supported by basicCON 4121)
UART_PARITY	Defines whether a parity bit is formed and transmitted. 0 = no parity bit , 1 = even parity, 2 = odd parity (only supported by basicCON 4121)

11.2 I²C Master Mode

In the I²C master mode, the *Video Dragon* is the I²C master for the (de-)serializer IC used on the *Media Interface* module. Depending on this, the mode can be set to `I2C_MASTER_TO_DESERIALIZER` or `I2C_MASTER_TO_SERIALIZER`.

11.2.1 Communication

Once the correct data mode and parameters have been set, communication can be made using the firmware commands `G_Lvds_Common_Data_I2cTransfer` or `G_Lvds_Common_Data_I2cTransferEx`. The start command initiates a communication and determines the (successful) completion by polling the status. With this command, any communication on the bus can be initiated independently of any protocol.

11.2.1.1 I²C Transfer on the Bus

The transfer on the I²C bus is bitwise. First, the master sends a start signal followed by the bytes it wants to send to the slave. Then he drives the clock for as many bytes as he wants to read bytes from the slave, which in turn now drives the data line with the answer. Thereafter, a stop signal is sent from the master and communication is completed.

The bytes sent by the master are individually acknowledged by the slave (ACK) or not (NACK). Accordingly, the master acknowledges the bytes from the slave (ACK) or not (NACK). Additionally, the master can provide another byte (in

addition to the first one) that it writes with a start signal. This "Repeated Start" signal does not require a previous stop signal.

11.2.1.2 I²C Transfer by Command

The command to start the transfer has the following parameters:

Parameter	Description
NumberOfTxBytes	The number of bytes to be written by the master. The master drives the clock for exactly as many bytes and places the data on the data line. A maximum of 255 bytes can be transmitted.
NumberOfRxBytes	The number of bytes to be read by the master. The master drives the clock after writing for exactly as many bytes and reads the data from the data line. A maximum of 255 bytes can be received.
SendStartMask	Bytemask indicating the position of an additional (Repeated) start signal (Example: 0x02 = 2nd byte is provided with start signal, 0x04 = 3rd byte, ...).
TxData	The data bytes to be written. A maximum of 255 bytes can be transmitted.

The transfer command has the following return values:

Return Value	Description
AckMask	Byte mask indicating the position of the acknowledged (ACK) bytes (Example: 0x01 = 1st byte acknowledged, 0x02 = 2nd byte, 0x03 = 1st and 2nd byte, ...).
NumberOfRxBytes	The number of bytes read by the master (or written by the slave). A maximum of 255 bytes can be received.
RxData	The read data bytes. A maximum of 255 bytes can be received.

To read a register entry, the transfer function must be executed twice. Once to send the read command and a second time to receive the response. The reason for this is that during I²C communication with a microcontroller, the microcontroller needs a certain amount of time to prepare the response.

```
// data register read, with STOP between Write and read
bool  G_Lvds_DataRegisterReadWithSTOP(G_Lvds_Common_Data_Register_Read_Cmd_t
cmd, G_Lvds_Common_Data_Register_Read_Rsp_t * rsp)
{
    G_Error_t rc;

    G_Lvds_Common_Data_I2cTransferEx_Cmd_t  cmdI2c;
    G_Lvds_Common_Data_I2cTransferEx_Rsp_t  rspI2c;

    memset(&cmdI2c, 0, sizeof(cmdI2c));
    memset(&rspI2c, 0, sizeof(rspI2c));

    cmdI2c.Flags =
G_LVDS__COMMON__DATA__I2C_TRANSFER_EX__CMD_FLAG__NORMAL_READ_WRITE;

    cmdI2c.NumberOfTxBytes = 2;
    cmdI2c.NumberOfRxBytes = 0;
    cmdI2c.TxData[0] = cmd.DeviceAddress<<1;
    cmdI2c.TxData[1] = cmd.RegisterAddress;

    rc = G_Lvds_Common_Data_I2cTransferEx(m_portHandle, &cmdI2c, &rspI2c);
    Sleep(100);
    cmdI2c.NumberOfTxBytes = 1;
    cmdI2c.NumberOfRxBytes = cmd.NumberOfRegisters;
    cmdI2c.TxData[0] = (cmd.DeviceAddress<<1) + 1;
    cmdI2c.TxData[1] = 0;
}
```

```

rc = G_Lvds_Common_Data_I2cTransferEx(m_portHandle, &cmdI2c, &rspI2c);

if(!ErrorHandler(rc, __LINE__, __FILE__, Q_FUNC_INFO))
{
    return false;
}

rsp->NumberOfRegisters = rspI2c.NumberOfRxBytes;

for (int i = 0; i < rsp->NumberOfRegisters; i++)
{
    rsp->Data[i] = rspI2c.RxData[i];
}

return true;
}

```

In case the slave does not respond, the master reads a high for each bit, which results in a 0xFF.

11.2.1.3 I²C Protocol

When transmitting with I²C, the same protocol is used in most cases. This is for writing or reading registers of a slave device. When writing a register, the following data is transferred in this order:

1. Device address (7bit address) and bit (0) for write access
2. Register address (1 byte or more, depending on the device)
3. Register value (1 byte or more, depending on the device)

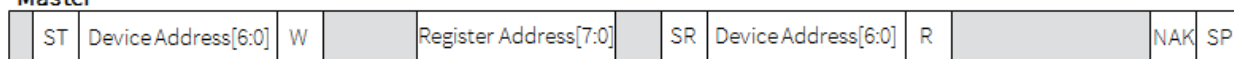
All bytes must be confirmed by the slave. At the beginning and end of the transfer, there is a start or stop signal.

Reading a register is done in two steps. The device and register addresses are written to the slave first, then the device address is written again to initiate reading of the contents:

1. Device address (7bit address) and bit (0) for write access
2. Register address (1 byte or more, depending on the device)
3. Repeated Start signal is sent
4. Device address (7bit address) and bit (1) for read access
5. Register value is sent by the slave (1 byte or more, depending on the device)

All master bytes must be confirmed by the slave. The master confirms all slave bytes (register value) except the last. At the beginning and end of the transfer, there is a start or stop signal. A read access to a 1-byte register is shown as an example in the following graphic:

Master



Slave



11.3 I²C Slave Mode

In the I²C slave mode, the *Video Dragon* is the I²C slave for the (de-)serializer IC used on the *Media Interface* module. Depending on this, the mode can be set to `I2C_SLAVE_TO_DESERIALIZER` or `I2C_SLAVE_TO_SERIALIZER`.

11.3.1 Communication

In I²C slave mode, no independent communication can be triggered because only the master drives the clock of the I²C

bus and also determines the type of access. In slave mode, the *Video Dragon* therefore simulates one or more slave devices in the manner of the protocol, as described in section [I²C Protocol](#) of the chapter [I²C Master Mode](#). The slave then behaves accordingly and responds to requests initiated by the master.

The commands `G_Lvds_Common_Data_I2cSlaveDevice_Define` and `G_Lvds_Common_Data_I2cSlaveRegister_Define` are used to define the simulated devices and their associated registers.

Here, a device must always be defined first before an associated register can be created. Equivalently, with the commands `G_Lvds_Common_Data_I2cSlaveDevice_Delete` and `G_Lvds_Common_Data_I2cSlaveRegister_Delete` devices or registers are removed from the simulation. Deleting a device automatically removes all associated registers.

11.3.1.1 I²C Slave Definition by command

The command for defining a slave device to be simulated has the following parameters:

Parameter	Description
Flags	<p><code>NONE</code> .. no flag set</p> <p><code>REG_ADD_WIDTH_16BIT</code> .. the address width of the device registers is 2 bytes (otherwise only 1 byte).</p> <p><code>DATA_WIDTH_16BIT</code> .. the data width of the device registers is 2 bytes (otherwise only 1 byte).</p> <p><code>ACCESS_WO_ADDRESS</code> .. the master sends only the device address but no register address for access.</p> <p><code>OVERWRITE_DEVICE</code> .. an already defined device with this address will be overwritten (otherwise an error message appears, that the device does not exist).</p> <p><code>DEVICE_ADDRESS_7BIT</code> .. the specified device address is 7bit (otherwise 8).</p>
DeviceAddress	The I ² C address to which the device should respond.
NAckByte	<p>Specifies from how many bytes the slave should respond with a NACK.</p> <p>0 ... every byte is confirmed, 1 ... no confirmation from the 1st byte, ...</p>

The command for defining a slave register to be simulated has the following parameters:

Parameter	Description
Flags	<p><code>NONE</code> .. no flag set</p> <p><code>OVERWRITE_REGISTER</code> .. an already defined register with this address will be overwritten (otherwise an error message appears, that the device does not exist).</p> <p><code>DEVICE_ADDRESS_7BIT</code> .. the specified device address is 7bit (otherwise 8).</p>
DeviceAddress	The I ² C address to which the device should respond.
RegisterAddress	The address of the register that is defined.
RegisterValue	The value of the register that is defined.

11.4 SPI Mode

An SPI bus system typically consists of a master and many slaves. The clock line is the same for all participants and is driven by the master. The data from the master goes via the MOSI line (MasterOutSlaveIn) to the slaves. The slaves provide their data via MISO (MasterInSlaveOut). The selection of the slave with which the master communicates is made via the corresponding SlaveSelect (SS). Further communication parameters are the clock polarity and the clock phase. The clock polarity indicates whether the master clock is high or low active. The clock phase defines on which edge of the clock the data is taken over.

11.5 SPI Master Mode

In the SPI master mode, the *Video Dragon* is the SPI master for the (de-)serializer IC used on the *Media Interface*

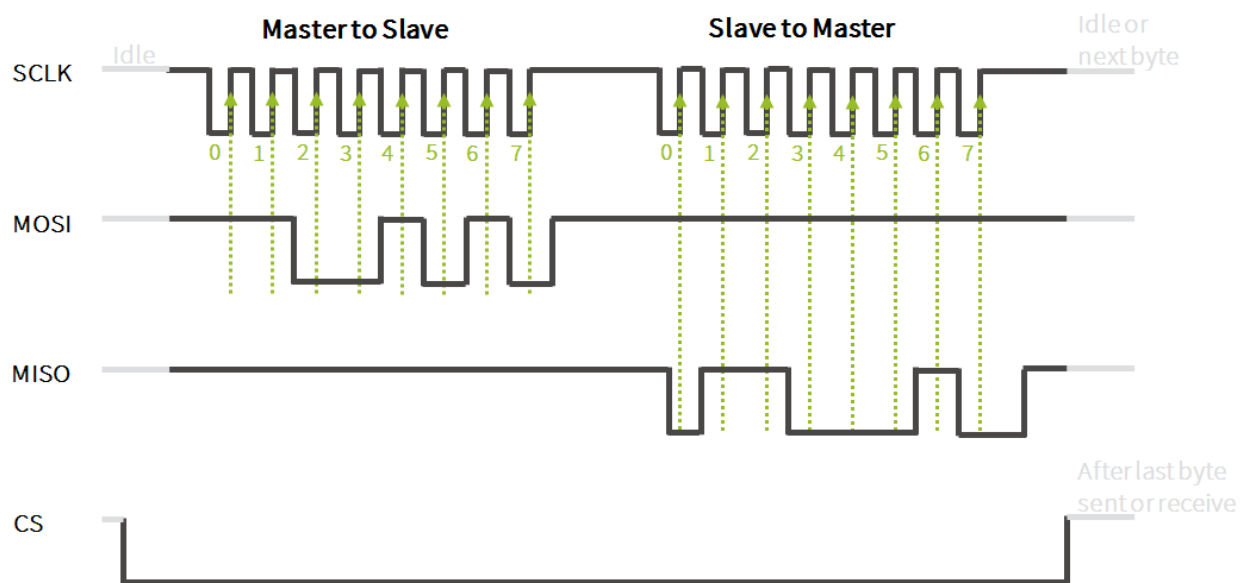
module. Depending on this, the mode can be set to `SPI_MASTER_TO_DESERIALIZER` or `SPI_MASTER_TO_SERIALIZER`.

11.5.1 Communication

Once the correct data mode and parameters have been set, communication can be made using the firmware command `G_Lvds_Common_Data_SpiTransfer`.

11.5.1.1 SPI Transfer on the Bus

The transfer on the SPI bus is bitwise. There are no special control signals. In addition to the clock driven by the master, there is a dedicated data line for the data from the master to the slave and one for the data from the slave to the master. A bidirectional communication can thus take place simultaneously. For addressing a slave device, a special line (ChipSelect) is used. This is pulled by the master to a special level and leads individually to only one slave. The data lines are called **MOSI** – Master Out Slave In and **MISO** – Master In Slave Out. Bidirectional communication is exemplified in the following figure:



There is no special protocol for communication via SPI. The validity of the data (at the first or second edge of the clock) and the idle state of the clock (high or low level) are also not normalized and therefore parameterizable under the parameters **CPHA** and **CPOL** (see chapter [Setting the Parameters](#)). For example, in the figure above, CPHA is set to the second edge and CPOL is set to high.

The other parameters define the data rate and the behavior of the ChipSelect signal between individual bytes.

11.5.1.2 SPI Transfer by Command

The command to start the transfer has the following parameters:

Parameter	Description
NumberOfTxBytes	The number of bytes to be written by the master. The master drives the clock and puts the data on the data line.
NumberOfRxBytes	The number of bytes to be read by the master. The master drives the clock and reads the data from the data line.
TxData	The data bytes to be written.



Reading and writing are always done simultaneously on the bus. Therefore, the total number of bytes for which the master drives the clock is the maximum of the values of the bytes to be written and read. The count of the bytes starts from the beginning of the transfer. This must be taken into account in the number of bytes to be read. For example, if you want to read a response byte after 3 bytes written by the master, you must set the number of bytes to be written to 3 and those of the bytes to be read to 4 (3 + 1). At first 3 bytes are read during writing. These should each have the value 0 due to the prevailing resting level on the MISO line. Subsequently, the payload byte is read with the answer in 4th place.

The transfer command has the following return values:

Return Value	Description
NumberOfRxBytes	The number of bytes read by the master (written by the slave).
RxData	The value of the read data bytes (valid from 0 to <code>NumberOfReadBytes - 1</code>).

11.6 SPI Slave Mode

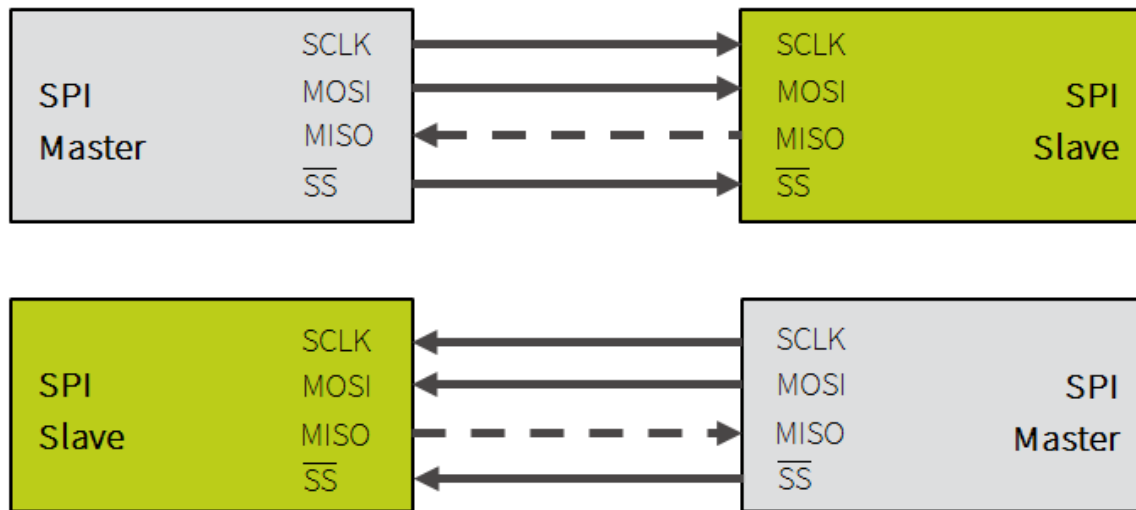
In the SPI slave mode, the *Video Dragon* is the SPI slave for the (de-)serializer IC used on the *Media Interface* module. Depending on this, the mode can be set to `SPI_SLAVE_TO_DESERIALIZER` or `SPI_SLAVE_TO_SERIALIZER`.

11.6.1 Communication

In SPI slave mode, no independent communication can be triggered because only the master drives the clock of the SPI bus and also determines the type of access. Nevertheless, data can be read and written. The used command `G_Lvds_Common_Data_SpiTransfer` is the same as in the [SPI Master Mode](#). The Video Dragon stores data received via the MOSI line in an internal FIFO so that bytes that have already been read can be retrieved. The start command thus does not initiate a transfer, but provides information about whether a successful transfer initiated by the master has taken place and reads out received data. The bytes to be written transferred by parameter are sent to the master from the beginning of the subsequent transfer. As soon as the master starts a new communication and restarts the clock, the data is put on the bus. Again, it should be noted that for a reasonable answer, it may be necessary to prepend some bytes with zero values to the actual user data (see note in chapter [SPI Transfer by Command](#)).

11.7 SPI Dual Mode

SPI Dual Mode is only used with INAP375 boards that communicate via the *APIX* standard. The special feature of these ICs is that a direct bi-directional communication is possible, in which both ICs involved can trigger a transfer. This is possible because two parallel SPI interfaces are used, and only the MOSI data line is used at a time. An answer to sent data thus does not take place in the same transfer and is limited by the cycle length and time of the master. But by the independent spontaneous sending of data on the self-driven parallel bus.



The transmitter mode is used for the INAP375T and the receiver mode for the INAP375R.

11.7.1 Communication

11.7.1.1 SPI Transfer by Command

The commands used are the same as in [SPI Master Mode](#). They differ only in their execution on the bus. Thus, for read bytes, the clock is not driven by the master, but (as in slave mode) the bytes of a previous transfer are fetched from the internal FIFO. If there were no received bytes, the return value of the number of bytes read equals 0. The writing of bytes is thus triggered by the command and takes place accordingly in time thereafter. The reading refers to a transfer that took place before the command was executed.

The sideband always transmits a fixed number of 8 bytes. If fewer bytes are transferred to the command for transmission, they are padded with zero bytes for the transfer to the required number. When reading it should be noted that 8 bytes should be read, even if the number of user data is lower. Any following zeros can be ignored when evaluating the data.

11.8 UART Mode

UART communication is handled differently on *Video Dragon 6222* than on *basicCON 4121*. While there is a Data Mode for the *basicCON 4121*, which is based on the FIFO, the FIFO of the IO interface is directly addressed at the *Video Dragon 6222*. This results in more flexibility in the use.

11.8.1 Communication of basicCON 4121

For UART mode the data mode of the *Video Dragon* has to be set to `UART_TO_DESERIALIZER` or `UART_TO_SERIALIZER`.

There are no master or slave roles in UART communication. Both parties can send data via their TX line, which serves as the RX line for reading. The writing of data therefore triggers a transfer on the TX line. The reading refers to already transmitted data, which are stored in a FIFO when received from the *Video Dragon*. Once the correct data mode and parameters have been set, communication can be made using the firmware command `G_Lvds_Common_Data_UartTransfer`.

11.8.1.1 UART Transfer by Command

The command to start the transfer has the following parameters:

Parameter	Description
Flags	NONE .. no flag set. READ_ALL .. regardless of NumberOfReadBytes, all data is read. The FIFO is completely emptied. WITH_TIMESTAMP .. the receive data is read out with a 64-bit time stamp (time of receipt).
NumberOfTxBytes	The number of bytes to write.
NumberOfRxBytes	The number of bytes to read.
TxDData	The data bytes to be written.

The transfer command has the following return values:

Return Value	Description
NumberOfRxBytes	The number of bytes to read.
RxDData	The value of the read data bytes (valid from 0 to <code>NumberOfReadBytes - 1</code>), or first all timestamps followed by all data bytes if timestamps are enabled.

11.8.2 Communication of Video Dragon 6222

Depending on the *Media Interface* board used, appropriate multifunctional purpose (MFP) pins and SerDes GPIOs must be configured. Due to the high complexity caused by our large product variety, we advise you to get assistance from our [technical support](#).

