Orlando™

frame grabber

User Manual

Software Development Kit

Draft version

doc 1.07

ARVOO Imaging Products BV
P.O. Box 439
3440 AK Woerden
The Netherlands
Tel +31-348-413 897
Fax +31-348-417 242
website: http://www.arvoo.com
general e-mail: info@arvoo.com
support e-mail: support@arvoo.com

Content

1 Requirements

Requirements orlando PeX-CL and Pex-AN:
- PC based on a x86 32-bit or x86 64-bit (AMD64 or Intel EM64T) CPU
- One free PCI Express slot

Requirements orlando 104-BASE24 and 104-AN:
- PC/104-plus motherboard based on a x86 32-bit or x86 64-bit (AMD64 or Intel EM64T) CPU

General Requirements
- Operating System:
  Microsoft Windows 2000, XP, XP-x64, Vista, Vista-x64 or newer
  Linux version 2.4 or newer, with kernel sources installed
  QNX 6.1 for x86 or newer

2 Windows SDK

2.1 Installation

2.1.1 Windows 2000

The Driver Signing option of Windows 2000 should set to Warn or Ignore:
click Start button
click Settings
click Control Panel
click System, tab Hardware
click Driver Signing and select Ignore or Warn

Installation
Start your computer with Windows 2000
When Windows 2000 detects new hardware, "Found New Hardware Wizard" starts
Insert the orlando SDK CD.
Select: Specify a location, and deselect other options.
Browse to the orlando SDK CD, select arvoo.inf.
The orlando board will be installed

Run orlando_setup.exe to install the Orlando SDK.

2.1.2 Windows XP

The Driver Signing option of Windows XP should set to Warn or Ignore:
click Start button
click Control Panel, click Performance and Maintenance
click System, tab Hardware
click Driver Signing and select Ignore or Warn

Installation
Start your computer with Windows XP
Log in with Administrator rights

When Windows XP detects new hardware, "Found New Hardware Wizard" starts
Insert the "orlando SDK" CD.
Select: Install from a list or specific location (Advanced)
Select: Search for the best driver in these locations
Select only: Include this location in the search:
Browse to the orlando SDK CD, select arvoo.inf
The orlando board will be installed
If a dialog box appear with "The software you are installing for this hardware: ARVOO Orlando has not passed... " click on Continue Anyway

Run orlando_setup.exe to install the Orlando SDK.

2.2 Installed files

file name	destination	description
	windows directory	
arvdrv.sys	\system32\drivers	kernel driver
arvdrv.dll	\system32	low level driver
orlando.dll	\system32	orlando driver
	program files\arvoo\..	
arvtyp.h	...include	declarations of basic types
orltyps.h	...orlando\include	orlando typs, enums, structs declarations
orlando.h	...orlando\include	orlando function declarations
orlc_cl source code	...orlando\orlc_cl	source code of orlando ANSI-C sample application for Orlando CL
orlvc source code	...orlando\orlvc	source code of orlando MFC sample application for Orlando CL
orlc_an_mod soource code	..\orlando\orlc_an_mod	source code of orlando ANSI-C sample application for Orlando AN
orlc_an source code	...orlando\orlc_an	source code of orlando ANSI-C sample application for Orlando AN obsolete
orl_java	...orlando\orl_java	source code and JAR file of orl_java Java/SWT application for Orlando CL and AN
orlc_cl.exe	...orlando\bin	binary of ANSI-C sample application
orlvc.exe	...orlando\bin	binary of MFC sample application
orlc_an_mod	...orlando\bin	binary of ANSI-C sample application
orlc_an.exe	...orlando\bin	binary of ANSI-C sample application obsolete
orlando.lib	...orlando\lib	Visual-C include library of orlando.dll
orlando.def	...orlando\lib	module definition file if orlando.dll
firmware files	...orlando\firmware	orlando firmware files

2.3 Develop environment settings

Include header file orlando.h. This header file needs orltyps.h and arvtyp.h. If using Microsoft Visual Developer you should add orlando.lib file as input for the Linker. Other develop environments should use orlando.def file.

It is possible to use a non-C compiler, because the orlando.dll has a standard function interface. The orlando.h C header file should be converted to declarations for such C compiler.
orl_java application can be used with Eclipse™ development platform, www.eclipse.org. You need Java Runtime Environment to run orl_java or Eclipse IDE.

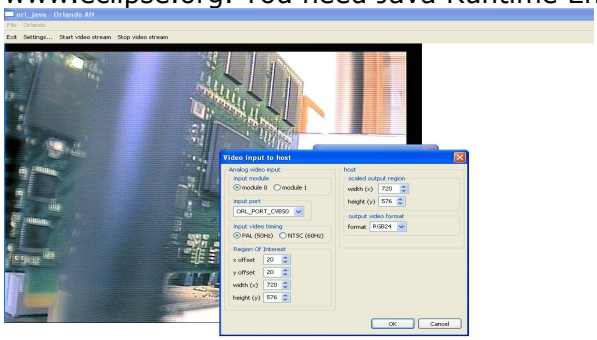


Figure 1 orl_java application, Windows XP

3 Linux SDK

3.1 Before installation

Be sure the Linux kernel source code is installed on the host computer. You can check this to look in directory: /usr/src/linux-<kernelversion>. There should be a symbolic link to this directory named: /usr/src/linux. If this link isn't present, you can create it by

```
ln -s /usr/src/linux-<kernelversion> /usr/src/linux
```

The GNU C-compiler (GCC) should be installed on your computer too.
Check this by:

```
gcc -dumpversion
```

GNU Make (make) should also be installed.
This can be checked by

```
make -v
```

3.2 Installation

Insert the Orlando SDK CD for Linux
Run the install_orlando script
This script copies the Orlando sample application to /opt/arvoo/orlando and the Orlando shared libraries to /usr/lib. The sources of the arvdrv kernel module will be copied to /opt/arvoo/module.
Go to directory /opt/arvoo/module and call make. Now the arvdrv module will be created. Next, call the load script, which loads the arvdrv module in memory.
The kernel module can be unloaded with the unload script.

3.3 Installed files

file name	destination	description
arvdrv.ko	/lib/modules/<linux-version>/misc	kernel module
libarvdrv.so.x.y(*)	/usr/lib	low level library
liborlnd.so.x.y(*)	/usr/lib	orlando library
	/opt/arvoo/..	
arvtyp.h	...include	declarations of basic types
orltyps.h	...orlando/include	orlando typs, enums, structs declarations
orlando.h	...orlando/include	orlando function declarations
orlc_cl source code	...orlando/orlc_cl	source code of orlando ANSI-C sample application for Orlando CL
orlc_an_mod source code	...orlando/orlc_an_mod	source code of orlando ANSI-C sample application for Orlando AN
orlc_an source code	...orlando/orlc_an	source code of orlando ANSI-C sample application for Orlando AN (obsolete)

orlc_cl	...orlando/bin	binary of ANSI-C sample application for Orlando CL
orlc_an_mod	...orlando/bin	binary of ANSI-C sample application for Orlando AN
orlc_an	...orlando/bin	binary of ANSI-C sample application for Orlando AN (obsolete)
orl_java	...orlando/orl_java	source code and JAR file of orl_java Java/SWT application for Orlando CL and AN
firmware files (.arvor)	...orlando/firmware	orlando firmware files

(*) x = major version number, y = minor version number

3.4 Building Orlando application

The compiler should include orlando header files:
orlando.h, which needs orltyps.h and arvtyp.h.
The linker should link to orlando library (liborlnd.so.x.y).

orl_java application can be used with Eclipse™ development platform, www.eclipse.org. You need Java Runtime Environment to run orl_java or Eclipse IDE.