11.9 Sideband Communication Tool

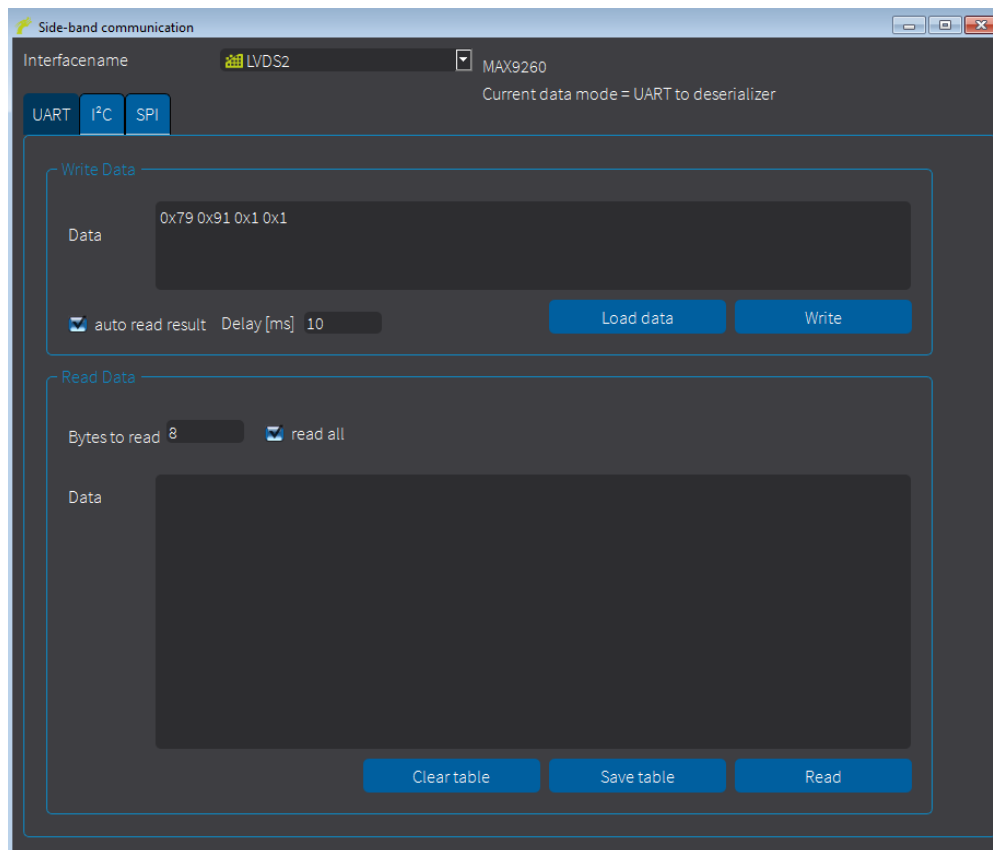
The Sideband Communication tool is for reading or writing register data to a device via UART, I²C or SPI.



The sideband communication is only available with enabled sideband activation. The activation can be obtained through GÖPEL electronic [sales department](#).

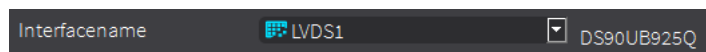


Manual manipulating of the configuration data list needs an extreme good knowledge of the meaning of each register value. This can only be obtained from the data sheets of the serializer and deserializer. Even a single wrong setting of only one register may render the complete configuration invalid and the device inoperative.



First select a [data mode](#) and set it on the sideband settings tab of the device to work correctly with this tool. The current data mode is displayed below the extension board information.

All available interfaces are listed in the drop down menu. The currently selected interface is shown in the text field of the drop down menu. Selecting the interface will automatically open the sideband tab of the defined data mode.



11.9.1 UART

For UART communication, there is no master or slave roles. Both parties can send data over their Tx line, which is used as Rx line on the other. The writing of data triggers a transfer on the Tx line. The reading of data refers to already transmitted data that is stored in the FIFO when the Video Dragon receives it.

UART I2C SPI Current data mode = UART to deserializer

Write Data

Data

0x79 0x91 0x0 0x1;
0x79 0x91 0x1 0x1;
0x79 0x91 0x0 0xf;

☒ auto read result Delay [ms] 10 Load data Write

Read Data

Bytes to read 8 ☒ read all

Data

0xc3,0x80
0xc3,0x90
0xc3,0x80,0x90,0x1f,0x0,0x3,0x29,0xf,0x54,0x30,0xc8,0x12,0x20,0x0,0x0,0x0

Clear table Save table Read

To start the transfer, use the **Write Data** table. Enter the data in the table, following the UART Protocol specified in the serializer / deserializer data sheet. The package values can be entered in decimal or hexadecimal format. Separate the single package values with "," (decimal point) or " " (blank space). To write several package frames at once, use ";" (semicolon) to separate them. Select the **auto read result** checkbox to automatically read the received results after writing and display them in the **Read Data** table. Add a **Delay** in [ms] to define the time delay between two frames (useful only if several frames are written simultaneously). Press the **Write** button to write the data displayed in the table to the device.

The data can also be loaded from a text file via the **Load data** button. The data of each sequences must be defined in the order shown in the example below.

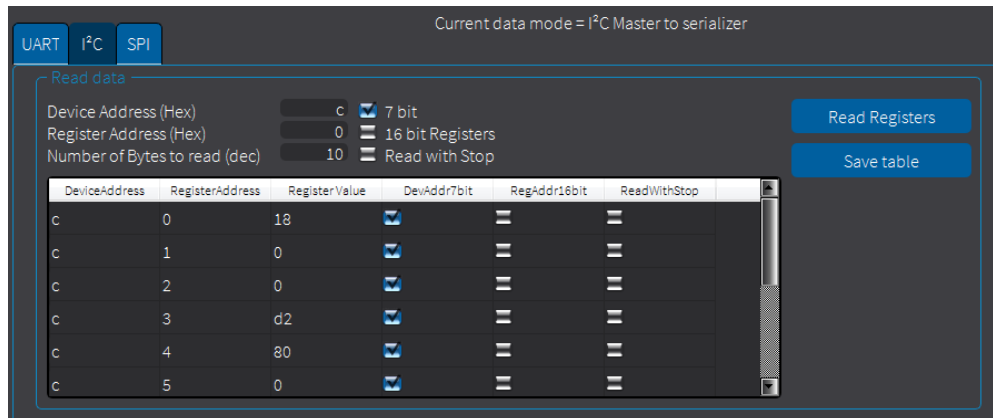
```
1 # Video Dragon Suite UART sequence import file
2 # syncByte, device address, register address, number of bytes to read/write, data to write, ...
3 0x79,0x91,0x0,0x1;
4 0x79,0x91,0x0,0x2;
5 0x79,0x91,0x0,0x3;
6 0x79,0x91,0x0,0x4;
7 0x79,0x91,0x0,0x5;
```

If the auto-read function is not checked, the result data can be read using the **Read** button. It is possible to **read all** data using the check box or to define a specific number of **Bytes to read**. Use the **Clear table** button to clear the table, or the **Save table** button to save the table entries in a text file.

11.9.2 I²C

Transmission on the I²C bus is performed byte by byte. First, the master sends a start signal followed by the bytes it wants to send to the slave. Then the master continues the clock for as many bytes as it would like to read from the slave. The slave drives the data line with the response. Thereafter, the master sends a stop signal and the communication is terminated.

The bytes sent by the master are individually acknowledged by the slave with a confirmation signal (ACK) or not (NACK). Similarly, the master acknowledges the bytes from the slave (ACK) or not (NACK).

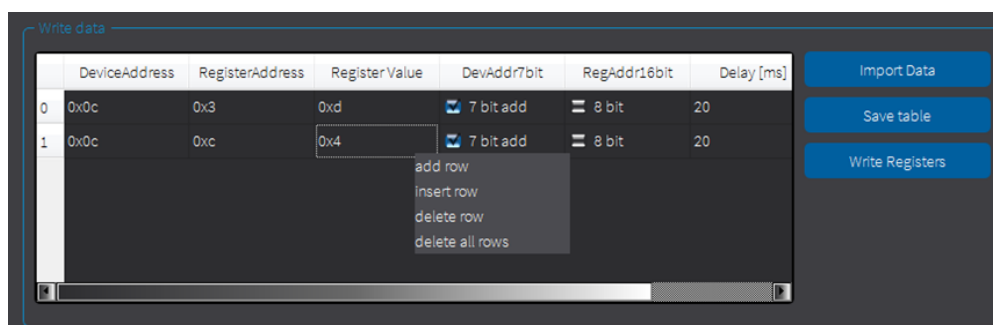


To read register data via I²C, first define the device address (hex value) and the first register address (hex value) from which the registers are to be read. The device addressing can be in 7-bit or 8-bit format, chosen in the **7-bit** check box. If you do not check **7-bit**, the 8-bit format is set. The registers can be read in 8-bit or 16-bit format, chosen in the **16-bit Registers** check box. If you do not check **16-bit**, the 8-bit format is set. This settings can be found in the specifications of the device. In addition, specify the number of bytes you want to read starting at the defined register address (decimal value). The single registers are read and written simultaneously. Select the **Read with Stop** check box to first write all registers before reading them.

Click the **Read Registers** button to read the data and display them in the data list. Save this data with the button **Save table** in a file.

```
4 # DeviceAddress,RegisterAddress,RegisterValue,7-bit Device Address ?, 16-Bit RegisterAddress?, Read with Stop ?, Waittime between lines
5 0x71,0x4,0x0,1,0,0,100
6 0x71,0x4,0x1,1,0,0,10
7 0x71,0x4,0x3,1,0,0,10
8 0x71,0x2,0xffe,1,0,0,20
9 0x71,0x6,0xff,1,0,0,20
```

There are two ways to write data to the device. The first option is to import a list of data from a text file using the **Import Data** button. Select a path to the file and the list will appear in the data table. Possible comments from the text file on the individual register write items are also transferred to the table.



The other way is to add data one by one by hand. Right-click in the table and select **add row** to add a register row at the end of the table. To create a new row at a specific location in the table, select **insert row**. The data can be costumed

by clicking in the list entry and changing the value. The value **Delay** defines the write delay after each register (default is 20) in milliseconds.

In the column **Mask** can be defined which bits of the register should be overwritten. If 0xFF is set for **Mask**, everything is written. If **Mask** = 0x01, then bit 0 is written.

This data table can be saved to a file by using the **Save table** button.

Press the **Write Registers** button to write the table data to the selected device.



In the I²C dialog up to 4 bytes can be sent in one row, as a 32 bit value. The bytes to be written start with the lowest byte.

This means, for example, if you want to write 0x0010 into a 16 bit register, you have to enter 0x1000.



If **use Group Mode** is set, consecutive register addresses are grouped together as one command. This can save time during communication.

So in the example below the register addresses 0x50 and 0x51 are set in one write command.

However, this only happens if no delay is specified (**Delay** = 0) and if **Mask** = 0xFF.

	DeviceAddress	RegisterAddress	RegisterValue	DevAddr7bit	RegAddr16bit	Delay [ms]
0	0x48	0x11	0x00	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	100
1	0x48	0x1	0x02	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	100
2	0x48	0x10	0x31	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	200
3	0x48	0x320	0x2C	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	0
4	0x48	0x50	0x01	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	0
5	0x48	0x51	0x00	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	0
6	0x48	0x150	0x50	<input checked="" type="checkbox"/> 7 bit add	<input checked="" type="checkbox"/> 16 bit	0

Buttons: Import Data, Save table, Write Registers, ☒ use group mode

11.9.3 SPI

As with the I²C, the *Video Dragon* also takes over the master role of the IC on the extension board in SPI master mode. The transmission also takes place byte by byte. In addition to the clock controlled by the master, there is a dedicated line for the data from the master to the slave (MOSI - Master Out Slave In) and one for the data from the slave to the master (MISO - Master In Slave Out). Therefore, bidirectional communication can take place simultaneously. For addressing a slave device, the Chip Select line is used.

There is no specific protocol for SPI communication. The data validity (on the first or second edge of the clock) and the idle state of the clock (high or low level) are also not standardized. Thus they can be parametrized under the parameters CPHA and CPOL.

Current data mode = SPI pass-through dual

Write Data

Data: 0x0 0x80 0xff 0x0 0x0 0x0 0x0

☒ auto read result Delay [ms] 100

Load data Write

Read Data

Bytes to read 8 ☒ read all

Data: 0x0 0x80 0xff 0x0 0x0 0x0 0x0

Clear table Read

To start the transfer, use the **Write Data** table. Enter the data in the table, following the SPI Protocol specified in the serializer / deserializer datasheet. The package values can be entered in decimal or hexadecimal format. Separate the single package values with "," (decimal point) or " " (blank space). The data can also be loaded from a text file via the **Load data** button. To write several package frames at once, use ";" (semicolon) to separate them. Select the **auto read result** check box to automatically read the received results after writing and display them in the **Read Data** table. Add a **Delay** in [ms] to define the time delay between two frames (useful only if several frames are written simultaneously). Press the **Write** button to write the data displayed in the table to the device.

If the auto-read function is not checked, the result data can be read using the **Read** button. It is possible to **read all** data using the check box or to define a specific number of **Bytes to read**. Use the **Clear table** button to clear the table, or the **Save table** button to save the table entries in a text file.

11.9.4 Indigo



Currently, this feature is only supported by *basicCON 4121*.

The Indigo tab is used for sending automotive shell (AShell) messages via *APIX*. Thanks to AShell, the data exchange of the bidirectional *APIX* connection is error-free and secure.

UART I2C SPI Indigo

Address [hex] 1000 Size 32 Bit Data [hex] 00463351 ReadIndex 3

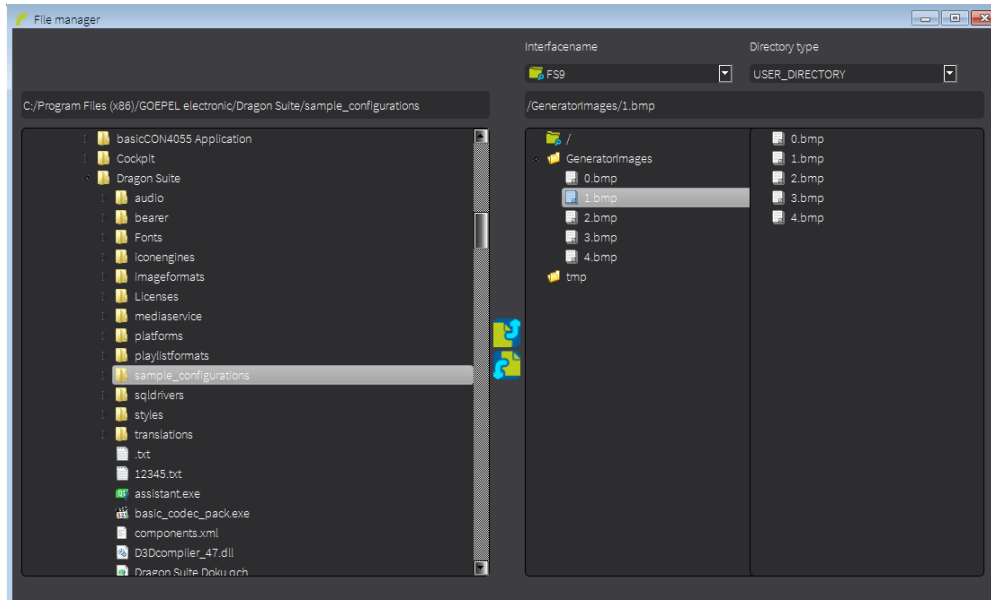
☐ address offset
☐ auto Offset address
☐ restore base address

write read

Insert the address, the data size (8/ 16/ 32 bit) and the data to be written. Click the **Write** button to send a command or the **read** button to read an AShell message.

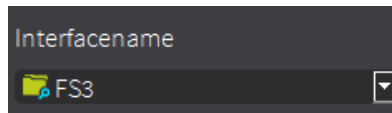
12 File Manager Tool

Amongst other things, the **FS** (File System) software interface allows you to create, copy, delete, execute and search files on the hardware. Thus, a uniform access to the OnBoard File System is possible. The *Dragon Suite* file manager helps organize the File Systems of *GÖPEL electronic* devices.




On the left side of the file manager, the data files of the PC are listed. The right side shows the files of the *GÖPEL electronic* device. On the upper right side there is a drop down menu with all available *GÖPEL electronic* interfaces.

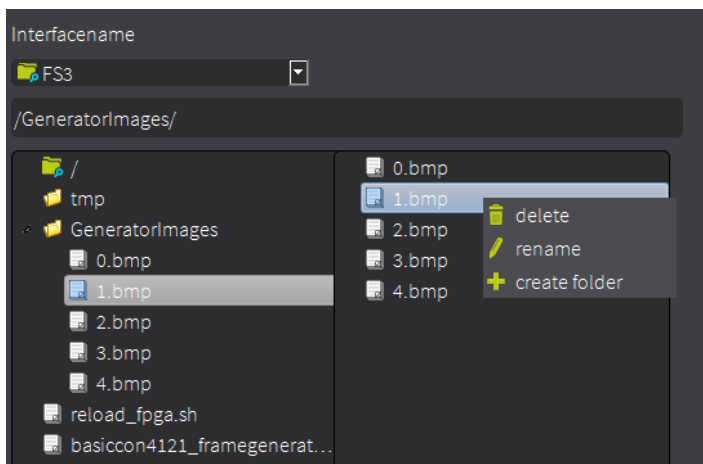
The currently selected interface is displayed in the text field of the drop down menu.



To copy a file from the PC to the device, select the appropriate file on the left and the correct folder

on the right. Press  to add this file to the

device. Use  to download this file from the device to the PC. Right-clicking on a device file (right side) opens a small menu with options for deleting or renaming the selected file or to creating a new folder.



13 IO Tool

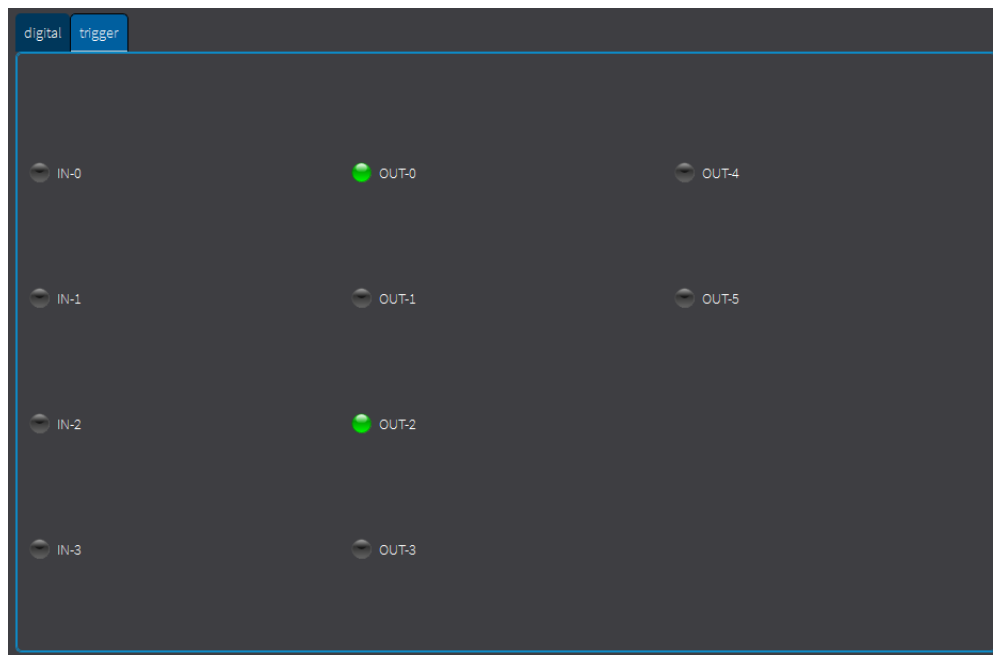
Open the IO dialog by using one of the alternatives illustrated in chapter [Using the GUI](#).

All available IO interfaces are listed in the drop down menu.



13.1 Digital IO

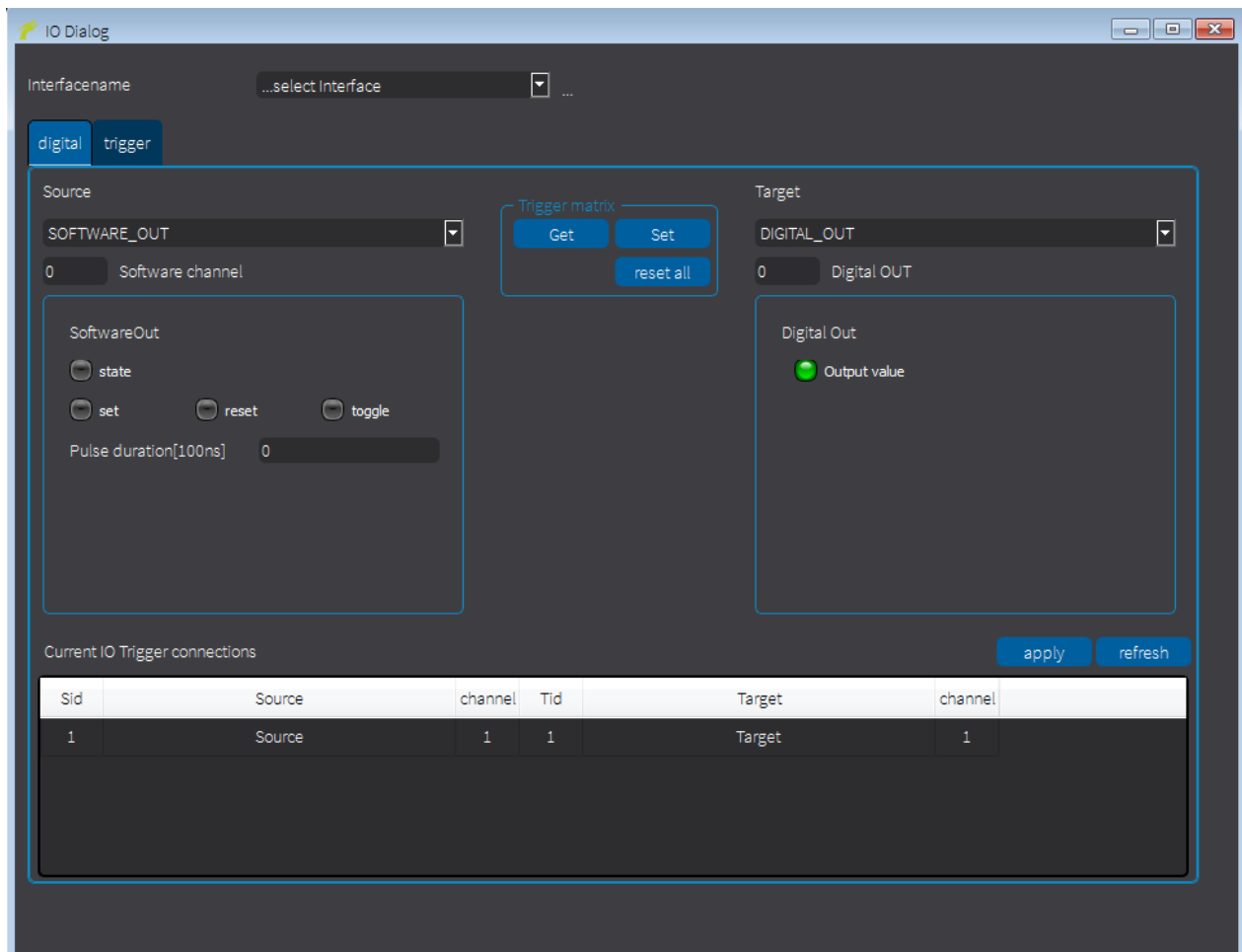
This dialog window shows the digital inputs and outputs of the connected *GÖPEL electronic* hardware. Depending on the connected hardware and its number of digital IOs the tab can display, for example, 6 digital outputs (*basicCON 4121*), 16 (*GÖPEL electronic* relay card) or even a different quantity.