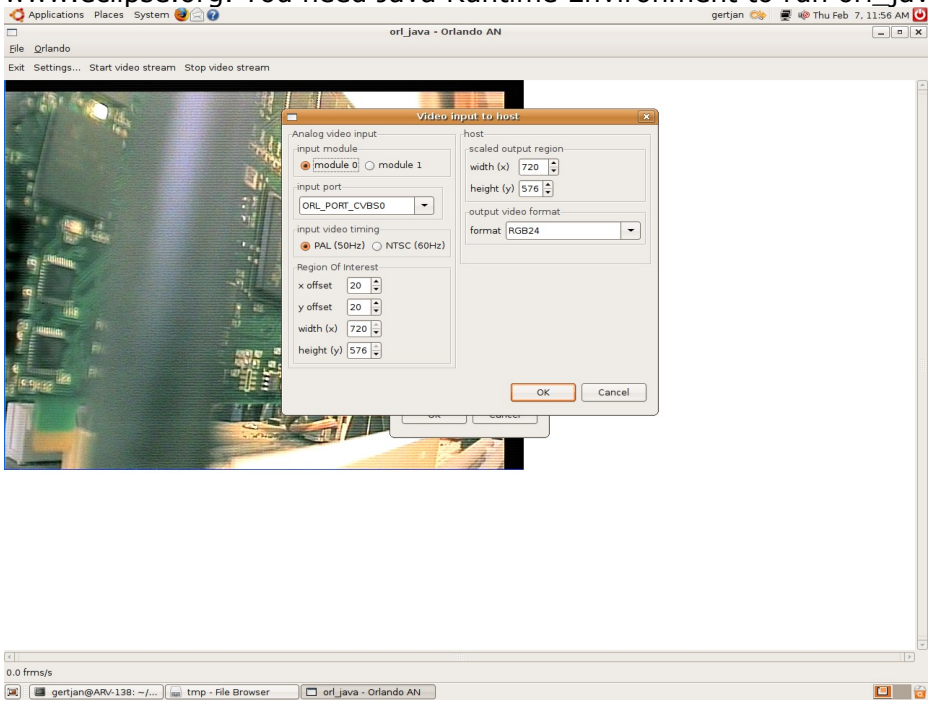


Figure 2 orl_java, Linux GTK

4 QNX6 SDK

4.1 Installation

Insert the Orlando SDK CD for QNX 6.
Run the install_orlando script
This script copies the orlando sample application to /opt/ARVOO/orlando and copies the orlando shared libraries to /lib/dll
The documentation will be left on the installation CD (/doc).

4.2 Installed files

file name	destination	description
libarvdrv.so.x.y(*)	/lib/dll	low level library
liborlnd.so.x.y(*)	/lib/dll	orlando library
	/opt/arvoo/..	
arvtyp.h	...include	declarations of basic types
orltyps.h	...orlando/include	orlando typs, enums, structs declarations
orlando.h	...orlando/include	orlando functions declarations
orlc_cl source code	...orlando/orlc_cl	source code of orlando ANSI-C sample application, for Orlando CL
orlc_an_mod source code	...orlando/orlc_an_mod	source code of orlando ANSI-C sample application, for Orlando AN
orlc_an source code	...orlando/orlc_an	source code of orlando ANSI-C sample application, for Orlando AN (obsolete)
orlc_cl	...orlando/bin	binary of ANSI-C sample application for Orlando CL
orlc_an_mod	...orlando/bin	binary of ANSI-C sample application for Orlando AN
orlc_an	...orlando/bin	binary of ANSI-C sample application for Orlando AN (obsolete)
orl_java	...orlando/orl_java	source code and JAR file of orl_java Java/SWT application for Orlando CL and AN
firmware files (.arvor)	...orlando/firmware	orlando firmware files

(*) x = major version number, y = minor version number

4.3 Building Orlando application

The compiler should include other olando header files: orlando.h, which needs orltyps.h and arvtyp.h.
The linker should link to orlando library (liborlnd.so.x.y).

orl_java application can be used with Eclipse™ development platform, www.eclipse.org.

To run orl_java you need a Java Virtual Machine, eg PVM: Perc Virtual Machine (AONIX) which is used by Eclipse development platform under QNX 6.3.2.

5 Support

When problems occur with the orlando software or hardware, you should contact ARVOO Support Department by e-mail. If possible, generate the orlando log file and attach this file to the e-mail.
Send the problem description with log file to support@arvoo.com.

5.1 Log files for support

The orlando libraries are able to generate log-files. By default, these log-files are not generated. Follow next steps to create the correct log-files:

- enable generating of log files with all messages
- (re)start the application which gives the problem
- close application

These steps are OS-dependent and explained in next paragraphs.

5.1.1 Log files in Windows

Start orlvc sample application
Go to orlando | Settings..., tab Support.

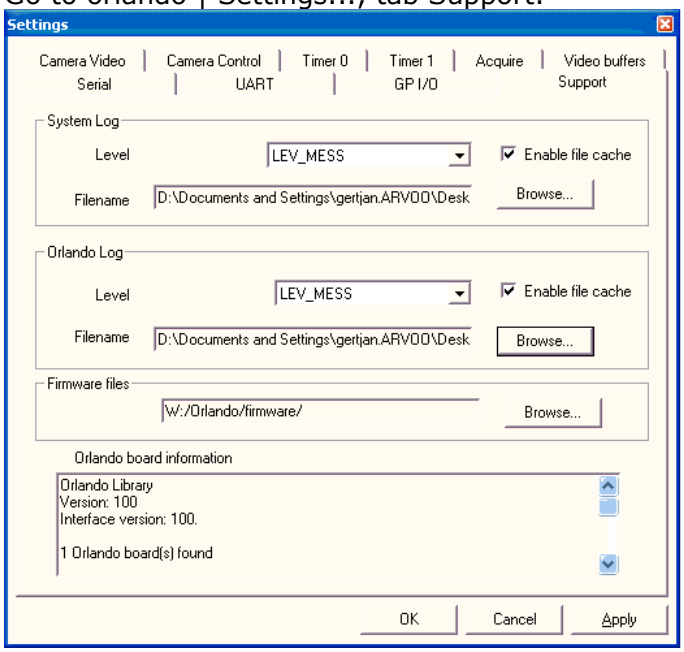


Figure 3: Support tab of Orlando Settings

Set both Levels to LEV_MESS. If the computer crashes when the error occurs, deselect 'Enable file cache'.
Now click OK button and close orlvc.
Now the logging is enabled

Let the problem occur by your own application or the orlando sample application. Close the used application and send the log file (arvdrv.log) to support@arvoo.com with a description of the problem.

If logging isn't necessary, it is recommended to disable this function, because it loads the host processor.

5.1.2 Log files in Linux and QNX

Open file /etc/arvdrv.conf
In section [SysLog] set level to '0', which means 'LEV_MESS': log all.
In section [Orlando] set level to '0' too.
Save and close arvdrv.conf.
Now the logging is enabled.