If a digital input or output is set to 1, the control element of this input or output lights up green in the dialog. Is it set to 0, it does not light up. The digital outputs can be set manually in the dialog by setting the control element to 1 (lit green) or 0 (not lit) by clicking on it. This does not apply to the digital inputs, here the control element is only used for status indication.

13.2 Trigger

With the help of the IO tool, the trigger functionality can be controlled by configuring the trigger matrix. It is used to assign a trigger source to a trigger output.



Seen across all *GÖPEL electronic* devices, there are a variety of possible trigger applications. This document only discusses those trigger signals that are supported by the video devices *basicCON 4121* and *Video Dragon 6222*.

The trigger source can be generated internally or taken from an external input. The available sources are listed in the drop down menu.



The following source values are possible:

Source	Description
NO_SOURCE	No source is set.
DIGITAL_IN	The trigger signal is taken from a digital input.
SOFTWARE_OUT	If the trigger source is assigned to a software output, the trigger signal can be generated internally.
TRIGGER_BUS_IN	If the trigger source is assigned to the trigger bus in, trigger signals of the trigger bus line of a PXI, PCI or USB Rack are used for triggering.
LVDS_VIDEO_LOCK	The LVDS video lock signal is used as trigger source.
LVDS_VIDEO_ACTIVE	The LVDS video active signal is used as trigger source.
LVDS_GRABBER_READY	The LVDS <i>Frame Grabber</i> ready signal is used as trigger source.
LVDS_GRABBER_COMPLETE	The LVDS <i>Frame Grabber</i> complete signal is used as trigger source.
LVDS_GRABBER_ERROR	The LVDS <i>Frame Grabber</i> error signal is used as trigger source.
LVDS_0_SER_DES_GPIO	A GPIO signal of the serializer or deserializer board of the first LVDS interface is used as trigger source.
LVDS_1_SER_DES_GPIO	A GPIO signal of the serializer or deserializer board of the second LVDS interface is used as trigger source.
LVDS_2_SER_DES_GPIO	A GPIO signal of the serializer or deserializer board of the third LVDS interface is used as trigger source.
LVDS_3_SER_DES_GPIO	A GPIO signal of the serializer or deserializer board of the fourth LVDS interface is used as trigger source.
LVDS_0_TRIGGER_OUT	A trigger output signal of the first LVDS interface is used as trigger source.
LVDS_1_TRIGGER_OUT	A trigger output signal of the second LVDS interface is used as trigger source.
LVDS_2_TRIGGER_OUT	A trigger output signal of the third LVDS interface is used as trigger source.
LVDS_3_TRIGGER_OUT	A trigger output signal of the fourth LVDS interface is used as trigger source.
UART_TX	The UART output is used as trigger source.
UART_A	The UART analyser is used as trigger source.
INTERNAL_SOURCE	An internal source is used as trigger source.
SPI_M_MOSI	The SPI Master Master Out Slave In is used as trigger source.
SPI_M_SCLK	The SPI Master Serial Clock is used as trigger source.
SPI_M_SS0	The SPI Master Slave Select 0 is used as trigger source.
SPI_M_SS1	The SPI Master Slave Select 1 is used as trigger source.
SPI_M_SS2	The SPI Master Slave Select 2 is used as trigger source.
SPI_S_MISO	The SPI Slave Master In Slave Out is used as trigger source.
LVDS_MI_0_MFP	The LVDS media interface 0 multi function pin is used as a trigger source.



The possible trigger sources depend on the used *Media Interface*.

The trigger signals can be used internally or routed to an output. The available outputs are listed in the drop down menu.

Output

SOFTWARE_IN



The following output values are possible:

Source	Description
TRIGGER_BUS_OUT	If the trigger output is assigned to the trigger bus out, all trigger signals will be routed to the dedicated trigger bus line that feeds all devices of one PXI, PCI or USB bus.
DIGITAL_OUT	The trigger signals are routed to a digital output port. Further configuration options for this parameter are explained below.
SOFTWARE_IN	If the trigger output is assigned to a software input, the trigger signal can be used by the communication interfaces of the device.
LVDS_GRABBER__START	The LVDS Grabber device will start the capture operation when the trigger source is active.
LVDS_GRABBER__STOP	The LVDS Grabber device will stop the capture operation when the trigger source is active.
LVDS_0_SER_DES_GPIO	The trigger signals are routed to a serializer/ deserializer GPIO of the first LVDS interface.
LVDS_1_SER_DES_GPIO	The trigger signals are routed to a serializer/ deserializer GPIO of the second LVDS interface.
LVDS_2_SER_DES_GPIO	The trigger signals are routed to a serializer/ deserializer GPIO of the third LVDS interface.
LVDS_3_SER_DES_GPIO	The trigger signals are routed to a serializer/ deserializer GPIO of the fourth LVDS interface.
LVDS_0_TRIGGER_IN	The trigger signals are routed to the trigger input of the first LVDS interface.
LVDS_1_TRIGGER_IN	The trigger signals are routed to the trigger input of the second LVDS interface.
LVDS_2_TRIGGER_IN	The trigger signals are routed to the trigger input of the third LVDS interface.
LVDS_3_TRIGGER_IN	The trigger signals are routed to the trigger input of the fourth LVDS interface.
UART_RX	The trigger signals are routed to the UART input.
SPI_M_MISO	The trigger signals are routed to SPI Master In Slave Out.
SPI_S_MOSI	The trigger signals are routed to SPI Master Out Slave In.
SPI_S_SCLK	The trigger signals are routed to SPI Slave Serial Clock.
SPI_A_MISO	The trigger signals are routed to SPI Analyzer Master In Slave Out.
SPI_A_MOSI	The trigger signals are routed to SPI Analyzer Master Out Slave In.
SPI_A_SCLK	The trigger signals are routed to SPI Analyzer Serial Clock.
SPI_A_SS0	The trigger signals are routed to SPI Slave Select 0.
SPI_A_SS1	The trigger signals are routed to SPI Slave Select 1.
SPI_A_SS2	The trigger signals are routed to SPI Slave Select 2.
LVDS_MI_0_MFP	The trigger signals are routed to the LVDS media interface 0 multi function pin.
UART_A	The trigger signals are routed to the UART analyser.



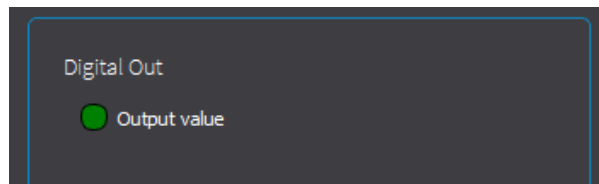
The possible trigger outputs depend on the used *Media Interface*.

Depending on the source or output value, another selection field automatically appears under the dropdown lists. In most cases, this is just the input field for a channel. Enter the necessary **source** or **target channel** here. The count starts at 0. The number of channels depends on your device.

0

Source channel

If, for example, **DIGITAL_OUT** is selected as the output, a configuration field opens. Here the digital output value can be set to 0 or 1 (green).



In the middle of the dialog window there are three buttons:

Button	Description
Get	Load the current IO setting values of the selected interface.
Set	Overwrite all current IO settings on the device with the settings displayed on the tabs of the window.
reset all	Reset all IO settings of the selected interface.



The trigger functionality of the devices offers a variety of configuration options. The software's IO tool offers only a subset of this options. What options are also possible, can be read in the manual, in order to realize this in own applications (if necessary).

The lower part of the dialog window contains a table with the currently set **IO Trigger Connections**. The channel entries can be changed manually by clicking on the desired field and editing the value. Use the **Apply** button to set the values. Click the **Refresh** button to load the actual settings.

Current IO Trigger connections						apply	refresh
Sid	Source	channel	Tid	Target	channel		
7	G_IO_TRIGGER_SOURCE_TYPE_LVDS_VIDEO_LOCK	1	2	G_IO_TRIGGER_OUTPUT_TYPE_DIGITAL_OUT	1		
9	G_IO_TRIGGER_SOURCE_TYPE_LVDS_GRABBER_READY	1	5	G_IO_TRIGGER_OUTPUT_TYPE_LVDS_GRABBER_START	1		
11	G_IO_TRIGGER_SOURCE_TYPE_LVDS_GRABBER_ERROR	1	6	G_IO_TRIGGER_OUTPUT_TYPE_LVDS_GRABBER_STOP	1		

By using the right click, one or all entries can be deleted from the list. This action also deletes the routed trigger!

In addition, the table entry can be copied to the clipboard as a [script command line](#) by right clicking.



The IO Trigger connection table can also be found and edited in the Settings Window of the respective hardware.

13.2.1 SerDes GPIO



Manual manipulating of the GPIO configuration needs an extreme good knowledge of the meaning of each GPIO. Please contact the *Göpel electronic* [Support](#) to get help for this functionality.



An incorrectly configured device can be reset to its default values by a restart.

The GPIOs of the serializers and deserializers can be used for different configurations, depending on the *Media Interface* board.

The SerDes GPIOs of the FPGA are directly connected to the serializers and deserializers.



The SerDes GPIOs of the FPGA can be defined as input or output (push / pull or open drain), in this case seen from the FPGA.

If the two GPIOs to be configured are to be connected to each other, this must be done via IO triggering.

If **SER_DES_GPIO** is selected as trigger source or output, a larger configuration field opens.

The GPIOs can be used to trigger specific actions. In the configuration field you can set the input/ output controller pin configuration of the serializer/ deserializer board. For the **Output Type** of the SerDes GPIO the following values are possible:

- input only
- push/ pull output mode
- open drain output mode

In addition, the configuration field contains two digital display segments: **Input Value** and **Output Value**. **Input Value** shows only the on/ off value and can not be changed manually. If the value is 1, the item turns green. The **Output Value** can only be changed if the output type is "push/ pull output mode" or "open drain output mode". The digital value can be changed manually to on (green) or off.

With the button "Set" the SerDes GPIO configuration is set. If the output value has been set to 1, the input value now also goes to 1 (and turns green), since output and input are internally connected.

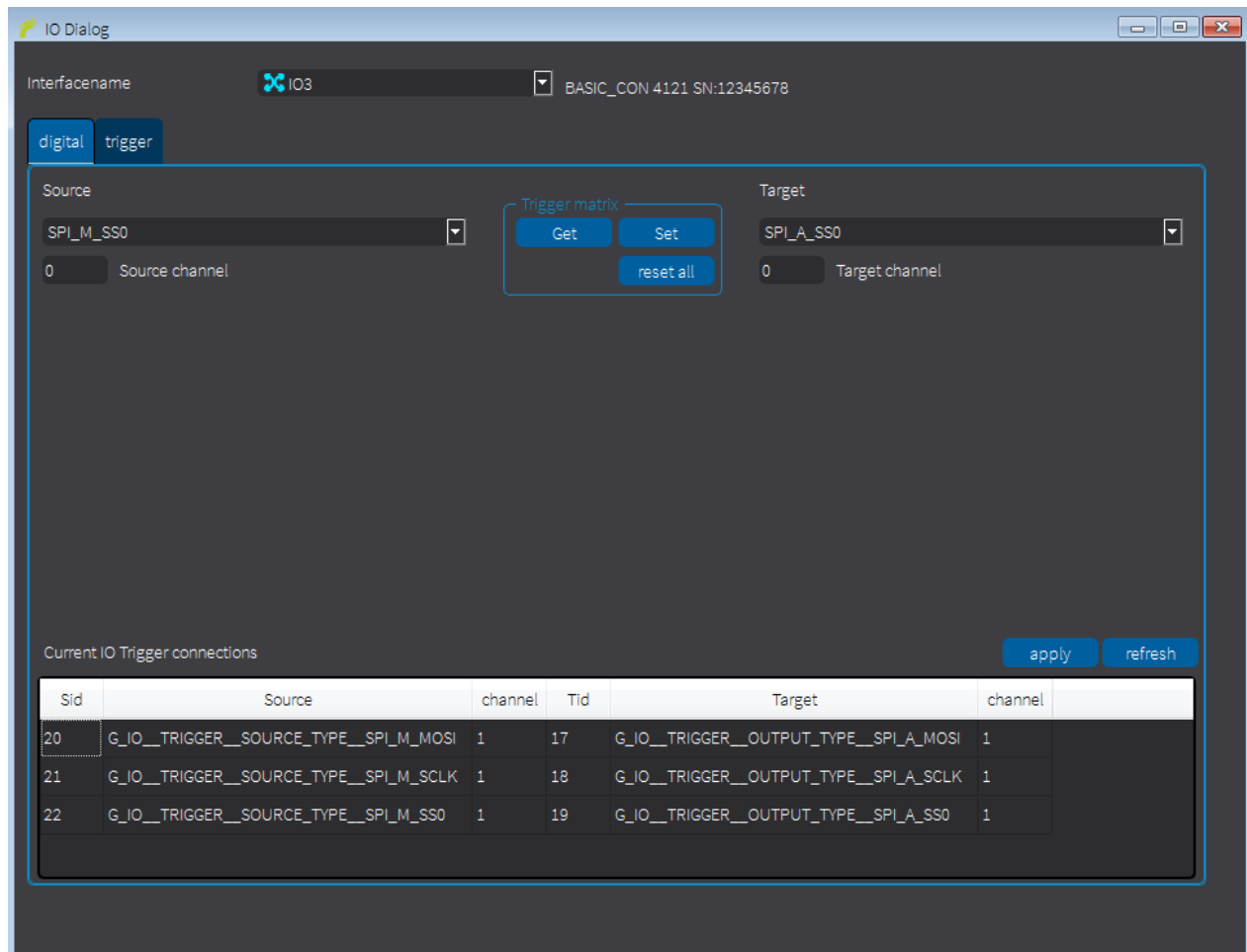
13.2.2 Examples

Example 1: Routing Video Lock to Digital Output

The hardware used is an LVDS frame grabber. The trigger source is **LVDS_VIDEO_LOCK** on channel 0. As output the digital output 2 is defined. With "Set" this configuration is set. As soon as a lock signal occurs, the digital output 2 is activated.

Example 2: Routing SPI Analyzer to SPI Master for SPI Monitoring

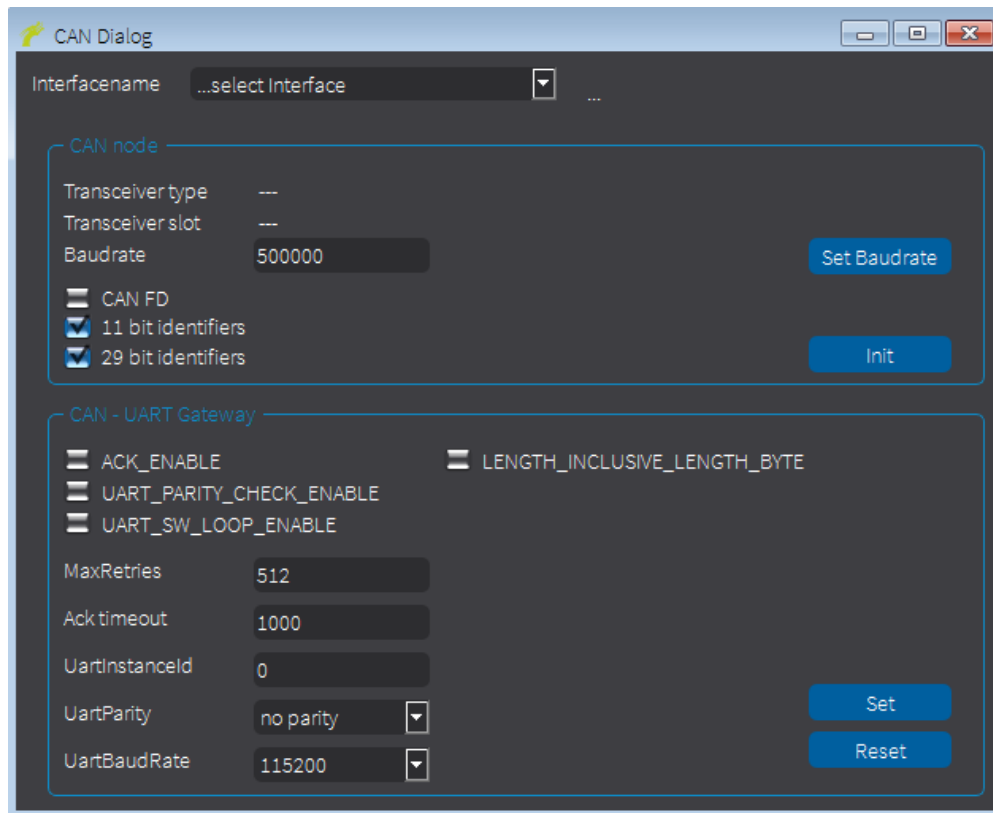
To monitor SPI communication of the video device, the SPI Analyzer must be routed to the SPI Master signals. Therefore route the targets `SPI_A_MOSI`, `SPI_A_SCLK` and `SPI_A_SS0` to the sources `SPI_M_MOSI`, `SPI_M_SCLK` and `SPI_M_SS0`. Now the SPI communication can be monitored in the [Monitor Dialog](#).



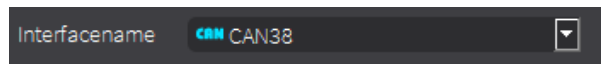
The SPI monitor is only available in [Dragon Suite Advanced](#).

14 CAN Tool

Open the CAN dialog by using one of the alternatives illustrated in chapter [Using the GUI](#). In the upper tab the CAN node can be initialized. The lower tab is for defining a CAN - UART Gateway.



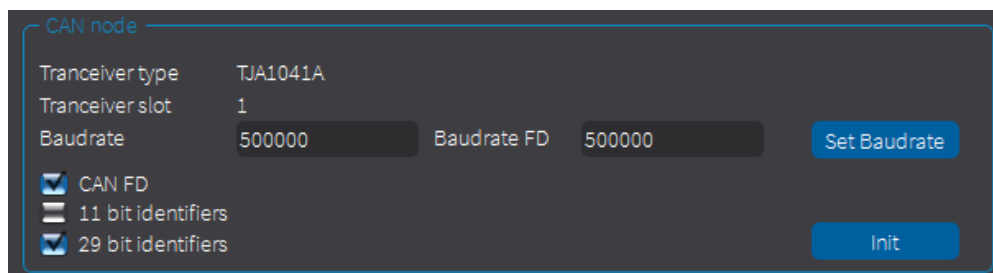
All available CAN interfaces are listed in the drop down menu.



14.1 CAN Node

After selecting the interface, the current values are displayed in the tab. **Transceiver type** and **Transceiver slot** are immutable values.

Enter a baud rate value (in baud, e.g. 500000 for 500kBaund) and click the "SetBaudrate"-Button to set the baud rate of the CAN node. CAN FD and Extended Identifiers can be enabled or disabled by setting the associated flags and clicking the "Init"-Button. When using CAN FD, the CAN FD baud rate can also be set.



14.2 CAN - UART Gateway

The CAN - UART Gateway can be set to configure a gateway for gating CAN messages to UART and vice versa. Some test devices can not communicate directly via CAN, e.g. to receive a wake-up message. The *Video Dragon* offers the possibility to receive CAN messages via the CAN interface and to route them to a UART signal. The UART signal can then be received and processed by the test device via the LVDS stream.



A configuration of the [IO interface](#) is necessary to use the CAN - UART Gateway (see [example](#) below).