Start the Orlando application and let the problem occur. Close this application.
Send log file arvdrv.log to support@arvoo.com with a description of the problem.

If logging isn't necessary, it is recommended to disable this function, because it loads the host processor. You should set the level parameters if arvdrv.conf to '4' to obtain this.

6 Programming Guide CL model

This chapter describes the functions of the Orlando CameraLink (CL).

6.1 Overview

Next figure shows the functional block diagram of the Orlando CL (CameraLink).

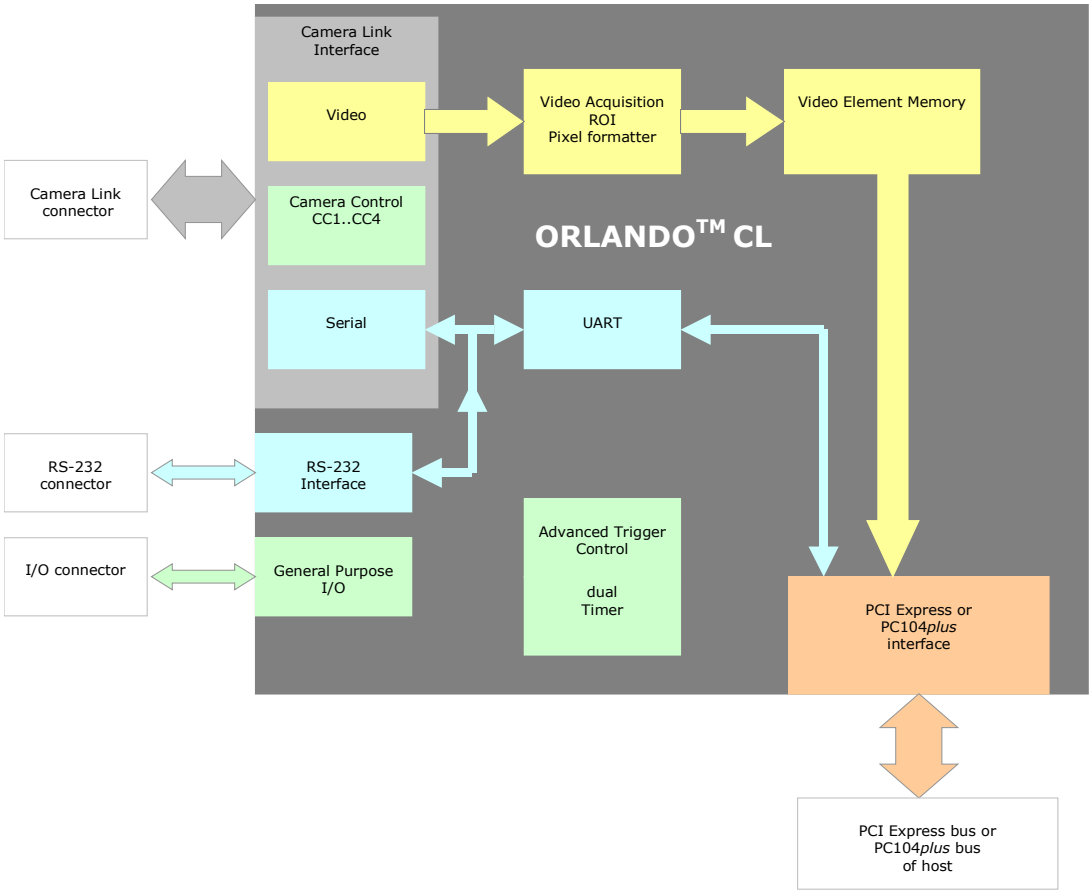


Figure 4: Function block diagram

The Orlando board consists of three functional parts:

- Video acquire and process
- Serial communication
- Advanced Trigger Control and I/O

6.1.1 Video acquire and process

The video data and video timing of the Camera Link camera will be received by the Video module of the Camera Link interface. The video data will be streamed through the Video acquisition, Region Of Interest and the Pixel formatter.
Next, the data will be converted in Video Elements and placed in the on board Video Element Memory.

The Video Elements can be transferred to computer's host memory via the PCI Express or PC104*plus* interface of the Orlando board.

Detailed
Camera Link Video consists of two parts:
- Camera Link Data
- Camera Link Timing

The Orlando is able to acquire Camera Link Base video data. This means that the acquisition module acquires up to three taps, with a Pixel Strobe (PSTRB) of up to 85 MHz (conforms the Camera Link specification).

Camera Link video contains also timing signals:
FVAL: Frame VALid
LVAL: Line VALid
DVAL: Data VALid
Strobe
See Figure 5.

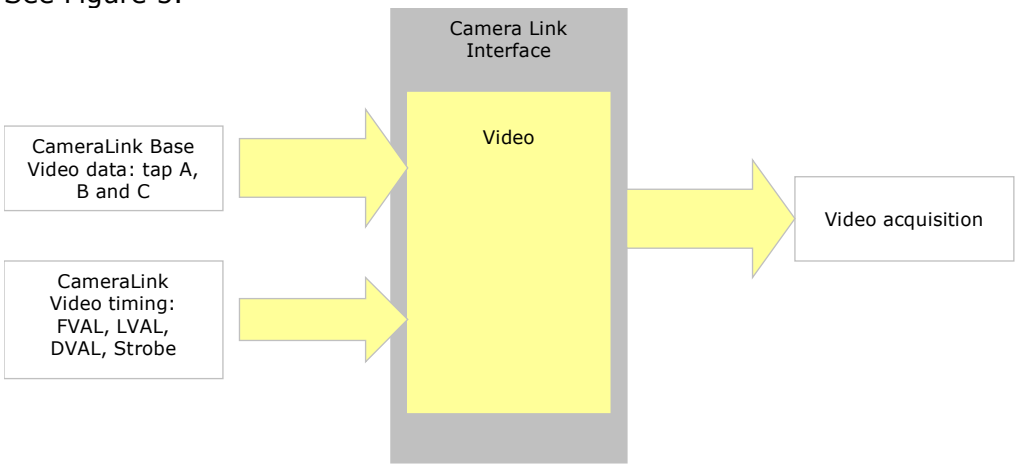


Figure 5 Video acquisition, detailed

6.1.2 Serial communication

Serial communication with the camera can be done by SerTFG (Serial To Frame Grabber) and SerTC (Serial To Camera) signals, which are part of the Camera Link interface, see Figure 6. There are two methods to control the SerTFG and SerTC: by host and by RS-232 connector.
Serial communication by host uses the on board UART. This UART is software controlled.
When using the on board RS-232 connector the serial data will be transferred directly to Camera Link SerTFG/SerTC signals.