For using the CAN - UART Gateway the [Data Mode](#) of the device must be set to "UART to serializer" or "UART to deserializer".

CAN - UART Gateway

☐ ACK_ENABLE ☐ LENGTH_INCLUSIVE_LENGTH_BYTE

☐ UART_PARITY_CHECK_ENABLE

☐ UART_SW_LOOP_ENABLE

MaxRetries: 512

Ack timeout: 1000

UartInstanceld: 0

UartParity: no parity

UartBaudRate: 115200

Set

Reset

The Gateway tab contains several flags for configuration:

Flag	Description
ACK_ENABLE	Enables the acknowledge handling. (Not supported yet)
UART_PARITY_CHECK_ENABLE	Enables parity checking in received UART frames. Frames with a false parity bit are disregarded.
UART_SW_LOOP_ENABLE	If the flag is set, the received CAN messages are ignored and the received UART messages are not gated to CAN. Received UART messages are looped back to UART.
LENGTH_INCLUSIVE_LENGTH_BYTE	If the flag is set, this length byte also contains the length byte itself in addition to the number of bytes after the length byte. If the flag is not set, it only contains the number of bytes after the length byte.

The parameter **MaxRetries** specifies the maximum number of TX repeats if no acknowledge is received. If the specified maximum is reached, the frame is discarded. **AckTimeout** defines the time (in μ s) to wait for an acknowledge before repeating the message. In **UartInstanceld** a UART instance has to be defined, starting with 0. The **UART parity** can be set to even, odd or none. Additionally the **UART baud rate** needs to be specified. By clicking the "Set"-Button the parameters are written to the device. The "Reset"-Button resets the UART Gateway.

14.2.1 Example

This chapter shows a short example of the CAN - UART Gateway configuration. For this purpose, in addition to the CAN configuration, the IO trigger interface must also be set. For this example, a *g PCIe 6222* is used. The CAN configuration is

shown in the following picture:

The image shows two configuration panels. The top panel, titled 'CAN node', contains the following settings: Tranceiver type is 'TJA1041A', Tranceiver slot is '1', Baudrate is '500000', and Baudrate FD is '1000000'. There are 'Set Baudrate' and 'Init' buttons. Checkboxes for 'CAN FD', '11 bit identifiers', and '29 bit identifiers' are all checked. The bottom panel, titled 'CAN - UART Gateway', contains settings for 'ACK_ENABLE', 'UART_PARITY_CHECK_ENABLE', and 'UART_SW_LOOP_ENABLE' (all disabled). It also has 'MaxRetries' (512), 'Ack timeout' (1000), 'UartInstanceld' (0), 'UartParity' (even parity), and 'UartBaudRate' (1000000). There are 'Set' and 'Reset' buttons.

In addition to the CAN configuration, it must of course be specified how the CAN signal is routed to the UART output. In our example the GPIO 0 is used for the CAN signal (as source). The output is the `UART_Rx` signal.

The image shows a 'trigger' configuration window. It has three main sections: 'Source', 'Trigger matrix', and 'Output'. The 'Source' section has a dropdown menu set to 'LVDS_0_SER_DES_GPIO' and a button '0 SerDes GPIO'. Below it is a 'SerDes GPIO locPinConfig' section with 'Output type' set to 'open drain output mode' and radio buttons for 'Input value' and 'Output value' (the latter is selected). There is a 'Set' button. The 'Trigger matrix' section has 'Get', 'Set', and 'reset all' buttons. The 'Output' section has a dropdown menu set to 'UART_RX' and a button '1 UART instance'.

Equivalently, the `UART_Tx` signal is set as source and routed to GPIO 3. Thus, a communication can take place in both directions.

trigger

Source

UART_TX

1

UART instance

Trigger matrix

Get

Set

reset all

Output

LVDS_0_SER_DES_GPIO

3

SerDes GPIO

SerDes Gpio_locPinConfig OUT

...

Output type

☐ Input value

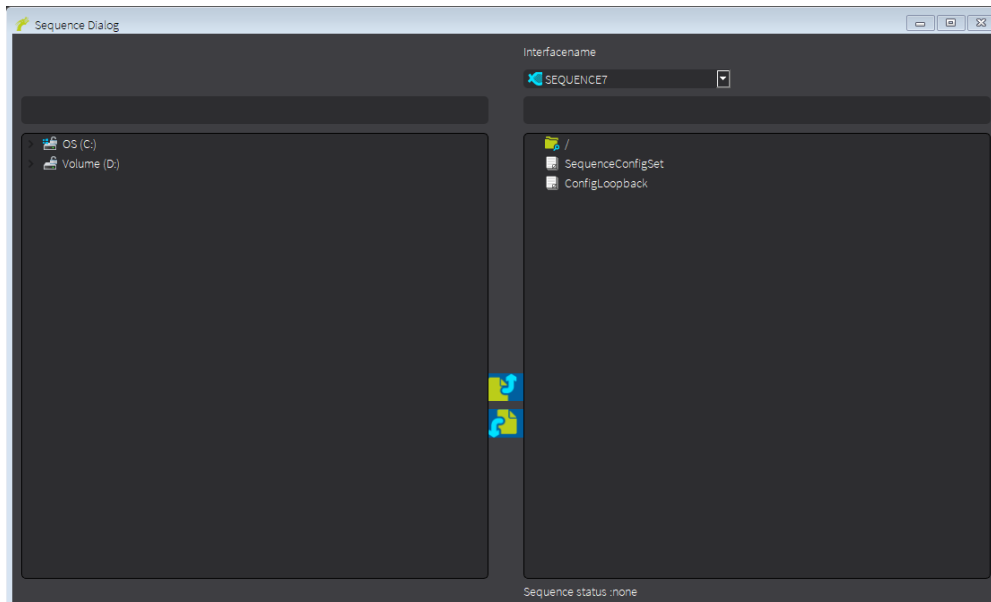
☐ Output value

Set

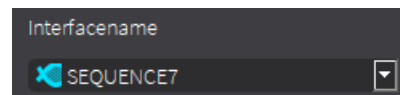
15 Sequence Interface

Open the Sequence Interface by using one of the alternatives illustrated in chapter [Using the GUI](#).

Via the Sequence interface there is the possibility to record the execution of *G API* functions on a device and to save them as a sequence. This sequence can be played repeatedly or set as the autostart sequence of the device.











All available Sequence interfaces are listed in the drop down menu.



In general, the dialog is structured like the [File Manager](#). However, the sequence files are not stored in the File Manager, but have their own storage space. The files can be copied from the PC to the device and vice versa as in the File Manager.

On the right side of the dialog window, the sequences on the device are listed. By right-clicking in the window, various functions can be executed:

Icon	Description
 playback sequence	Execute the sequence stored on the device (right click on the sequence file).
 record sequence	Start recording a sequence. All <i>G API</i> functions now executed will be recorded. Waiting times between functions caused by manual input are reduced to a minimum. The sequence is saved as a file in the Sequence Interface.
 stop sequence	Stop recording a sequence.
 enable autostart	The selected sequence is placed in the autostart of the device. At each restart the sequence is executed after booting the device.
 disable autostart	After booting the device, no autostart sequence is executed.
 delete	Delete the sequence file.
 rename	Rename the sequence file.
 refresh	Refresh the file list.

16 Command Line Interface

The *Dragon Suite* provides the ability to work with the command line interface. Open the GUI-less *Dragon Suite* as shown in the example below.

```
Windows-Befehlsprozessor
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Windows\System32>D:\LVDS_Suite\Qt_Sources\Dragon_Suite_Installer\packages\com.goepel.dragonsuite\data\Dragon_Suite.exe -?
***** Wellcome to Dragon Suite Guiless *****
-?                               :show this help
-Interface:[NAME]               :the interface which executes the feature, e.g.-Interface:LVDS1
-Config:[PATH]                  :configure the device by xml-file, e.g. -Interface:LVDS1 -Config:config.xml
-Info                            :show info about the device (LVDS-Info), e.g.-Interface:LVDS1 -Info

-Capture                         :start capturing and show the Image, e.g.-Interface:LVDS1 -Capture
-Colorformat                    :input image format RGB888, YUV422 or RAW12 , e.g.-Interface:LVDS1 -Capture -Colorformat:RAW12
-CaptureArea                    :set the capture area x,y,w,h only in combination with capture, e.g.-Interface:LVDS1 -Capture -CaptureArea:0,0,800,480
-MirrorV                        :flip the image vertical, only in combination with capture, e.g.-Interface:LVDS1 -Capture -CaptureArea:0,0,800,480 -MirrorV
-LVDS_Channel                    :select the LVDS channel as capture source, e.g.-Interface:LVDS1 -Capture -LVDS_Channel:1

-WriteI2C_FromFile              :write data over I2C side band from file, e.g.-Interface:LVDS1 -Capture -WriteI2C_FromFile:I2C_Sequence.txt

C:\Windows\System32>
```

17 Dragon Suite Advanced

Some *Dragon Suite* features are only available for *Dragon Suite Advanced*. The features explained in this chapter can only be used in advanced mode.

Category	Feature	BASIC	ADVANCED	Comment
Generator Config	Basic configuration	X	X	
	SerDes + MiMfP GPIO config		X	
Grabber Config	Basic configuration	X	X	
	SerDes + MiMfP GPIO config		X	
Generator Dialog	RGB generating	X	X	
	YUV, RAW generating		X	coming soon
	Generating on multiple channels simultaneously		X	coming soon
	Advanced pattern generator		X	
	Video output from file to PC or desktop (mirror)	X	X	
	Video output of recorded RAW data		X	coming soon
Grabber Dialog	Basic capturing	X	X	
	Advanced color format conversion (Bayer, Grey 8+12 bit, YUYV-UYYV 8+10 bit)		X	
	YUV, with limited FR		X	
	RAW data recording		X	
	RAW data to AVI converter		X	
	Grab on multiple channels simultaneously (raw)		X	
Sideband Dialog	UART, I2C, SPI	X	X	
	Send/Receive Ashell Messages		X	
	MII / Ethernet		X	coming soon
Monitoring	CAN Monitor		X	
	Sideband SPI Monitor		X	
	Sideband I ² C Monitor		X	coming soon
	Sideband MII Monitor		X	coming soon
	Sideband UART Monitor		X	
FS Interface	File System interface	X	X	
IO Interface	r/w digital IOs	X	X	
	Trigger settings	X	X	
Sequence Interface	Sequence Interface		X	
Script Interface	Script Interface		X	
CAN Interface	Basic message functions		X	via Script Interface
Ethernet	Send/ Receive UDP frames per Fifo		X	via Script Interface

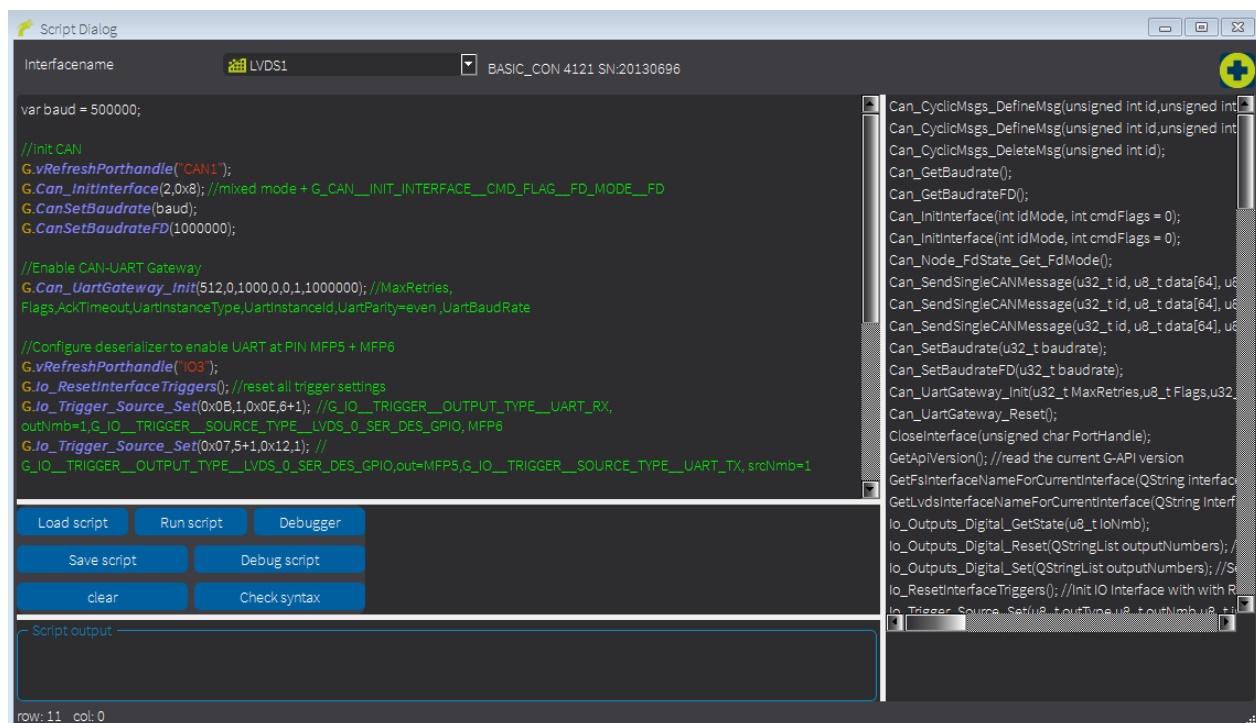
The Advanced version of the *Dragon Suite* is only available through paid licenses, which you can purchase from our [sales team](#). The activation is done via the hardware activation of your *Video Dragon*. If a device with this activation is detected when the software starts, it automatically opens in Advanced Mode. The activation can be done at any time, even after the purchase of the hardware. A free trial license can also be requested from the sales department.

17.1 Script Interface

The Script Interface is used to run Java (ECMA) scripts to automatically control the functions of the *Video Dragon* Hardware. Among other things, this includes configuring the *Video Dragon*, generating or capturing frames to file, switching IOs, and communicating via sideband. This means that all work steps can be completed with just a few clicks. With manual single steps the effort is considerably higher. In addition, all the manual steps must be carried out again and again with each restart or hardware change. The Script Interface makes work easier not only in development, but also in the manufacturing process.

For example, the following sequence can be performed with only one script:

- Configuring the LVDS interface
- Configuring register settings
- Configuring the *Frame Grabber* CAN - UART Gateway and SerDes GPIOs
- Sending I²C commands from the *Frame Grabber* to a connected camera
- Output I²C response strings
- Capturing frames to a file



All available LVDS interfaces are listed in the drop down menu.



The interface to be addressed can be changed in the script with `RefreshPorthandle () ;`.

Thanks to syntax highlighting and autocompletion, creating the script is as easy as possible. Autocompletion is intended for words longer than 2 characters. In addition, you can trigger the autocompletion with the shortcut Ctrl + E.

```

var i = 0;
while(1)
{
    for(i=0; i<255;i++)
    {
        G.LVDS_DisplayColor(1280,640,i,0,255-i);
        G.delay(100);
    }
    for(i=0; i<255;i++)
    {
        G.LVDS_DisplayColor(1280,640,255,i,255-i);
        G.delay(100);
    }
}

```



Enter "G." followed by Ctrl + E to access *Dragon Suite* and *G-API* methods. Use right-click to undo, copy and replace, or to select the entire typing.

Interfacename
LVDS19
GPCIE6222 SN:190858

```

var baud = 500000;

//Init CAN
G.vRefreshPorthandle("CAN1");
G.Can_InitInterface(2,0x8); //mixed mode + G_CAN__INIT_INTERFACE__CMD_FLAG__FD_MODE__FD
G.CanSetBaudrate(baud);
G.CanSetBaudrateFD(1000000);

//Enable CAN-UART Gateway
G.Can_UartGateway_Init(512,0,1000,0,0,1,1000000); //MaxRetries,
Flags,AckTimeout,UartInstanceType,UartInstanceId,UartParity=even,UartBaudRate

G|
vLVDS_sendI2CFromFile(QString path);
vRefreshPorthandle(QString InterfaceName); /*close PortHandle and open a new one*/
vResetDevice(QString Interfacename);
vResetInterface(QString Interfacename);
vSetLVDSChannel(unsigned char channel);
vStartCapturing();
vStopCapturing();

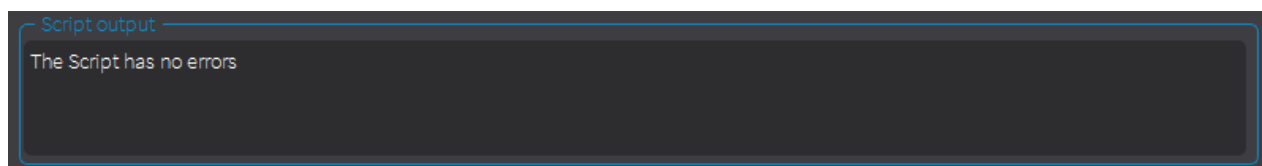
```


Script output


There are several buttons in the Script Dialog window:


Button	Description
Load script	Load the script for the selected interface by importing an external Java (ECMA) script file (*.js).
Save script	Save the current script of the selected interface by exporting them to an external Java (ECMA) script file (*.js).
clear	Deletes the entries in the main window.
Run script	Starts the loaded script on the selected interface.
Debug script	Open the Qt Script Debugger Tool and start debugging of the script.
Check syntax	Check the syntax of the script automatically and display found errors.
Debugger	Open the Qt Script Debugger Tool

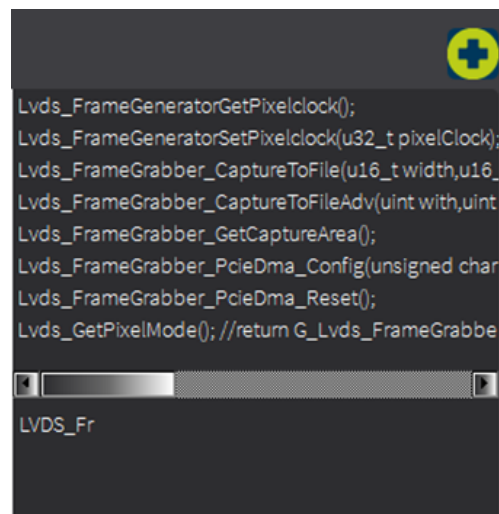
The **Script Output** gives feedback when the script is executed or displays any errors.




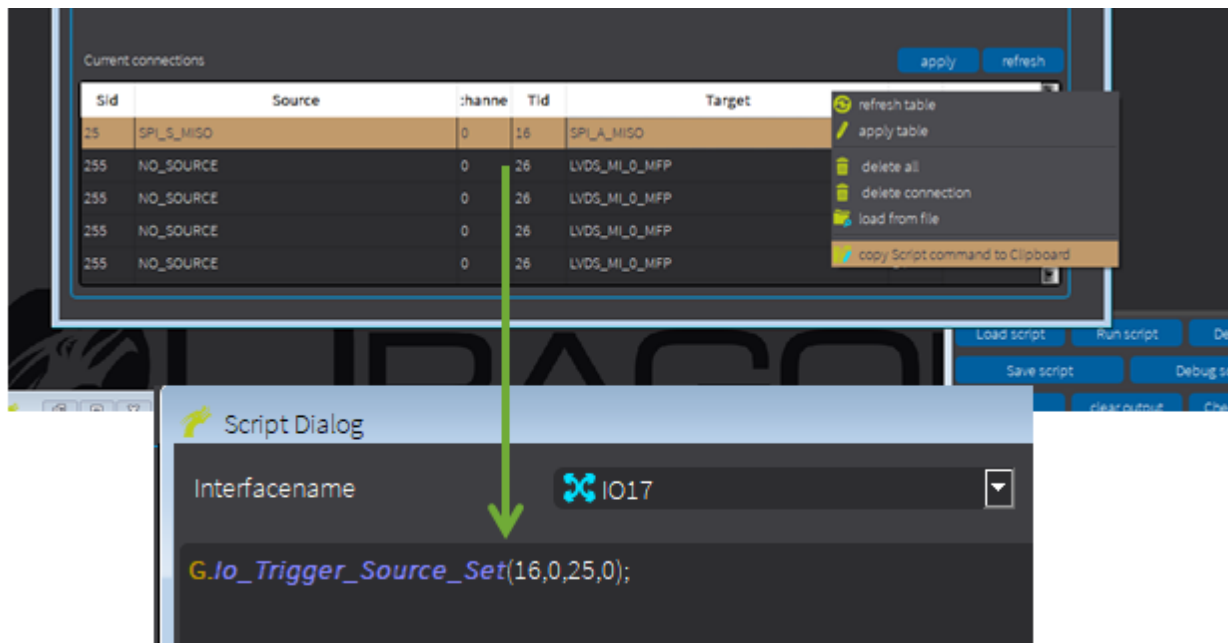
 Below the script output you can find the information in which row and column of the main window the cursor is located. This makes it easier to search for longer scripts.


The functions that can be used can be found in the **Help** window on the right. Open or close it by using the  icon. The search for functions can be simplified by using the lower input field.

 The functions can be dragged and dropped into the main window.




 Script commands can be created directly from various tables in the *Dragon Suite* (such as the IO Trigger or the MiMfp Tab). Right-click on the desired line entry and select **"copy Script command to clipboard"**. Then open the script window, right-click in the editing window and paste the command.