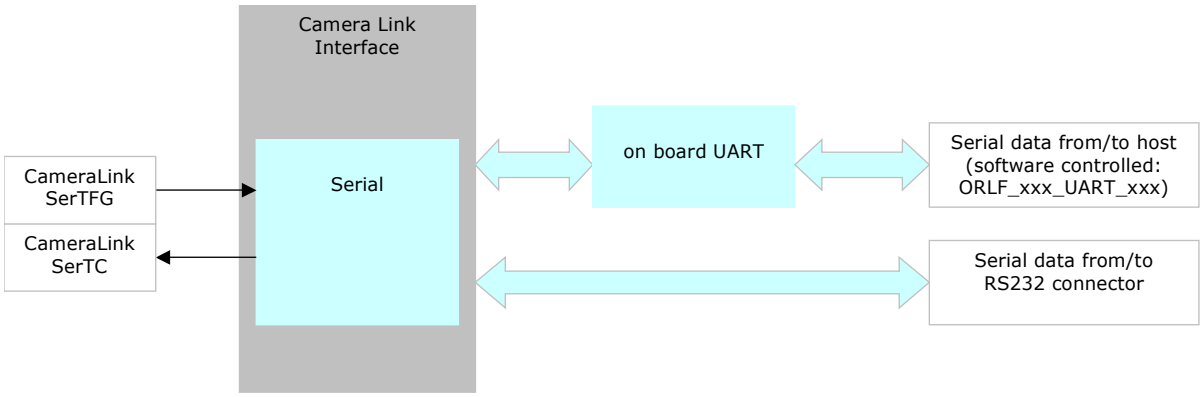


Figure 6 Serial Communication, detailed

6.1.3 Advanced Trigger Control and I/O

The Orlando consists of an Advanced Trigger Control module. This module controls

- Camera Control signals (CC1..CC4)
- General Purpose I/Os
- two on board timers
- start/stop of the Video acquisition

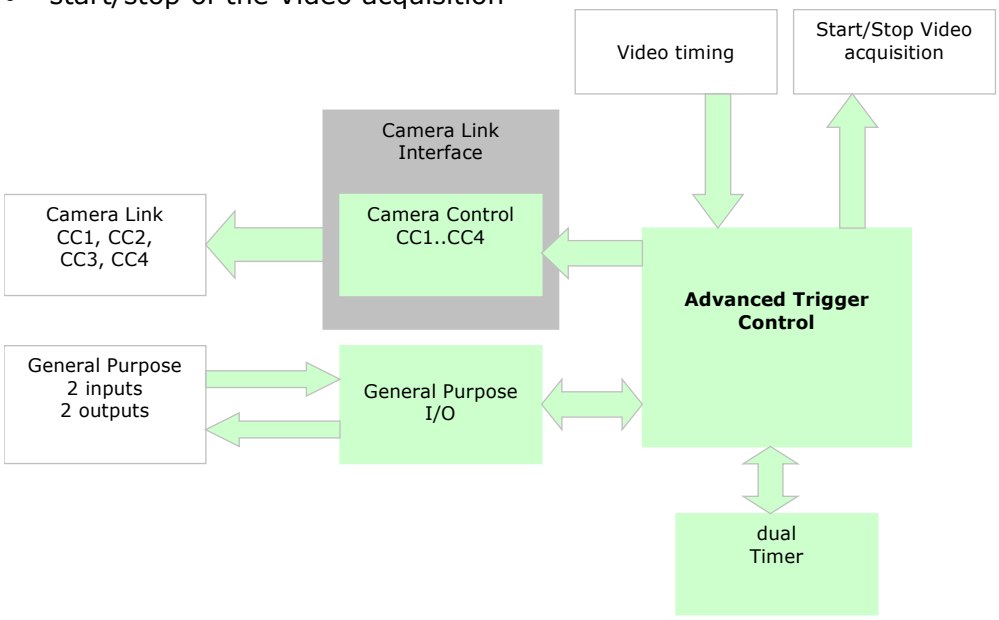


Figure 7 Trigger and I/O, detailed

The various output signals can be triggered by trigger signals.

The trigger signals are:

- General Purpose inputs
- Camera Link FVAL/VAL edges
- Output of the timers
- Trigger by software

Signals that can be triggered are:

- General Purpose outputs
- Camera Link CCx signals

- Start/stop video acquisition
- Start/stop of timers
- Output of the timers
- Output of the timers

7 Programming Guide AN model

7.1 Board interfaces

Next figure shows the external (physical) interfaces of the Orlando AN.

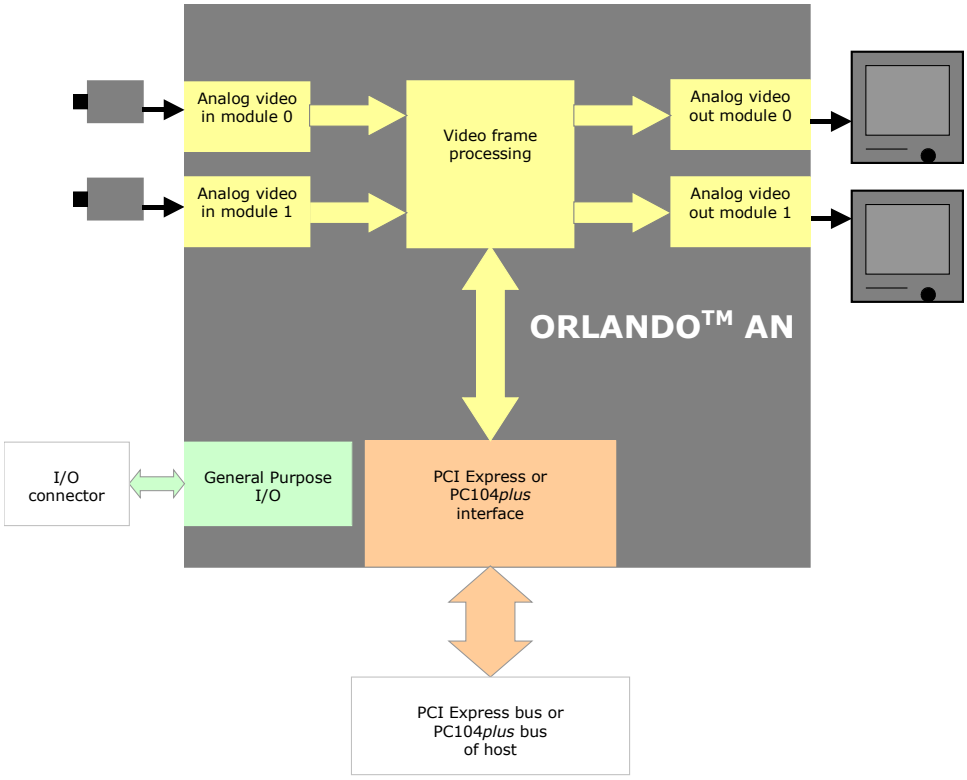


Figure 8 Functional block diagram

For video acquisition the Orlando AN has two the same independent video input modules: Analog video module 0 and 1. These modules acquire analog video and digitize it.

The Orlando has two the same analog video output modules: Analog video out module 0 and 1. These independent modules convert the digital video data to analog video signal. This video signal can be used to display the video frames on a monitor.

The host-Orlando interface is used for:
- board control, e.g. initialize a video stream
- transfer video frames from the video processing module of the Orlando board to the host
- transfer video frames from the host to the video processing module of the Orlando board

General Purpose I/O can be used to control external TTL signals.