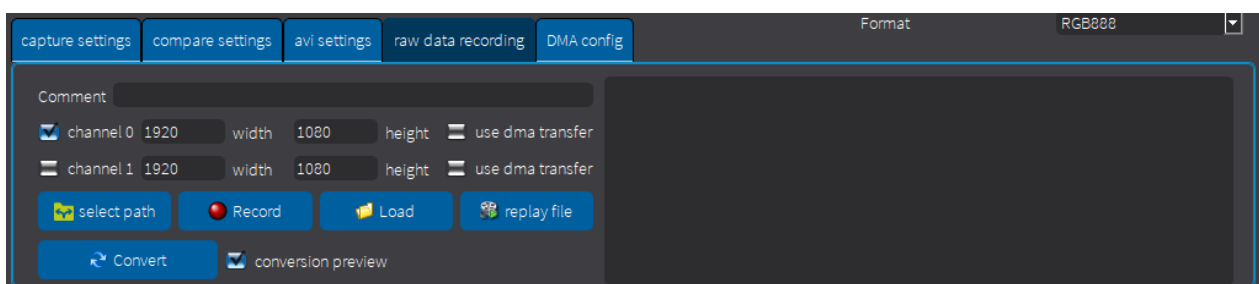


 Some sample scripts can be found in the installation folder of the *Dragon Suite*.

17.2 Raw Data Recording

Raw Data Recording is a *Frame Grabber* feature. This is used to store and play back received raw data.






 This feature is only supported for *Video Dragon 6222*.



Use **Comment** to insert a short note at the beginning of the RAW file. Select channel 1 or 2 and specify the resolution.

Also select whether to use [DMA](#) transfer.

Below are some buttons:

Button	Description
 select path	Select path to the directory where the *.raw files will be stored.
 Record	Start recording the raw data and save it in the path specified before.
 Load	Load saved *.raw file.
 replay file	Replay loaded raw data.
 Convert	Converts raw data to AVI.

Each recorded file is saved with a common file header. The structure is as follows:

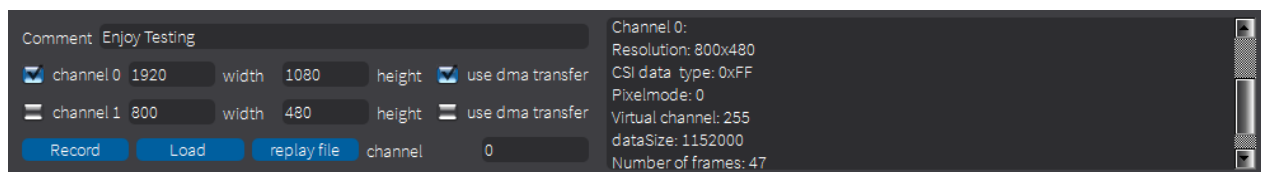
```
typedef struct{
    u32_t version; //use current suite version for header version
    u32_t headerSize;
    u8_t numberOfChannels; //number of LVDS channels to capture and save
    u8_t LVDSChannel[0xFF]; //the real selected LVDS channel
    u8_t pixelmode[0xFF];
    u32_t width[0xFF]; //frame width
    u32_t height[0xFF]; //frame height
    u32_t dataSize[0xFF]; //per channel
    u16_t reserved;
    u8_t CSI2_dataType[0xFF];
    u8_t virtualChannel[0xFF];
    u32_t frameHeaderSize; //for image/frame data
    u32_t framecounter[0xFF];
    u32_t deviceFramecounter[0xFF]; //device frame counter difference while
recording
    qulonglong duration; //total duration in ms
    char comment[0xFF]; //description for the file

    u16_t reserved0;
}RawDataStreamFileHeader_t;
```

The single frames are stored in the following structure:

```
typedef struct{
    u16_t version;
    u16_t rows;
    u16_t cols;
    u8_t pixelmode;
    u8_t CSI2_dataType;
    u8_t virtualChannel;
    u8_t dataChannel; //current channel in file
    u32_t headerSize;
    u32_t dataSize; //size of the video frame in Bytes
    qulonglong timestamp;
    u32_t frameNumber;
    u16_t reserved0;
}RawDataStreamFrameHeader_t;
```

When loading the file you will find the header information in the right window.



To replay the file it is not necessary that the Frame Grabber is configured. However, it may be necessary to make adjustments in the [Capture Settings](#) (Pixel Mode, Color Data Format). Select the desired channel before replaying (To

the right of the **Replay** button). Pressing the **Replay** button will play the loaded file once completely. During replay the process can be stopped with the same button.

17.3 Monitor Dialog

With the Monitor feature CAN and SPI signals can be monitored.



Not every media interface supports all sideband functions mentioned here. Furthermore, sideband and/or CAN must be activated for the device.

All available LVDS interfaces are listed in the drop down menu.

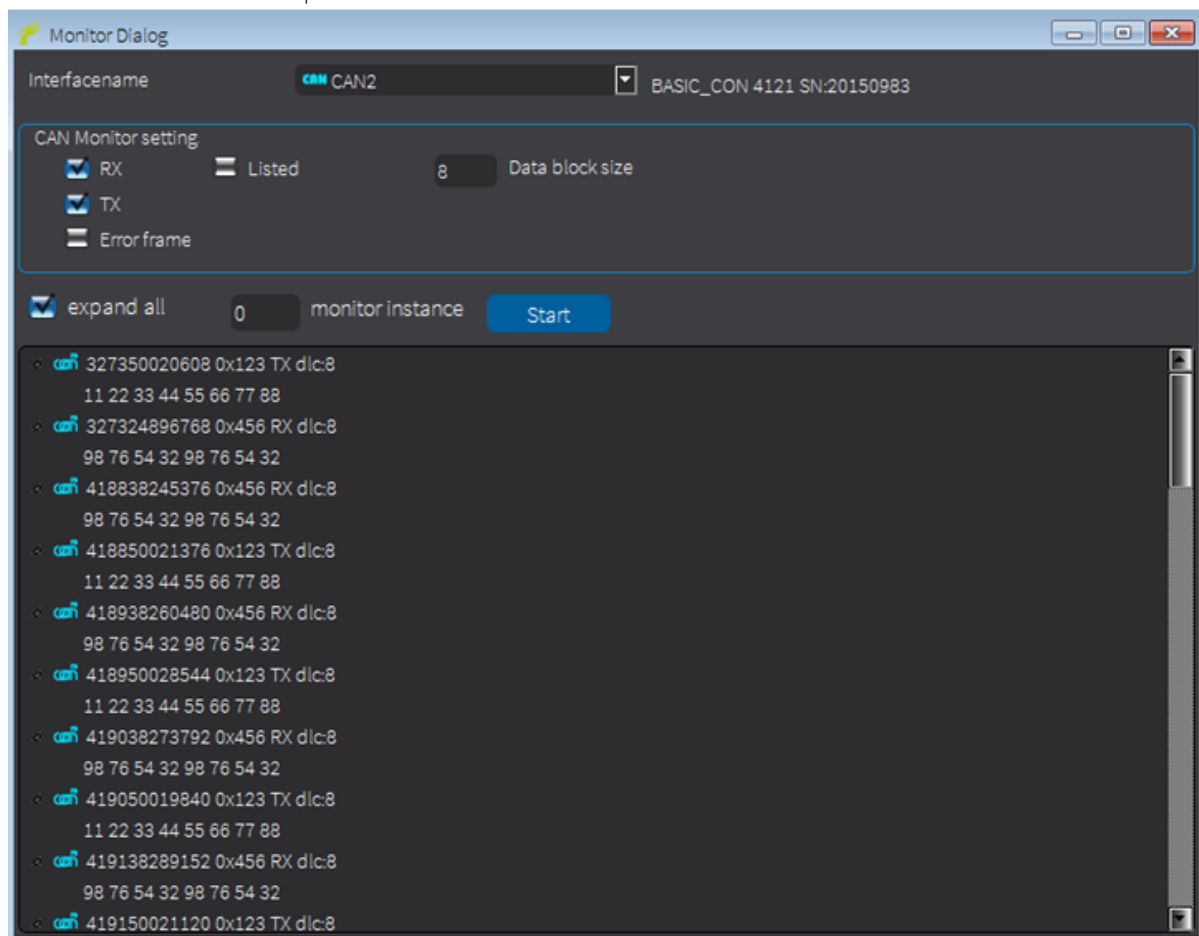
Interfacename CAN CAN2

The currently selected interface is displayed in the text field of the drop-down menu. Selecting the interface will automatically open the corresponding tab.

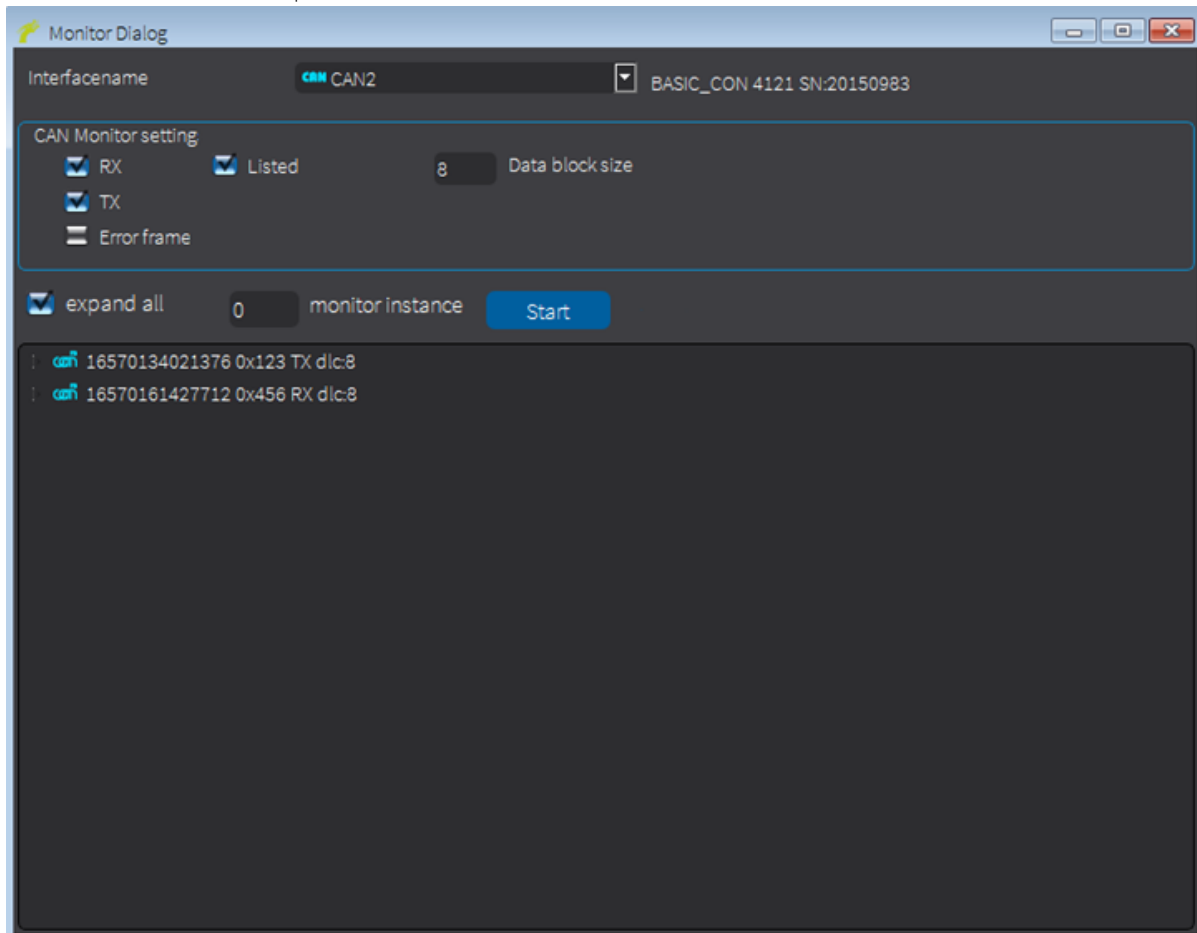
17.3.1 CAN Monitor

Set the monitor settings before starting the monitor. Determine whether **Rx** signals, **Tx** signals or **error frames** should be monitored. A combination of the signals is of course also possible. The monitored frames can be displayed consecutively below each other or **listed**. Listed means that for each signal there is one monitor entry whose time stamp is overwritten when this signal is repeated. **Data block size** indicates after how many bytes a line break occurs (the value must be at least 8).

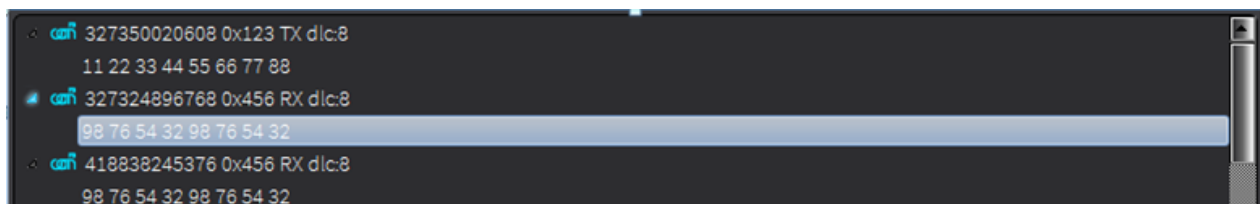
CAN Monitor - not listed & expand mode:



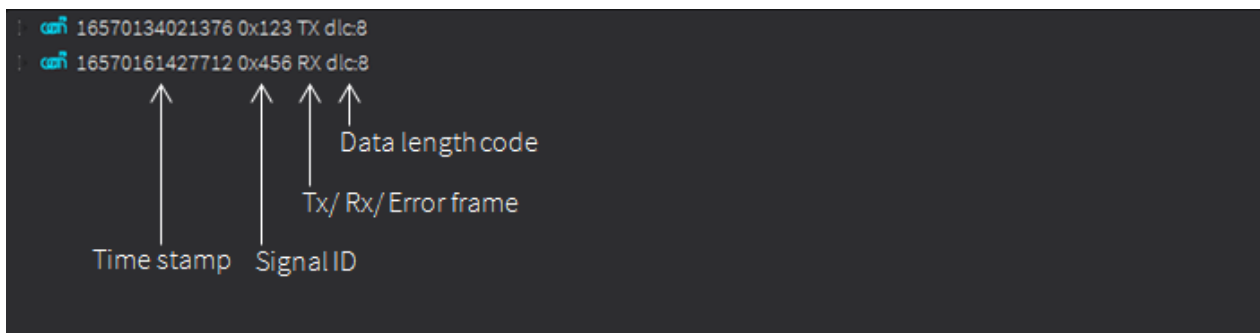
CAN Monitor - listed & no expand mode:



The signals can be displayed in **expand** mode or not. In expand mode, the data belonging to the signal ID is displayed in a second line. Without this mode the data is hidden, but can be opened individually in the monitor field. To do so, click on the small triangle to the left of the signal.



The displayed signal parameters are described in the following figure:



17.3.2 SPI Monitor/ SPI Analyzer

The API of the SPI Analyzer consists of two sections. The first section, **SPI_A**, configures the parameters, which are the

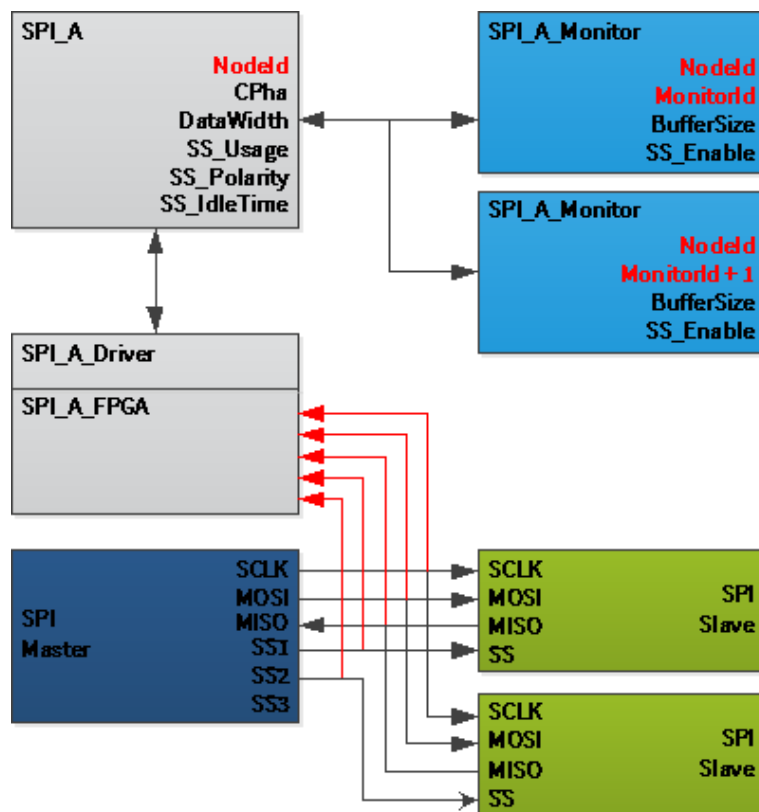
same for all monitors connected to this analyzer node and depend on the monitored SPI bus. These parameters include:

- CPha (ClockPhase): Data is active on first or second edge
- DataWidth: Data width from 8 to 32 bits
- SS_Usage: Which SlaveSelect should be monitored (Bit0..SS0, Bit1..SS1,...)
- SS_Polarity: Which polarity do these SlaveSelects have (0..Low, 1..High)
- SS_IdleTime: How long is the SlaveSelect at least inactive for the last SPI transfer to be considered completed (in ns)

The clock polarity is determined automatically.

In the second API section, **SPIA_Monitor**, several monitors can be configured for one SPI Analyzer node. This can be useful to allow each monitor to listen to its own SlaveSelect or a combination of SlaveSelects. The parameters are:

- BufferSize: internal buffer for monitor data
- SS_Enable: SlaveSelects observed by this monitor. They must be activated in the first API section (SS_Usage)



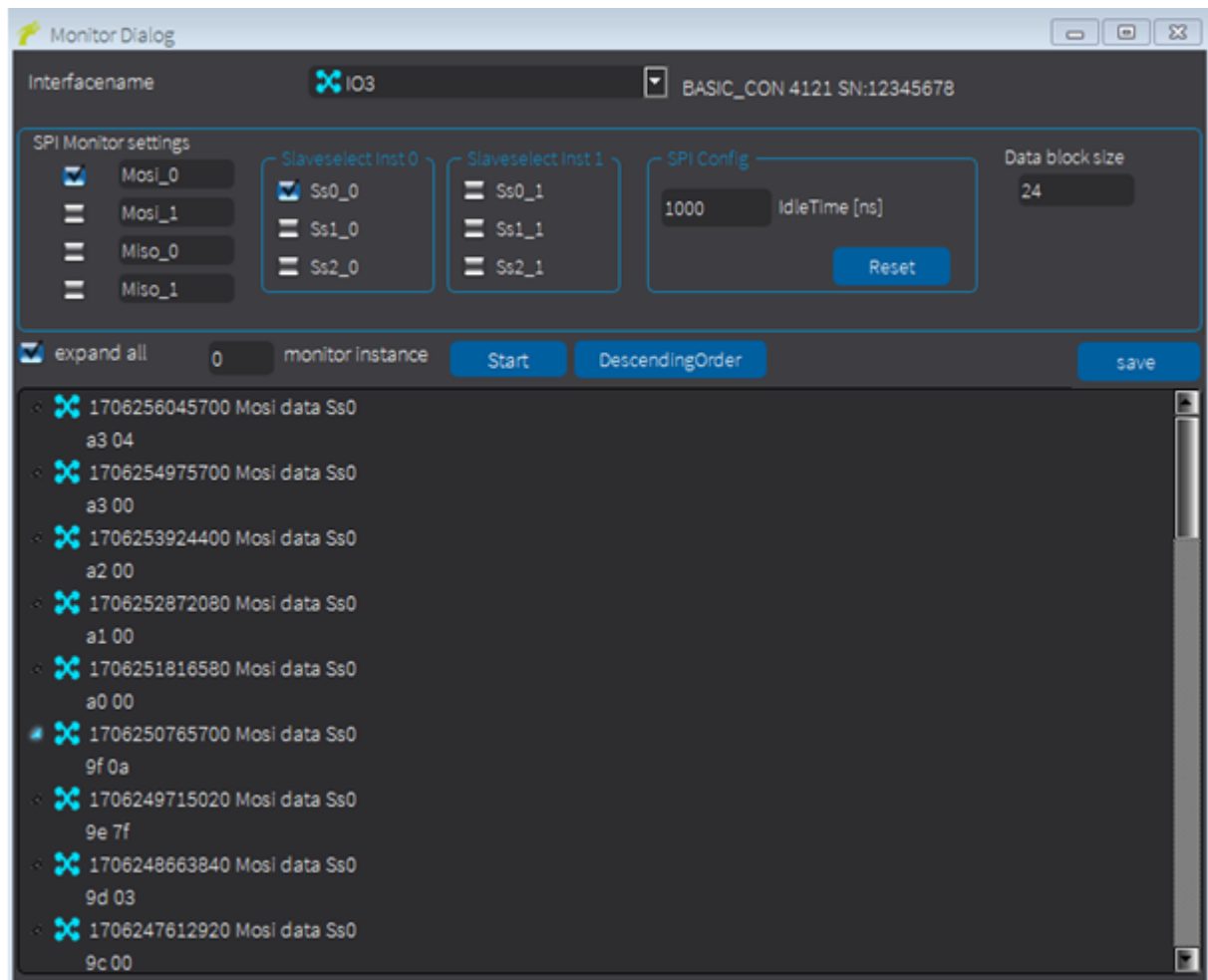
For the actual monitoring function it is necessary to route the inputs of the SPI Analyzer. For this we need the Clock, MISO, MOSI and SlaveSelect signals. These are supplied with the appropriate signal sources via **TriggerSource_Set** as corresponding targets. The routing can for example be like this:

- TargetType: **SPI_A_SCLK** (SPI Analyzer Clock Signal)
- TargetChannel: 0 (node ID of the SPI Analyzer)
- SourceType: **LVDS_MI_0_MFP** (Multifunctional purpose pins of the *Media Interface* LVDS)
- SourceChannel: 10



Since the SPI Monitor is configured via the IO interface, the IO interface must be opened here. Additionally the triggers must be configured correctly. Further information and **examples** can be found in the chapter [IO Tool](#) and at the end of this chapter.