7.2 Video modules

7.2.1 Module types

The programming model of the Orlando AN consists of video-modules. In general: a module acquires video, do something with and transfer it to the module output. Modules can be connected to each other to get a complete video stream.
There are six module types:

module type	#modules	enum OrlModule (x: index)	short description
video input	2	ORL_MODULE_VIDEO_INx	Physical module which acquires and digitize analog video
video output	2	ORL_MODULE_VIDEO_OUTx	Physical module which converts digital video data to analog output video
board-to-host	2	ORL_MODULE_BRD2HOSTx	Module for transferring images from board to host
host-to-board	6	ORL_MODULE_HOST2BRDx	Module which receives images from host
video fusion	8	ORL_VMODULE_FUSIONx	Fuses two video sources to one (e.g. Picture in picture)
video overlay	4	ORL_VMODULE_OVL_TEXTx	Generates texts for video overlay

7.2.2 Video stream creation

To create a video stream two or more video modules should be created and connected to each other. The order of module creation depends on the position in the video stream. The module at the beginning of a video stream must first be made. The last created module should be the end of the video stream.
Example: next stream should be created. The red numbers are the creating order.

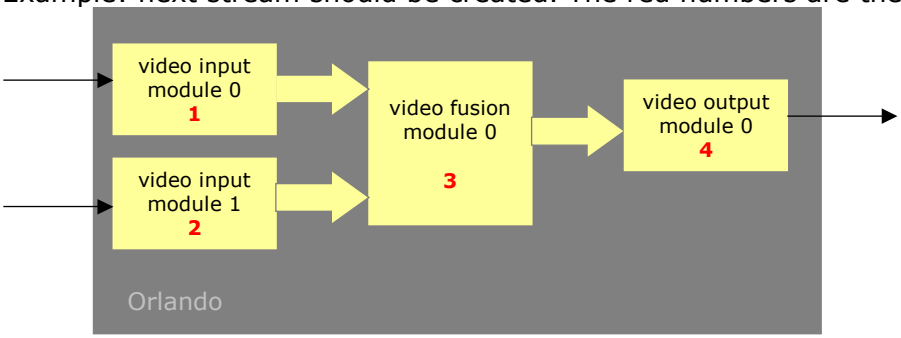


Figure 9 Module creation order

7.2.3 Video stream example 1

Example 1 creates a video stream which results in analog video that consists of scaled video input and a static text at the bottom. Next figure shows the result.

Figure 10 Result of example 1

There are four modules needed:

- video input module: acquires the input video and scale it
- overlay text module: creates the overlay text
- video fusion module: fuses the video and text on the correct position
- video output module: creates analog video of the fusion module result

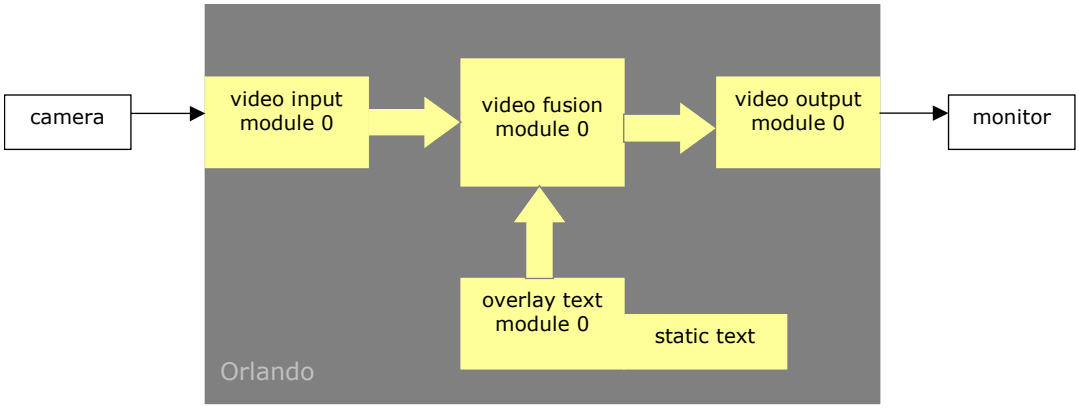


Figure 11 Functional block diagram of example 1

Figure 12 shows the resulting image again. The image consists of the results of the used modules. The black background is the result of the fusion module.

For more information:
Source code: orlc_an_mod project, file vstr_ex1.c.
Create analog input module 88
Create overlay text module on page 100
Create video fusion module on page 96
Create analog output module 93

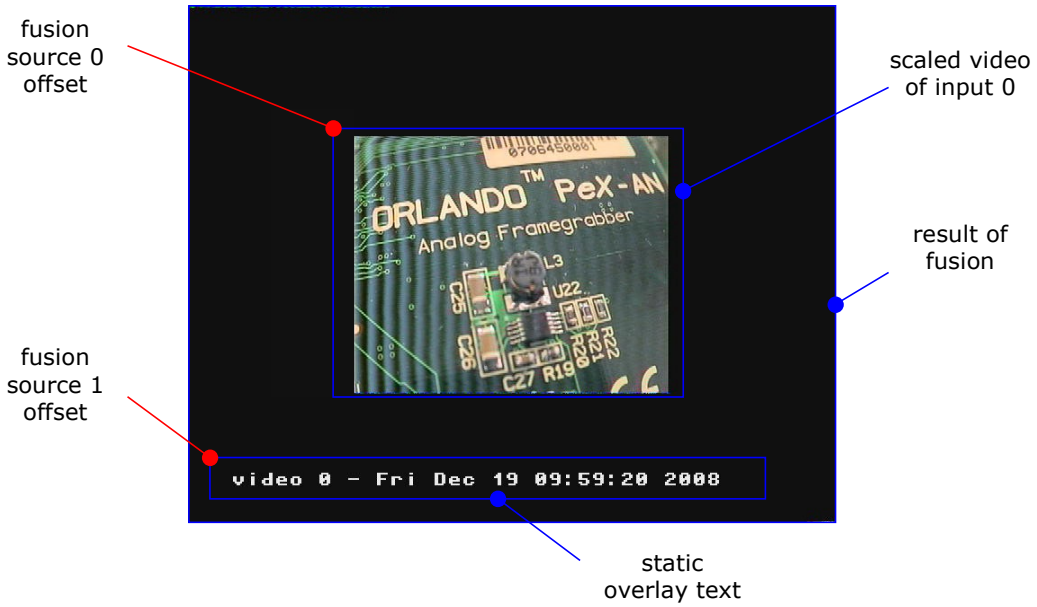


Figure 12 Modules in the resulting image

7.2.4 Video stream example 2

This example shows picture-in-picture. Video input 0 will be fused with scaled input 1. The resulting video is available as analog video and will be transferred to a host buffer. The host buffer will be stored as JPEG file.

Next figure shows the result.