Set the monitor settings before starting the monitor. Determine whether **MOSI** signals, **MISO** signals or which **SS** signals should be monitored. A combination of the signals is of course also possible. Since SPI communication often requires two analyzer instances, instances 0 and 1 can be selected by checkboxes for the individual signals. Depending on the selected signals, these are displayed in the monitor. In addition, an **idle time** can be set using the **Reset** button. **Data block size** indicates after how many bytes a line break occurs (the value must be at least 8).



TIP The name for Mosi/ Miso can be changed if necessary. To do this, click in the edit field and change the designation to e.g. Rx and Tx.

Info The data is written to the monitor from bottom to top and from right to left. Use the button **Descending Order** to change the data order.

The signals can be displayed in **expand** mode or not. In expand mode, the data belonging to the signal is displayed in a second line. Without this mode the data is hidden, but can be opened individually in the monitor field. To do so, click on the small triangle to the left of the signal.

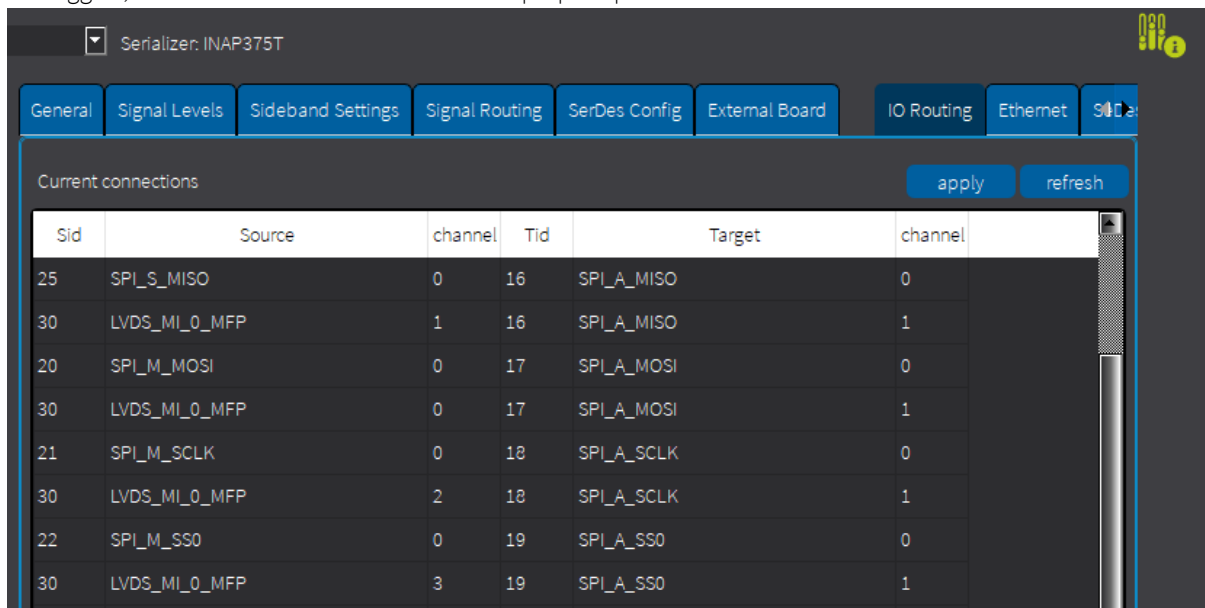


Use the **Save** Button to save the monitor data to a *.txt file.

17.3.2.1 Example for SPI Monitor

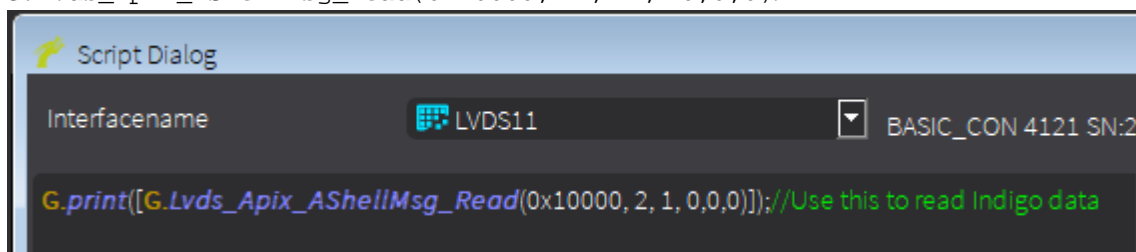
For understanding, here is an example of configuring the SPI analyzer via the IO interface and sending a read command to read the serial number of an *APIX* IC. For this purpose, a *basicCON 4121* with *INAP375T Media Interface* board is used, which communicates with an *APIX* deserializer. The following steps are done:

1. Set the IO triggers to configure the SPI analyzer. We want to read the sent and received messages with the monitor. For this two analyzer instances must be configured. The first instance is for reading the sent messages (instance 0), the second instance is for reading the received messages (instance 1). In the picture below you can see how the IO triggers are connected to each other. The target channels represent the analyzer instances. For instance 0 we use SPI triggers, for instance 1 the multifunctional purpose pins are used as source.



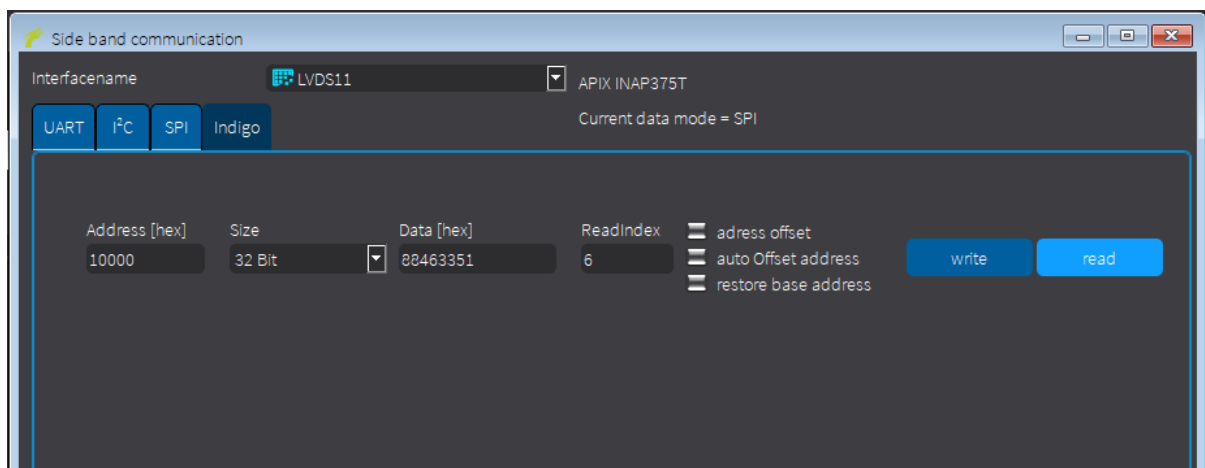
Sid	Source	channel	Tid	Target	channel
25	SPI_S_MISO	0	16	SPI_A_MISO	0
30	LVDS_MI_0_MFP	1	16	SPI_A_MISO	1
20	SPI_M_MOSI	0	17	SPI_A_MOSI	0
30	LVDS_MI_0_MFP	0	17	SPI_A_MOSI	1
21	SPI_M_SCLK	0	18	SPI_A_SCLK	0
30	LVDS_MI_0_MFP	2	18	SPI_A_SCLK	1
22	SPI_M_SS0	0	19	SPI_A_SS0	0
30	LVDS_MI_0_MFP	3	19	SPI_A_SS0	1

2. There are two ways to read the data. The first is to use the Script Interface. The *G-API* function to read the AShell message is included in the [Script Interface](#). Adapted to our DUT the script command is `G.Lvds_Apix_AShellMsg_Read(0x10000, 2, 1, 0,0,0)`.

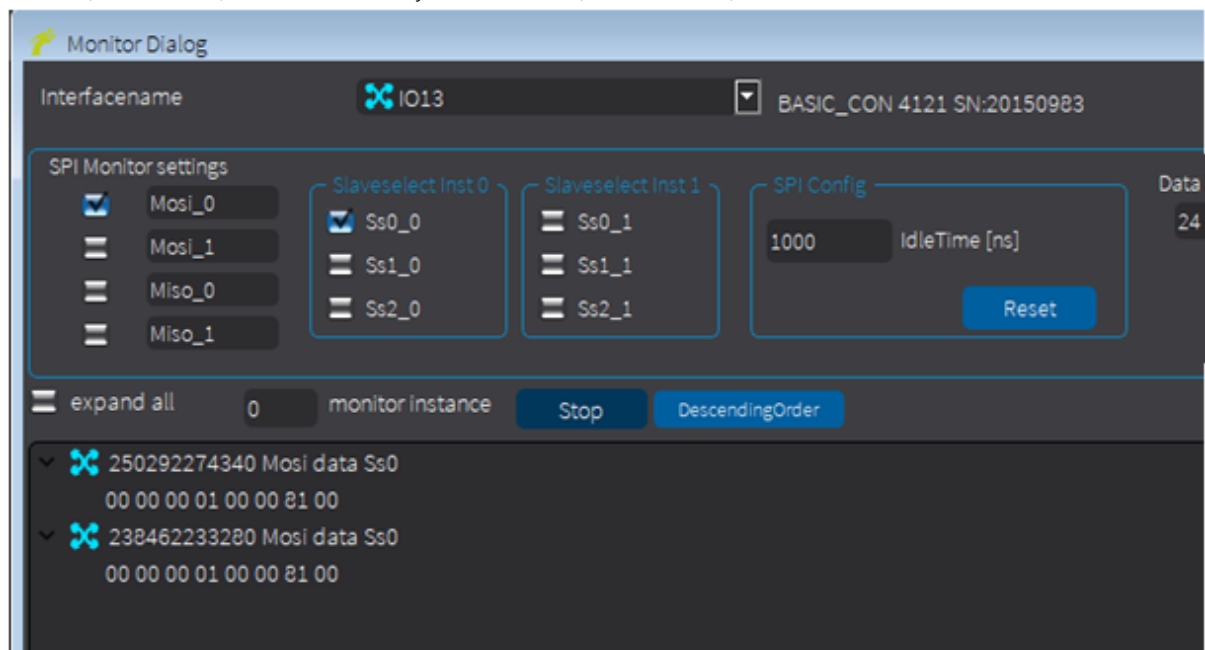


About the parameter values: The address is 0x10000 hex. Since the data size is 32 bit, the second value is "2". The Read Index is defined as "1". Since we do not set a flag, the remaining three values are specified as "0". If the command is packed in a *G.print* command, the answer is printed in the script output, in our case the serial number of the IC.

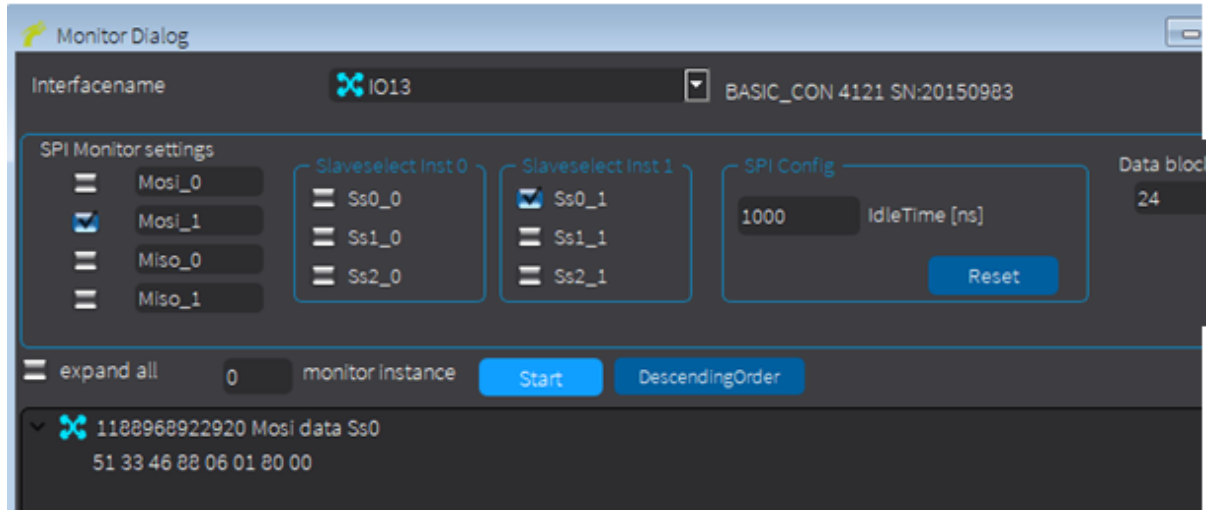
The second option is to use the [Indigo tab](#) in the sideband window. Here also the address and the data size are specified. The response (serial number) then appears in the data field.



- The SPI messages can be read in the SPI Monitor. Before sending the AShell Read command the monitor must be started (Start button). If we look at analyzer instance 0 (Mosi_0; Ss0_0), we see the sent data.



At analyzer instance 1 (Mosi_1; Ss0_1) we read the received data (from right to left). The serial number is 88463351.



17.3.3 UART Monitor/ UART Analyzer

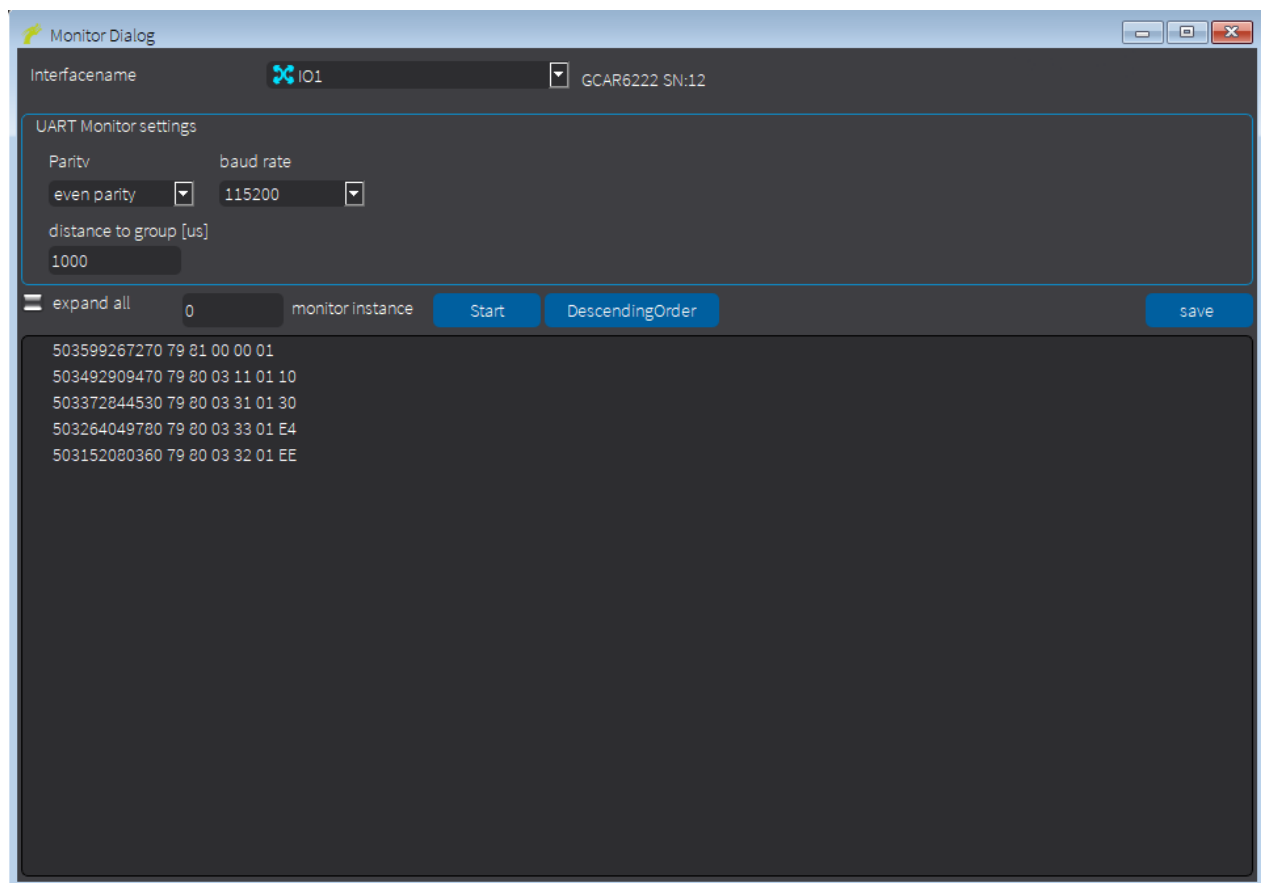
For the actual monitoring function it is necessary to route the inputs of the UART Analyzer. These are supplied with the appropriate signal sources via `TriggerSource_Set` as corresponding targets. The routing can for example be like this:

- TargetType: `UART_A` (analyser)
- TargetChannel: 0 (node ID of the UART Analyzer)
- SourceType: `LVDS_MI_0_MFP` (Multifunctional purpose pins of the *Media Interface* LVDS)
- SourceChannel: 25 (UART Tx signal for Serializer on *Media Interface* MAX9295_9296)



Since the UART Monitor is configured via the IO interface, the IO interface must be opened here. Additionally the triggers must be configured correctly. Further information and **examples** can be found in the chapter [IO Tool](#) and at the end of this chapter.

Set the monitor settings before starting the monitor. Parity and baud rate must match the values of the configured UART FIFO. **Distance to group** indicates the value of the sampling. The lower the value, the more frequently the signal is sampled and the messages are written to the monitor with a time stamp.



The data is written to the monitor from bottom to top and from right to left. Use the button **Descending Order** to change the data order.

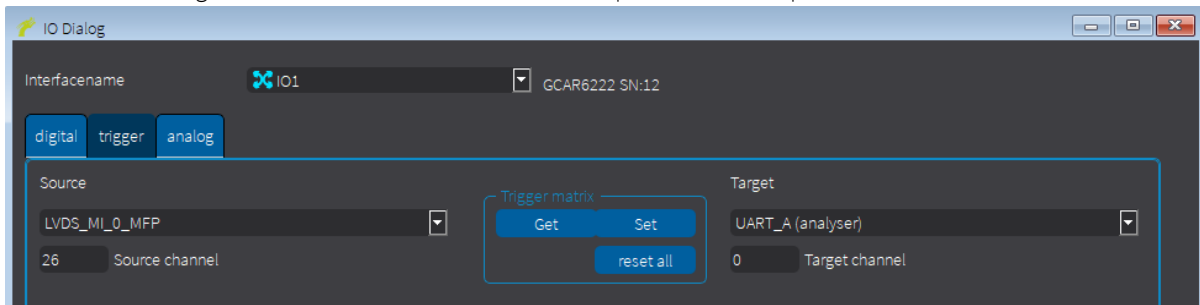
Use the **Save** Button to save the monitor data to a *.txt file.

17.3.3.1 Example for UART Monitor

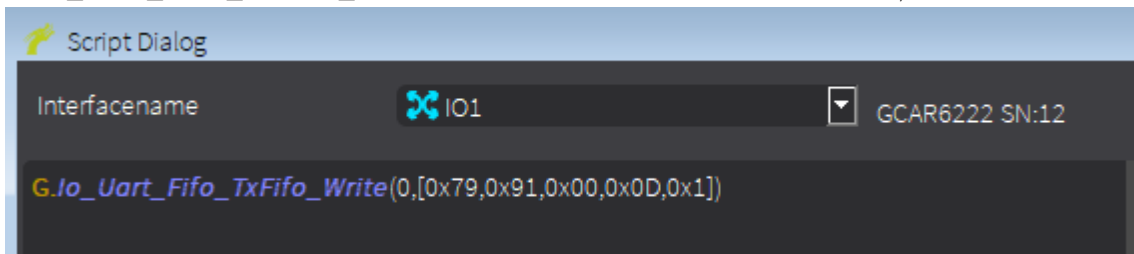
For understanding, here is an example of configuring the UART analyzer via the IO interface and sending a read command to read the serial number of an IC. For this purpose, a *Video Dragon 6222* with MAX9295 serializer is used,

which communicates with an MAX9296 deserializer. The following steps are done:

1. Set the IO triggers in the [IO Dialog](#) window to configure the UART analyzer. We want to read the device identifier of the deserializer. For this we have to pass the incoming Rx signal of the serializer to the UART analyzer. In the picture below you can see how the IO triggers are connected to each other. The target channel represent the analyzer instance. The Rx signal of the MAX9295 serializer is on MFP pin 26, which is specified as the source channel.

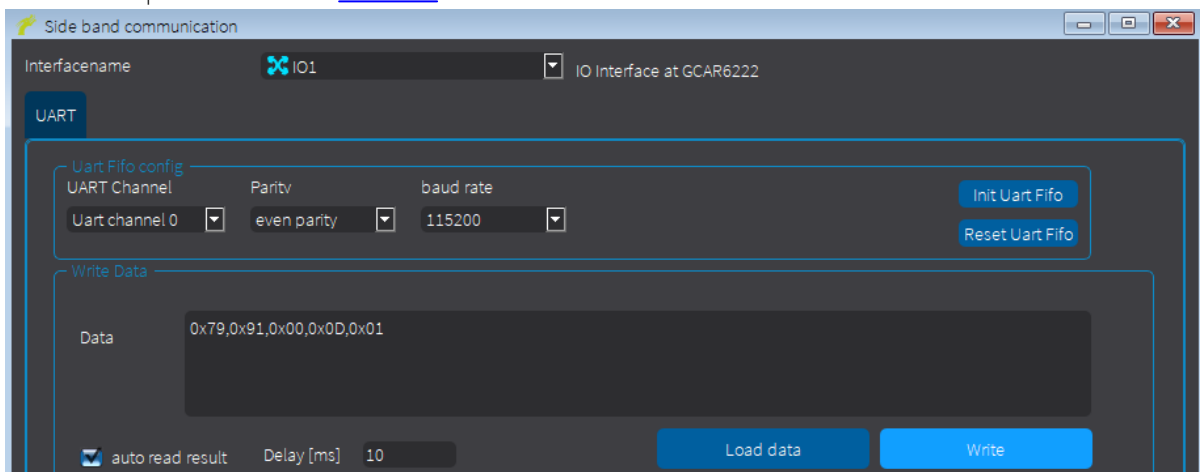


2. There are two ways to read the data. The first is to use the Script Interface. A FIFO is used for UART communication. To read a register, a write command must be executed first. The G-API function to write this command is included in the [Script Interface](#). Adapted to the Deserializer the script command is `G.Io_Uart_Fifo_TxFifo_Write(0,[0x79,0x91,0x00,0x0D,0x01])`.



About the parameter values: The first Byte 0x79 is the synchronization Byte. 0x91 is the device address in combination with the read command (last Bit). The register address is 0x00,0x0D (two Byte). The last Byte (0x01) is the number of Bytes to read.

The second option is to use the [UART tab](#) in the sideband window.



3. The UART Rx messages can be read in the UART Monitor. Before sending the Read command the monitor must be started (Start button). The received data is 0xC3, 0x94. 0xC3 is the acknowledge Byte, 0x94 is the register value.

UART Monitor settings

Parity

even parity

baud rate

115200

distance to group [us]

1000

expand all

0

monitor instance

Start


DescendingOrder

save


13303175094350 C3 94

17.4 MiMfp Config Tab

Use this tab to configure the Multifunctional Pins of the installed Media Interface. Each media interface has a different meaning of the MFP pins. An overview of the MFP pins of the individual media interface boards can be found in the *Video Dragon 6222* manual.



This feature is only supported for *Video Dragon 6222*.



Manual manipulating of the MFP configuration needs an extreme good knowledge of the meaning of the pins. Even a single wrong setting of only one pin may render the complete configuration invalid and the device inoperative. **Please make sure to consult with the [Göpel Support Team](#) before use.**

GeneralSignal LevelsSideband SettingsSerDes ConfigExternal BoardLVDS ChannelsIO RoutingEthernetSeDesGPIO ConfigMiMfp Config

SerDesGpio-index	locOutputType	locOutputMode	locInputValue	locOutputValue	active
Lvds MI Mfp 0	input only	no change			
Lvds MI Mfp 1	push/pull output mode	set			
Lvds MI Mfp 2	input only	no change			
Lvds MI Mfp 3	input only	no change			
Lvds MI Mfp 4	input only	no change			
Lvds MI Mfp 5	input only	no change			
Lvds MI Mfp 6	input only	no change			
Lvds MI Mfp 7	input only	no change			
Lvds MI Mfp 8	input only	no change			
Lvds MI Mfp 9	input only	no change			
Lvds MI Mfp 10	input only	no change			
Lvds MI Mfp 11	input only	no change			
Lvds MI Mfp 12	input only	no change			
Lvds MI Mfp 13	input only	no change			
Lvds MI Mfp 14	input only	no change			
Lvds MI Mfp 15	input only	no change			
Lvds MI Mfp 16	input only	no change			
Lvds MI Mfp 17	input only	no change			

This tab has the following parameters:

Parameter	Description
SerDes Gpio-index	MFP pin number.
locOutputType	Each pin can be defined as input or as output (seen from the FPGA). The output can be defined as push/pull or open drain mode. The pin is then driven by the FPGA.
locOutputMode	If the pin is defined as output, the value can be set via the Output Mode. <ul style="list-style-type: none"> • set: OutPut Value = high • reset: OutPut Value = low • toggle: OutPut toggles between high and low
locInputValue	Shows the actual input status at the pin. This is a read only value. <ul style="list-style-type: none"> • green = high • grey = low
locOutputValue	Shows the actual input status at the pin. This is a read/ write value. By clicking on the round button the output value can be manually set to high (green) or low (grey). Since the status of the pin in the device is updated immediately when changes are made in the tab, this is the fastest way to change the output value.
Active	This is an indicator whether the setting for the MFP pin should be saved in the serializer/deserializer configuration file. Also, when you click Apply to HW , only the active pins on the hardware are overwritten. <ul style="list-style-type: none"> • green = active • grey = not active

Right-clicking into the tab gives the following options:

Option	Description
Load all pins from device	Loads all pin configurations from the device and sets all Active indicators to low. (Attention: Using the button Load current values sets all Active indicators to high. When saving the settings to a file every MFP pin will be saved now. This can lead to faulty behavior when the file saved in this way is loaded onto the device.)
Apply table to hardware	Applies actual table to the hardware, no matter if the Active indicators are high or low.
Copy script command to clipboard	Right-click on the desired MFP pin and select this option. This will automatically copy the appropriate script command to the clipboard and can be pasted into the Script Interface .

18 Additional Features

18.1 TCP Remote Control

Via TCP a remote access to the script interface of the Dragon Suite is possible. For this, one or more script engines run on the TCP server to execute scripts. These scripts are transmitted from the client to the server using JSON format.

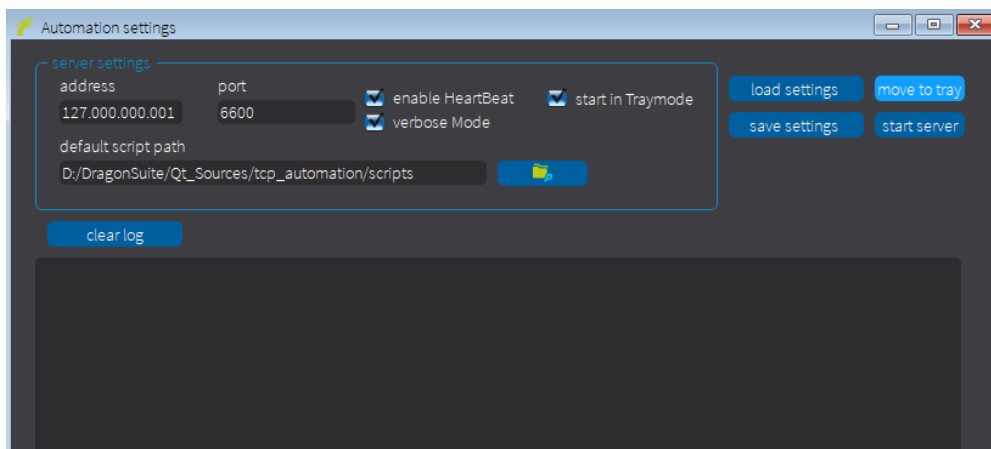


This feature is chargeable. A Dragon Suite Advanced license is also required for this feature.

18.1.1 Server Settings

The server settings must be set once in Dragon Suite before starting the transfer.

1. Open the Automation Settings via the toolbar icon
2. Set a server address and the server port. Optionally set the default path where the scripts are located. **Enable Heartbeat** sends a regular signal from the server. When **Verbose Mode** is selected, the server outputs all incoming and outgoing messages to the console. This helps with debugging.
3. These settings must be saved in a *.ini file via the **Save Settings** button. Once saved, the settings can also be loaded via the **Load Settings** button.
4. The server can be started by clicking the **Start Server** button. Now the client can connect to the server.



The server can also be opened via the Windows Console.

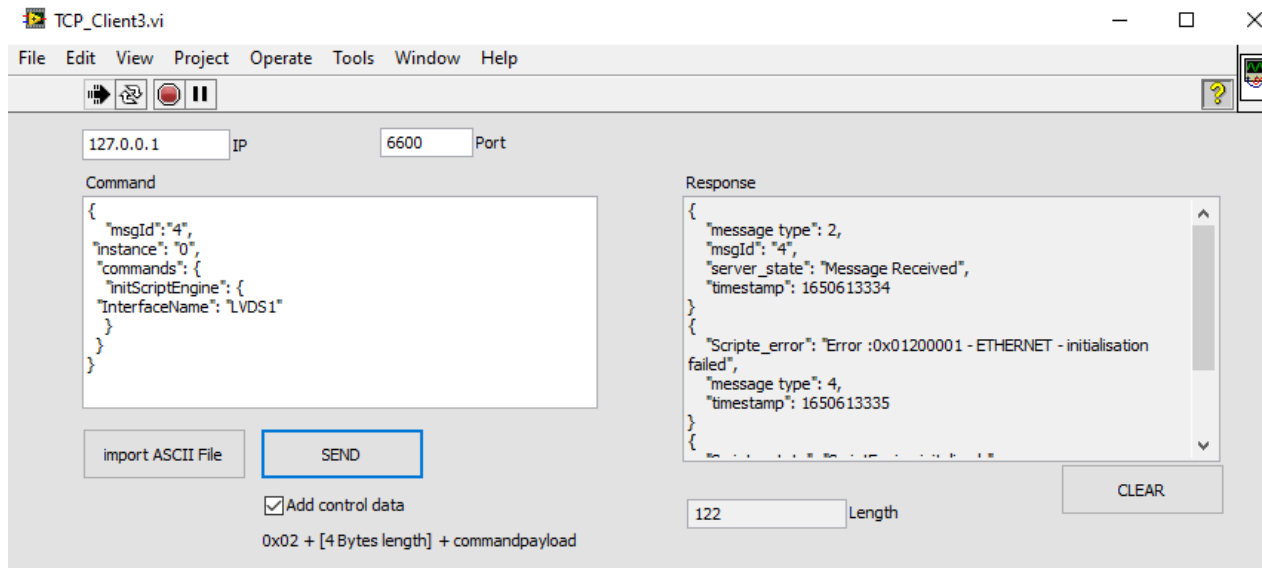
```
dragonSuite.exe -startServer          (starts as default)
dragonSuite.exe -startServer:*.ini    (loads the previously saved *.ini file)
```

18.1.2 Client Settings



Only one client can connect to the TCP server.

Once the server is running, a client can communicate with it. In the TCP Automation examples there is a LabVIEW VI which is used to display the client: "TCP_Client3.vi" as shown below.



With the help of this example, the steps to communicate with the server are described:

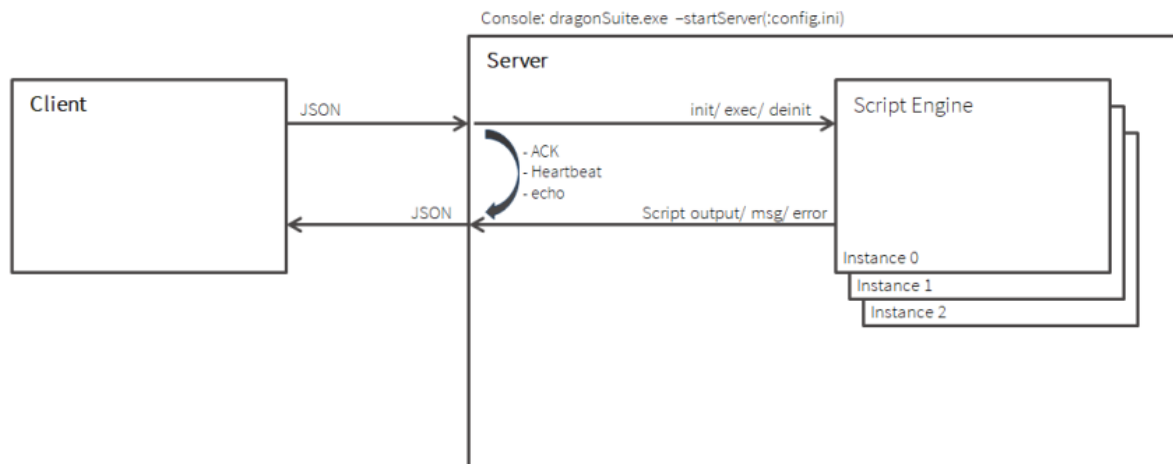
1. An IP connection to the server should be established from the client. So the IP address and port of the server are specified first.
2. A command must be defined. There are also some examples for this in JSON file format . A file can also be imported directly via the VI.
3. When sending a command, the following control data always should be added : **0x02 as control (1 Byte) + payload length (4 Bytes) + payload (JSON string as Byte array)** . This can happen in the example VI by the appropriate selection.
4. The command is now sent. The response message from the server is output in the response field.

18.1.3 Remote control

In the following, the JSON messages are presented as follows:

Message sent by the client
Message received from the client

The TCP connection has the following scheme:



To communicate with the server JSON messages are sent to the server via the client. The server acknowledges every client request. In addition, the server can optionally send a heartbeat every second. As soon as the server is started and the client has established the TCP connection to the server, the heartbeat is visible on the client side. Now the Script Engine can be initialized. The initialization must always be done to operate with the Script Engine. For this purpose, the instance of the engine must be defined. At the moment only one instance can be opened. The application with multiple instances will come soon. The command to initialize the engine should look like this:

```

{
  "msgId": "4",
  "instance": "0",
  "commands": {
    "initScriptEngine": {
      "InterfaceName": "LVDS1" //opens interface LVDS1 in Script Engine
    }
  }
}

```

In all future messages, the command will be assigned via the instance of the correct Script Engine. After initializing the Script Engine, you can execute

- script code:

```

{
  "instance": "0",
  "commands": {
    "execScriptCode": {
      "scriptCode": "G.print(\"Hallo World\")"
    }
  }
}

```

- script files:

```

{
  "msgId": "INIT_DUT",
  "instance": "0",
  "commands": {
    "initScriptEngine": {
      "InterfaceName": "LVDS1"
    },
    "execScriptfile": {
      "ScriptName": "INIT_DUT.js"
    }
  }
}

```



Remote scripts should always be tested directly in the Dragon Suite script interface first to make sure they work.

- or mathematical operations:

```
for(var i=0; i<10;i++){
G.sendJsonStringToClient("Hello World "+ i + " from Script ");
delay(1000000);
}
```



Although both are possible, G.print() should not be used to communicate with the client. G.sendJsonStringToClient(myJSON) is recommended.



In the script, no functions can be called that, for example, display an image.

The script engine executes the command and responds with

- **a message.** The following messages from server and Script Engine are possible:

```
typedef enum{
    TYPE_ERROR = 0,
    TYPE_MESSAGE,
    TYPE_STATUS, // e.g. Client connected to server
    TYPE_SCRIPT_STATE,
    TYPE_SCRIPT_ERROR,
    TYPE_SCRIPT_INFO,
    TYPE_SCRIPT_PROGRESS,
    TYPE_SCRIPT_OUTPUT // e.g. of G.print("")
}ServerToClientMessageType_t;
```

Example:

```
{
  "Script_output": "Hello World",
  "message type": 7,
  "timestamp": 1649682124
}
```

- or **an error message.** The following errors from server and Script Engine are possible:

```
typedef enum{
    No_Error = 0,
    Invalid_Json,
    JsonStringNoObject
}automation_error_t;
```

Example:

```
{
  "Scripte_error": "Error :0x0009020D - FW - LVDS - data - no acknowledge
23 02 10 00 01 01 00 00 00 00 00 60 06 00 00 00",
  "message type": 4,
  "timestamp": 1649682259
}
```

The deinitialization of the Script Engine is done by:

```
{
  "msgId": "5",
  "instance": "0",
  "commands": {
    "deInitScriptEngine": {
      "value": "doIt"
    }
  }
}
```

In addition, it is sometimes necessary to stop the scripts if for example they endlessly run a loop:

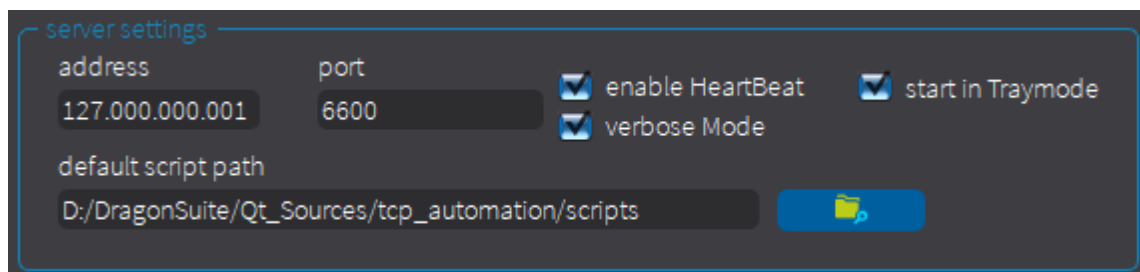
```
{
  "msgId": "3",
  "commands": {
    "stopAllScripts": {
      "instance": "0"
    }
  }
}
```

The **message ID** can be freely assigned. This is always returned with the script output. This allows the affiliation of the response to the command to be traced.

18.1.4 Tray mode

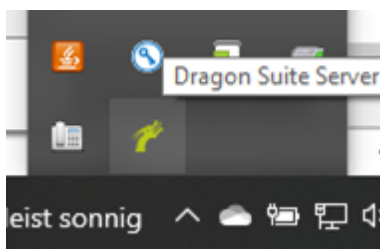
There is a possibility to use the TCP connection while *Dragon Suite* is running only in the background. This means you can put the application completely into the Windows tray.

If this is to happen when the server is started automatically, it must be saved in the configuration. For this **start in tray mode** must be set when saving the configuration.



To test the functionality or to pack the *Dragon Suite* window into the background, you can also simply use the **move to tray** button.

To reopen the window, open the Windows tray (usually located in the lower right corner on the toolbar) and go to the Dragon logo. Double click on the logo or use right click and then click on Restore.



19 First Steps

Prerequisite for these steps is a successful installation of the *G-API* and the *Dragon Suite*.



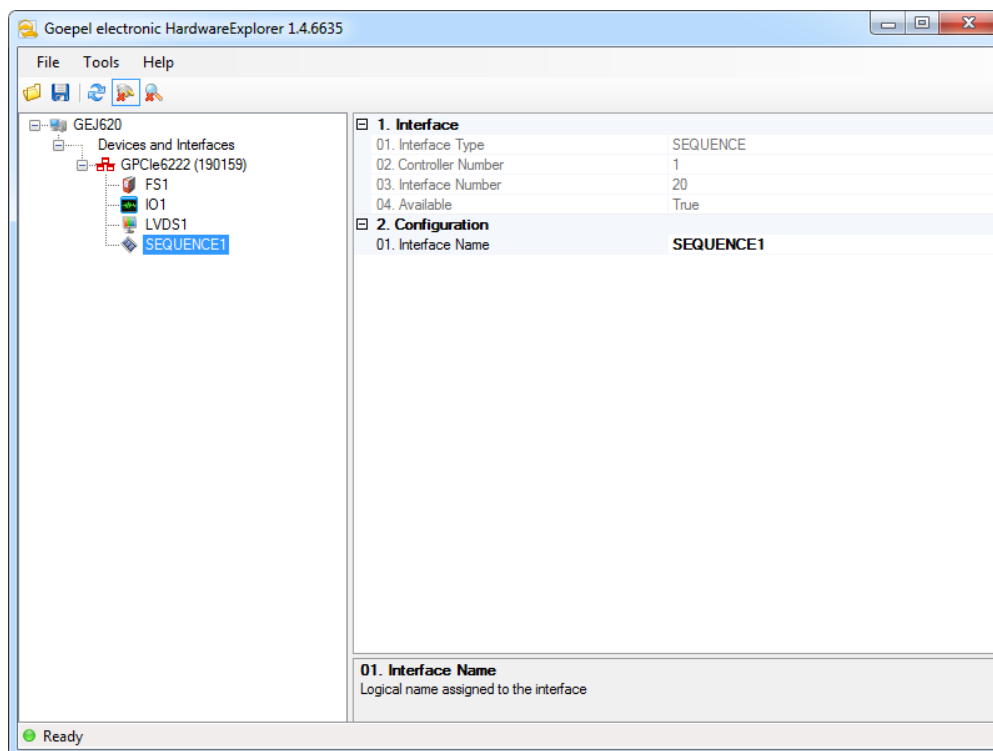
The supported *basicCON 4121* and *Video Dragon 6222* devices can be used with different Media Interface Modules and must be configured differently. For this chapter, it is assumed that a DS90UB954 deserializer module is installed in a *GPcie 6222* Board, and a compatible video source is already set up and ready for use.