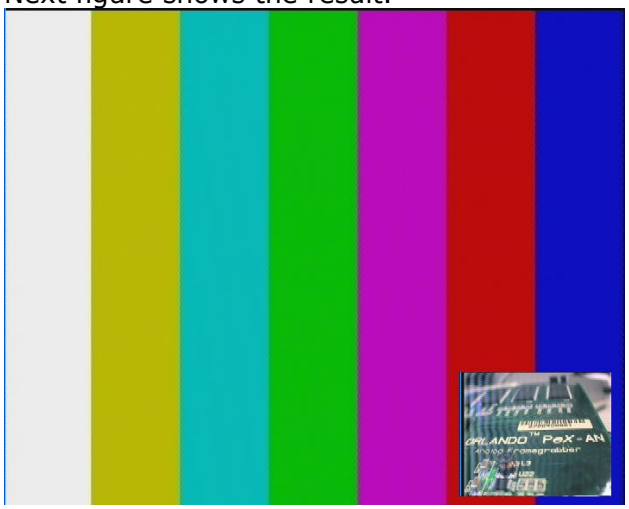


Figure 13 Result of example 2

There are five modules needed:

- video input module 0: acquires the video of camera 0
- video input module 1: acquires the video of camera 1 and scale it down
- video fusion module: fuses the input video 0 and video 1
- video output module: creates analog video of the result of the fusion module
- board-to-host module: used for transferring the result of the fusion module to the host

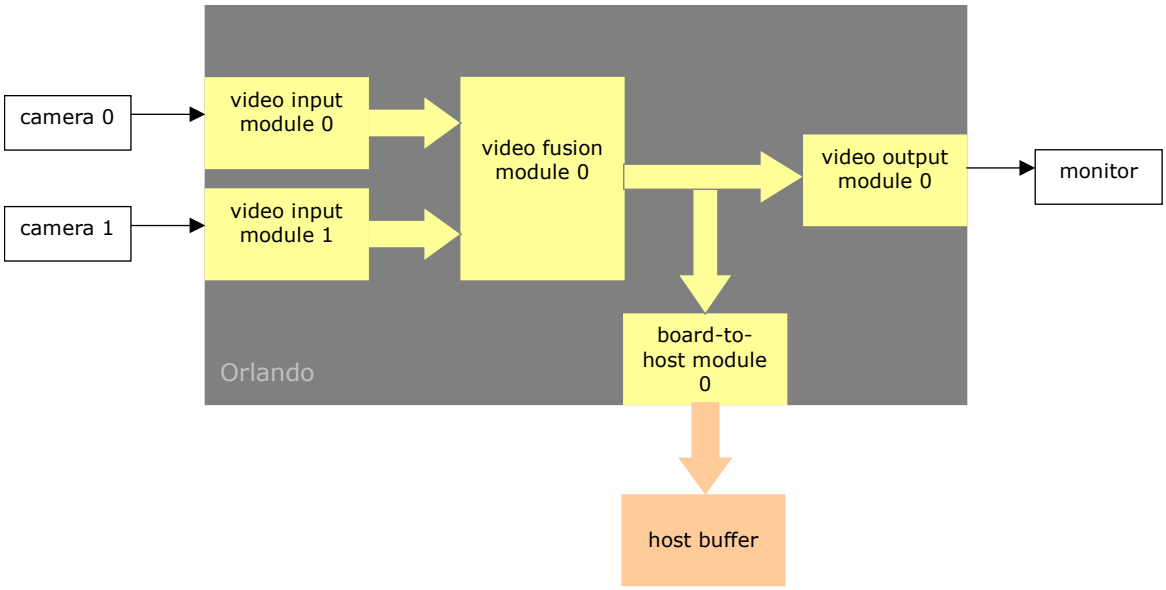


Figure 14 Functional block diagram of example 2

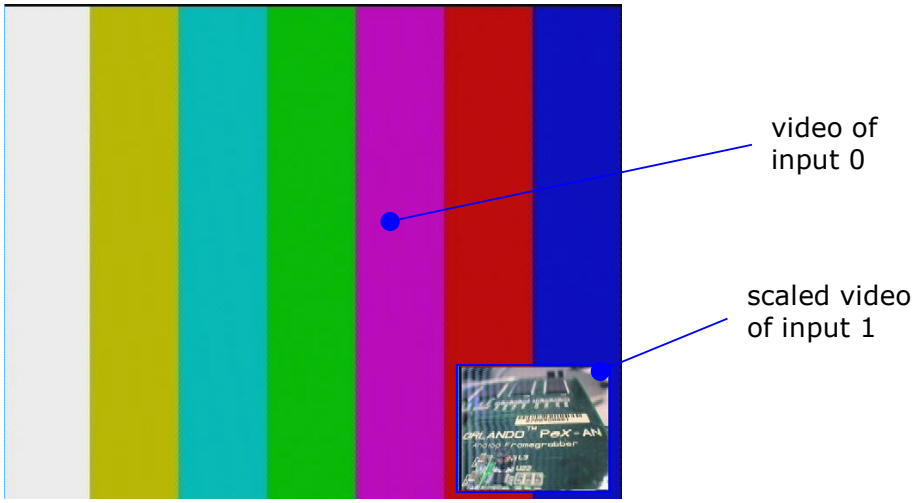


Figure 15 Modules in the resulting image

For more information:
Source code: orlc_an_mod project, file vstr_ex2.c.
Create analog input module 88
Create board-to-host module 108
Allocate host buffer 123
Transfer host frame buffer 130
Initialize/release module interrupts 114
Create video fusion module on page 96
Create analog output module 93

7.2.5 Video stream example 3

Example 3 shows how to fuse analog input video with a bitmap of the host. The result will be available as analog video.

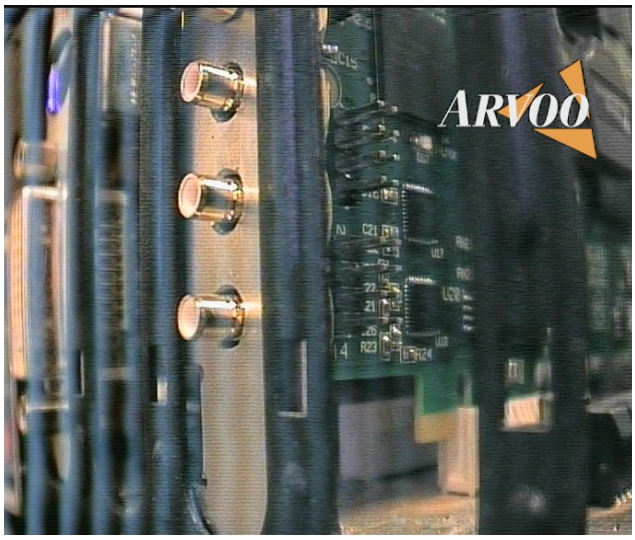


Figure 16 Result of example 3

There are four modules needed:

- video input module 0: acquires the video of camera 0
- host-to-board module 0: receives images of a host buffer
- video fusion module: fuses the input video 0 and host images
- video output module: creates analog video of the result of the fusion module