19.1 System Structure

1. Install the appropriate module on the main board (in this example it is DS90UB954). (Typically, this step is not required because a module is already installed at delivery.)
2. Install the *GÖPEL electronic* device in your (switched off) test system or your PC.
3. Connect your video source to the input connector of the *GÖPEL electronic* device using the supplied video cable.
4. Switch on the test system and thus the *GÖPEL electronic* device. As soon as the device is ready, LED2 starts flashing.

19.2 Registration

Before you can use the *GÖPEL electronic* device for the first time, it must have been registered in the *G-API*. The *G-API* is responsible for all future communication between the control PC or laptop and the *GÖPEL electronic* device. This registration is simply done by starting the *HardwareExplorer*. The following figure shows a *GPcie 6222* board with four interfaces:



The *GÖPEL electronic* device can be seen in the left column with all available software interfaces. If several devices are connected, the corresponding device can be identified by its serial number, which is shown in parentheses. The name of the LVDS interface (e.g., "LVDS1") is important to the following steps.



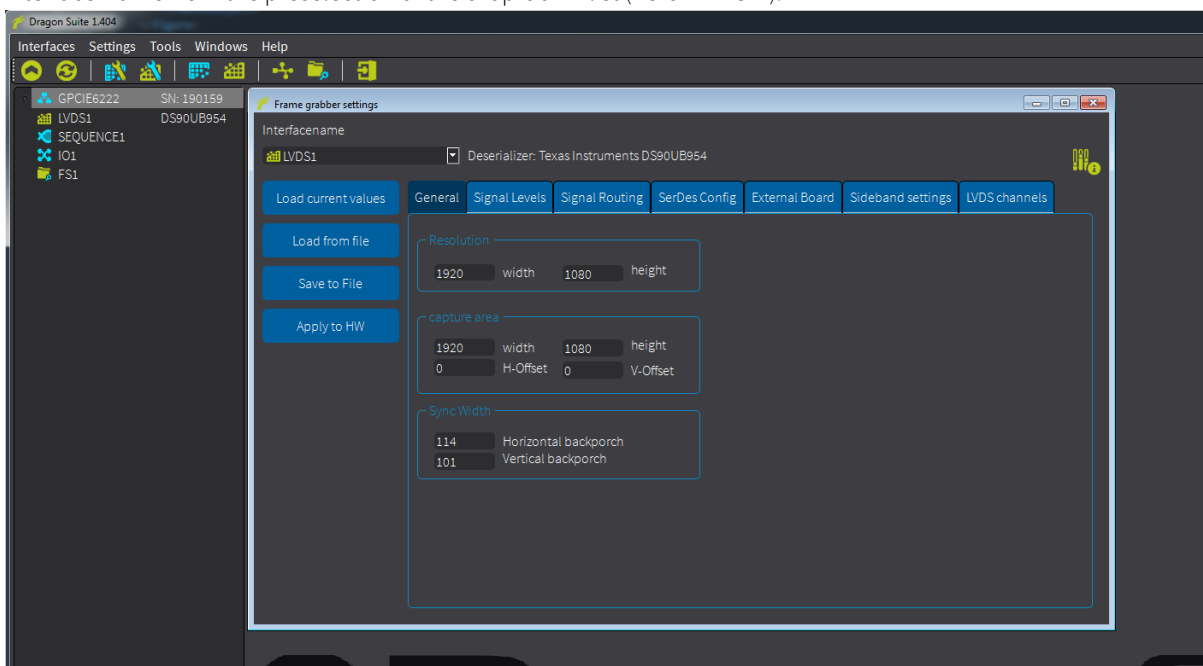
For more information about the *G-API*, its installation, and the *Hardware Explorer*, see *G-API Quickstart Guide*.

19.3 Configuration

Before the capturing of frames is possible, the device must be configured according to the currently transmitted video signal.

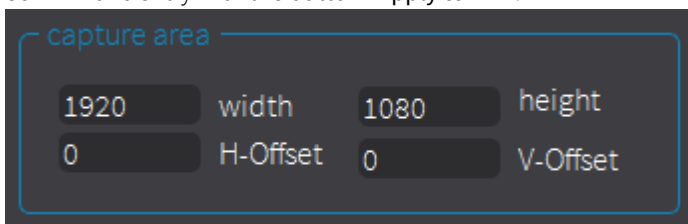
1. Start the *GÖPEL electronic Dragon Suite* software. On the left side in the [Interface Tree](#) the *G PCIe 6222 Board* and its interfaces appear, similar to the *Hardware Explorer*.

2. The icon  opens the [Settings Window](#) for the *Frame Grabber*. Select the corresponding LVDS interface as Interface Name from the preselection of the drop-down list (here "LVDS1").

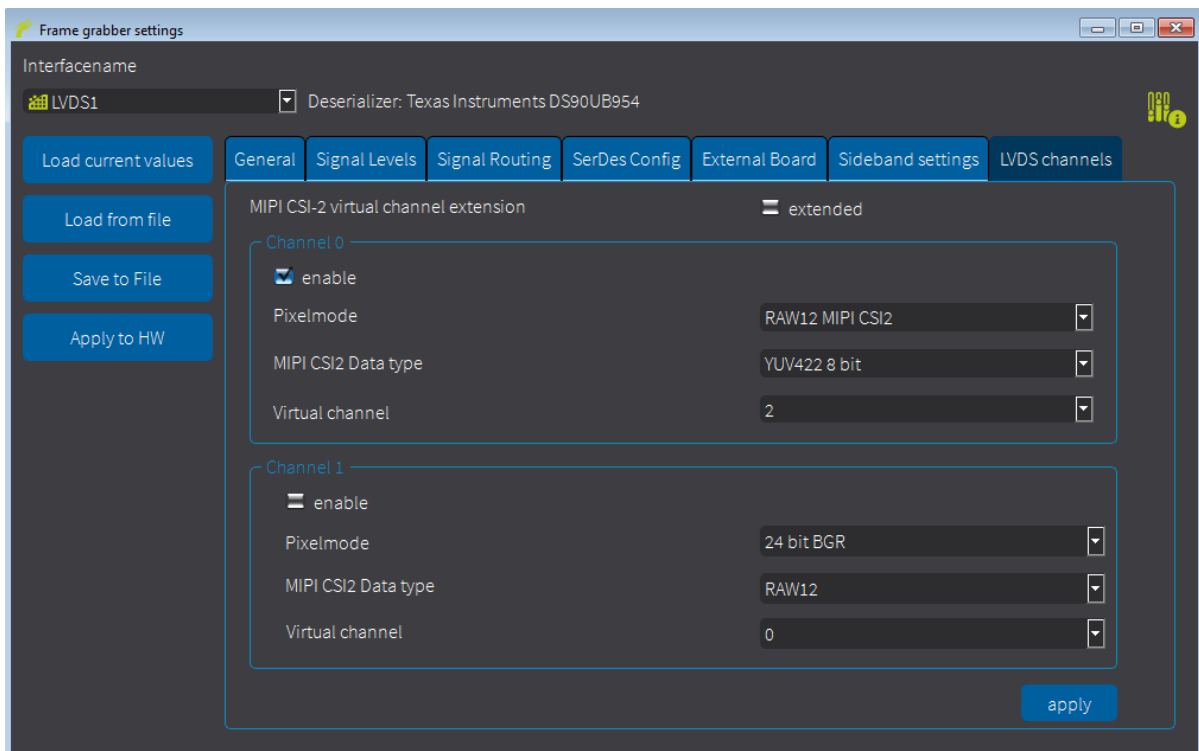


All settings for the deserializer can be made in this dialog.

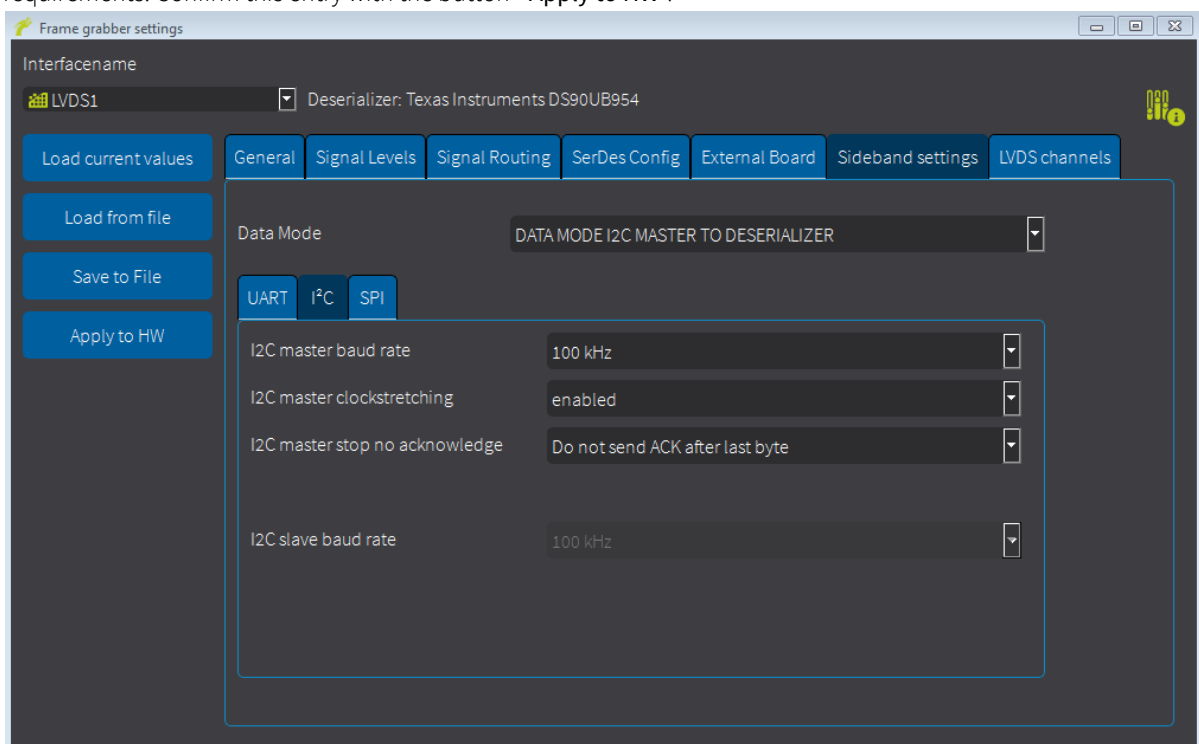
3. After selecting the LVDS interface, the values in the configuration window are overwritten with the current values of the *GÖPEL electronic* device automatically. The current configuration parameters can also be loaded into the settings window via the button "**Load current values**".
4. Enter the desired resolution in "**Capture Area**". This must not be higher than the resolution of the incoming frame. Confirm this entry with the button "**Apply to HW**".



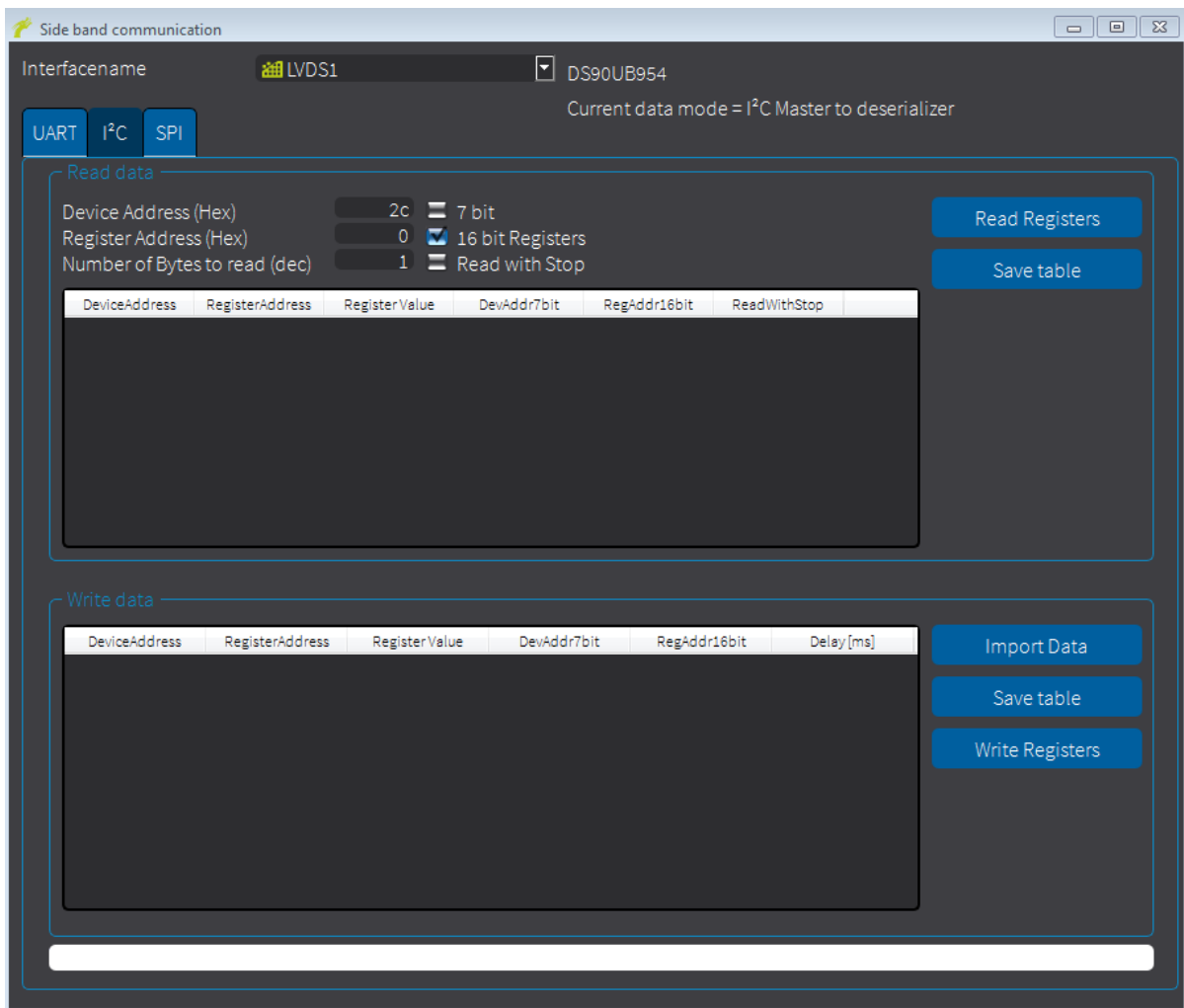
5. Switch to the [LVDS Channels](#) tab and adjust the parameters of the physical channels according to your test requirements. Confirm this entry with "**Apply**" in the lower right corner of the dialog box.



6. In most cases, the configuration registers of the deserializer must be adapted to the test environment. In this example, the registers are written using I²C communication. Switch to the [Sideband Settings](#) tab and change the Data Mode to "I²C Master to Deserializer". Adjust the other parameters (baud rate etc.) according to your test requirements. Confirm this entry with the button "Apply to HW".




7. Open the [Sideband Communication Window](#) with the icon . Also select the LVDS interface here ("LVDS1"). The tab for I²C opens automatically if the Data Mode was successfully overwritten before.





Here, the configuration registers of the deserializer can be read out and overwritten. Reading is done by entering the device address and register address and the **"Read Registers"** button. Save the table with **"Save Table"**. Now you can change the desired tabs in the saved text file. With **"Import Data"** the changed list is imported again and by clicking on **"Write Registers"** the registers of the deserializer are changed.

The device is now configured and ready to capture frames.

19.4 Capturing

The icon  opens the [Frame Grabber Dialog Window](#). It is possible to capture both single frames and a sequence of frames.

The icon  captures a single frame from the video stream and displays it in the dialog window. The capturing of a sequence of frames is started by pressing the icon . After each captured frame of the sequence, the dialog window is updated to show the frame being captured.



Your *Video Dragon* and also the *Dragon Suite* offers a multitude of additional functions. In this chapter, only general instructions for working with the device could be given by way of example.

20 Common Error Messages

The following table shows common *G-API* error messages and how to fix them:

Error Code	Error Message	Description
0x000000E0	FW - function not available	The selected feature (e.g., Sideband Communication or CAN) is not activated for this device.
0x00090002	LVDS - capturing not in progress	The deserializer does not receive image data from the serializer. Check the lock between serializer and deserializer. The cause may also be due to a bad signal, so that a frame grabber has difficulty capturing. Also check the connection between the PC and the device.
0x00090006	LVDS - no video lock	There is no lock between serializer and deserializer. Make sure the configurations of the devices match. In most cases, the Signal Levels (Polarities) or the Control Signals (HSync, VSync, DataEnable) do not match.
0x00090007	LVDS - initializing after lock lost	The link between serializer and deserializer was lost. This must now be reinitialized. The cause may be due to a bad signal, so that a frame grabber has difficulty capturing. Also check the connection between the PC and the device.
0x0009010A	LVDS - config - capture area bigger than frame	The defined Capture Area of the Frame Grabber is larger than the transmitted video frame.
0x00090201	LVDS-data-extensionboard function not available	One of the selected modes is not supported for the serializer/ deserializer.
0x0009020D	LVDS - data no acknowledge	The device receives no response from the receiver during sideband communication (e.g., I ² C). There may be no connection or the recipient address is incorrect.
0x0206000D	LVDS-sync error-videolock not active	There is no lock between serializer and deserializer. Make sure the configurations of the devices match. In most cases, the Signal Levels (Polarities) or the Control Signals (HSync, VSync, DataEnable) do not match.
0x2000107	API - event timeout	The command triggered a timeout in the <i>G-API</i> . There was no frame at the grabber input. The cause may be due to a bad signal, so that a frame grabber has difficulty capturing. Also check the connection between the PC and the device.
0x2000205	API - no response from device	The <i>G-API</i> has lost connection to the device. The cause maybe due to a bad signal, so that a frame grabber has difficulty capturing. Also check the connection between the PC and the device.

Some error messages can occur at the same time and have the same cause. This is because some applications call multiple functions of the *G-API*, and then each returns an error.



Please make sure to keep the *G-API* and the *Dragon Suite* [up to date](#). This can prevent some errors.



Always check the connection between the PC and the *GÖPEL electronic* device. We recommend the use of Ethernet. A USB2.0 connection, for example, is too slow to achieve error-free capture. It may also be necessary to restart the device.

21 Service and Support

21.1 Spare Parts and Accessories

If necessary, please contact our sales department:

GÖPEL electronic GmbH

ATS-Vertrieb

Göschwitzer Str. 58 / 60

D-07745 Jena

Tel.: +49-3641-6896-508

E-Mail: ats.sales@goepel.com

<https://www.goepel.com>

21.2 Warranty and Repair

21.2.1 Conditions

We guarantee the accuracy of the test system for a period of 24 months from the date of sale. The warranty does not apply to errors that are based on improper interventions or changes or improper use.

21.2.2 Identification

Furthermore, we ask you to announce possible warranty cases as such. Repair orders without reference to an existing warranty claim will in any case initially be paid. If the warranty has expired, we will of course also repair your test system in accordance with our general installation and service conditions.

If necessary, please contact our support service:

GÖPEL electronic GmbH

ATS-Support

Göschwitzer Str. 58 / 60

D-07745 Jena

Tel.: +49-3641-6896-597

E-Mail: ats.support@goepel.com

<https://www.goepel.com>

22 Disposal

22.1 Disposal of used Electrical / Electronic Equipment

The device implements the following EU directives:

- 2012/19/EU (WEEE) Waste Electrical and Electronic Equipment and
- 2011/65/EU on the restriction of the use of certain hazardous substances in electronic equipment (RoHS directive)

At the end of the life of the device, this product must not be disposed of with other household waste. The improper disposal of this type of waste can have a negative impact on the environment and health due to the potential hazardous substances in electrical and electronic equipment. Dispose of the product at a suitable collection point.



When disposing of the device in countries outside the EU, local laws and regulations must be observed.

22.2 Disposal of used Disposable / Rechargeable Batteries

At the end of the service life of disposable/ rechargeable batteries, these must not be disposed of with the normal household waste. Dispose of the disposable/ rechargeable batteries at a recycling center for disposable batteries and rechargeable batteries.

Please dispose of only discharged disposable/ rechargeable batteries.