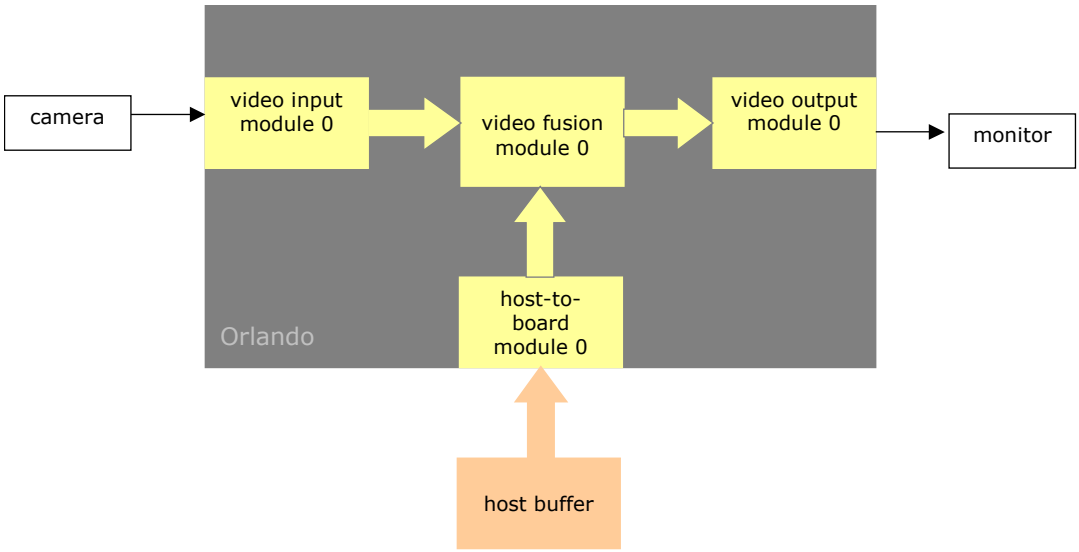


Figure 17 Functional block diagram of example 3

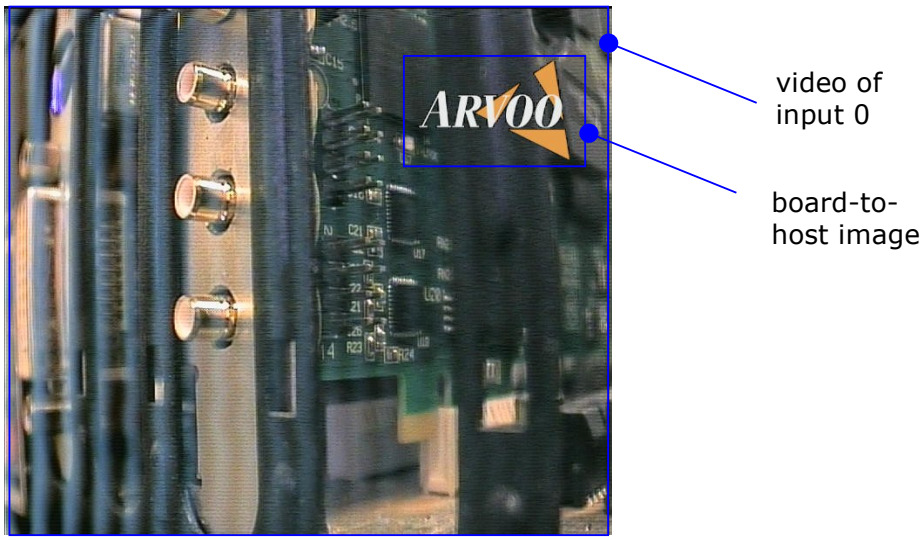


Figure 18 Modules in result

For more information:
Source code: orlc_an_mod project, file vstr_ex3.c.
Create analog input module 88
Create host-to-board module 111
Allocate host buffer 123
Transfer host frame buffer 130
Create video fusion module on page 96
Create analog output module 93

7.3 Video acquisition

Most analog cameras generate odd fields and even fields. The odd fields contain the odd-lines of the video signal and the even fields contain the even-lines of the same video signal. An odd and even pair is called an interlaced *frame*.
A frame grabber acquires these video fields and is able to 'add' an even and odd field to get a full resolution capture.

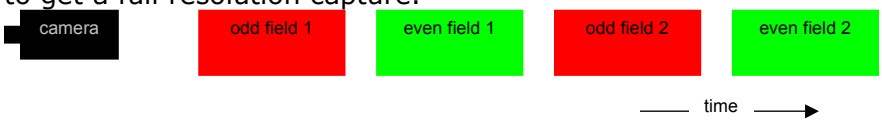


Figure 19 odd and even fields

The Orlando offers three possibilities how these fields are placed in frame buffers (enum OrlFieldArrange):

- ORLF_FLD_INTERLACED: interlaced lines (orginal image)
- ORLF_FLD_DUAL: dual fields
- ORLF_FLD_SINGLE: non-interlaced

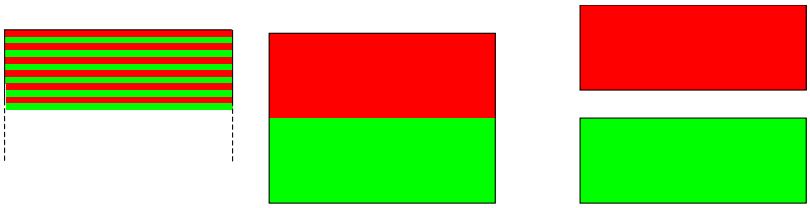


Figure 20 Field arranges

Next table gives the maximum frame dimensions. These values depends on the field arrange and video timing.

	ORL_FLD_INTERLACED/ ORL_FLD_DUAL		ORL_FLD_SINGLE	
maximum	PAL	NSTC	PAL	NSTC
width (x), pixels	720	720	720	720
height (y), lines	576	484	288	282
acquisition rate (Hz)	25	30	50	60

These values can be found in orltyps.h:

```
/* analog video standard dimensions */
#define ORL_PAL_WIDTH 720
#define ORL_PAL_FIELD_HEIGHT 288
#define ORL_PAL_FRAME_HEIGHT (ORL_PAL_FIELD_HEIGHT << 1)

#define ORL_NTSC_WIDTH 720
#define ORL_NTSC_FIELD_HEIGHT 242
#define ORL_NTSC_FRAME_HEIGHT (ORL_NTSC_FIELD_HEIGHT << 1)
```

8 Use of Orlando functions

Next paragraphs describe how to use the Orlando functions. Source code examples are part of the descriptions. In these examples the return value of the orlando functions are most time ignored. In real applications, it is recommended to check these return values!

8.1 Initialize and release of the orlando

Function sequence to initialize a board:

```
OrlOpenLibrary
OrlGetVersion
OrlGetNumBoards
OrlOpen
```

Function sequence for releasing:

```
OrlClose
OrlCloseLibrary
```

Example

```
OrlRet      ret;
SOrlLibVer  ver;
ArvDWORD    bi, numboards;

ORLINITSTRUCT( ver );

/* open the orlando library */
ret = OrlOpenLibrary();

/* retrieve library versions */
ret = OrlGetVersion( &ver );

/* check library version */
if ( ver.intf_ver < ORLANDO_INTERFACE_VERSION )
{
      printf( "Error: incorrect Orlando API-version: %lu.\n"
            "Application needs version: %lu, or higher\n",
            ver.intf_ver, ORLANDO_INTERFACE_VERSION );
      return;
}

/* retrieve the number of Orlando boards found */
ret = OrlGetNumBoards( &numboards );
```

9 Function Reference

9.1 Overview

9.1.1 ArvXXXX Parameter types

type	description
ArvPVOID	void pointer
ArvDWORD	32-bit unsigned value
ArvSDWORD	32-bit signed value
ArvCHAR	8-bit char value
ArvPCHAR	pointer to ArvCHAR

Declared in arvtyp.h.

9.1.2 Return values

The orlando functions returns always an item of OrlRet enum.

OrlRet item	description
ORL_RET_SUCCESS	successful
ORL_RET_ERROR	general error occured
ORL_RET_DRV_ERROR	the low level driver gives an error
ORL_RET_DRV_TIMEOUT	the low level driver detected a timeout
ORL_RET_STRUCT_SIZE_INVALID	the function parameter size is wrong
ORL_RET_PARM	a parameter item isn't correct
ORL_RET_NOT_FOUND	*something* isn't found (depends on function)
ORL_RET_PMEM_ALLOC_FAILED	allocation of physical memory is failed
ORL_RET_NOT_READY	a device isn't ready to process the function
ORL_RET_INSUFFICIENT_BUFFER	the buffer (size) is not sufficient. allocate a (larger) buffer
ORL_RET_DRV_BUSY	the library is not ready to handle the requested function
ORL_RET_NOT_SUPPORTED	a requested conversion isn't supported
ORL_RET_FILE_IO_ERROR	read or writing a file is failed

Declared in orltyps.h.

9.2 Orlando functions

9.2.1 OrlOpenLibrary

```
OrlRet OrlOpenLibrary( void )

_OrlOpenLibrary@0
```

description
Each process (application), which uses the Orlando board, should call this function first, before other orlando functions. This function opens the orlando library and initializes the internal data structures.
After using the orlando library (when your application exits), you should call OrlCloseLibrary.

9.2.2 OrlCloseLibrary

```
OrlRet OrlCloseLibrary( void )

_OrlCloseLibrary@0
```

description
Call this function to close the orlando library. It frees all resources it used. This function should be called once per process.

9.2.3 OrlGetVersion

```
OrlRet OrlGetVersion( SOrlLibVer *ver )

_OrlGetVersion@4
```

parameter
struct SOrlLibVer

item	type	direction
lib_ver	ArvDWORD	out
intf_ver	ArvDWORD	out

description

SOrlLibVer item	description
lib_ver	version of the orlando library
intf_ver	library interface version of the library

Before using the orlando library, you should check the library interface version. Call this function and check whether parameter intf_ver is equal or higher then ORLANDO_INTERFACE_VERSION define. When parameter intf_ver is lower then ORLANDO_INTERFACE_VERSION, you should update the orlando library.

example

```
OrlRet            ret;
SOrlLibVer        ver;

ret = OrlOpenLibrary( );
if ( ret != ORL_RET_SUCCESS )
{
      printf( "Error in OrlOpenLibrary.\n" );
      return;
}

ORLINITSTRUCT( ver );
ret = OrlGetVersion( &ver );
if ( ret != ORL_RET_SUCCESS )
{
      printf( "Error in OrlGetVersion.\n" );
      return;
}
if ( ver.intf_ver < ORLANDO_INTERFACE_VERSION )
{
      printf( "Orlando library function interface version is <%lu>,
      needs <%lu> or higher.\n",
      ver.intf_ver, (ArvDWORD)ORLANDO_INTERFACE_VERSION );
```

```
      return;
}

printf( "Orlando library opened. Version: %lu, Intf version: %lu.\n",
      ver.lib_ver, ver.intf_ver );
```

9.2.4 OrlGetNumBoards

```
OrlRet OrlGetNumBoards( ArvDWORD *num_boards )

OrlGetNumBoards@4
```

parameter

item	type	direction
num_boards	ArvDWORD*	out

description
This function counts the number of orlando boards that are installed in the computer. Parameter num_boards returns the number of boards found. OrlOpenLibrary have to be called before this function.

example

```
ArvDWORD    num;

num = 0UL;

/* orlando library should be opened at this point */

OrlGetNumBoards( &num );
printf( "%lu orlando boards installed.\n", num );
```

9.2.5 OrlGetBoardType

```
OrlRet OrlGetBoardType( ArvDWORD boardnum, OrlBoardType *type )

_OrlGetBoardType@8
```

parameters

item	type	direction
boardnum	ArvDWORD	in
type	OrlBoardType *	out

description

item	description
boardnum	the board number the first board has number '1'
type	type of orlando board • ORL_BOARDT_CL_PCIE: Camera Link PCI-Express • ORL_BOARDT_CL_PC104: CameraLink PC104plus • ORL_BOARDT_ANALOG_PCIE: Analog PCI-Express • ORL_BOARDT_ANALOG_PC104: Analog PC104plus

This function can be used to retrieve the orlando board type before opening the board.

9.2.6 OrlOpen

```
OrlRet OrlOpen( ArvDWORD boardnum, ORL_HANDLE *h )

_OrlOpen@8
```

parameters

item	type	direction
boardnum	ArvDWORD	in
h	ORL_HANDLE*	out

description

item	description
boardnum	Number of orlando board to open. First board has number '1'.
h	Handle of the opened board

This function opens an orlando board. The returned handle (h) should be used by the other functions of the library.

example
see OrlFunc at page 30

9.2.7 OrlClose

```
OrlRet OrlClose( ORL_HANDLE h )

_OrlClose@4
```

parameter

item	type	direction
h	ORL_HANDLE	in

description
This function closes the orlando board and releases all resources used by the board. After calling this function handle h becomes invalid, and shouldn't be used anymore.

example
see OrlFunc at page 30

9.2.8 OrlFunc

```
OrlRet OrlFunc( ORL_HANDLE h, OrlFuncType ft, ArvPVOID p )

_OrlFunc@12
```

parameters

item	type	direction
h	ORL_HANDLE	in
ft	OrlFuncType	in
p	ArvPVOID	in/out

description

item	description
h	handle of the orlando board

ft	orlando function
p	pointer to the orlando function parameter

This function is used to control the orlando settings and to start grabbing frames.
The parameter ft is used to control the type of action that should be performed.
For each function the parameter p is a pointer to a struct. Before using such struct it should be initialized by ORLINITSTRUCT( ), see example below.

See the chapter 10 OrlFunc reference (page 34) for more information.

example

```
ArvDWORD          num_boards;
ORL_HANDLE        h;

/* the orlando library should be opened here */

/* retrieve the numbert of orlando boards */
OrlGetNumBoards( &num_boards );

if ( num_boards == 0 )
{
      printf( "No Orlando boards found.\n" );
      return;
}

/* open the first board */
OrlOpen( 1 /* board num */, &h );

/* here you can use the Orlando handle 'h' in OrlFunc functions */

/* close board */
OrlClose( h );
```

9.2.9 OrlInitStruct

```
void OrlInitStruct( ArvPVOID v, ArvDWORD size )

_OrlInitStruct@8
```

parameters

item	type	direction
v	ArvPVOID	in/out
size	ArvDWORD	in

description

item	description
v	pointer to the struct which should be initialized
size	size of the struct in bytes

The parameter of OrlFunc is a pointer to a struct. This struct should be initialized by this function.
There's also a macro of this function avail:

```
ORLINITSTRUCT(v)
```

example

```
SOrlInit    init;
OrlInitStruct( &init, sizeof( init ) );

/* or by macro: */
ORLINITSTRUCT( init );
```

9.2.10 OrlEnumToStr

```
OrlRet OrlEnumToStr( OrlEnumType etype, ArvSDWORD itemofenum, ArvPCHAR
strbuff, ArvDWORD buffsize )

_OrlEnumToStr@16
```

parameters

item	type	direction
etype	OrlEnumType	in
itemofenum	ArvSDWORD	in
strbuff	ArvPCHAR	in/out
buffsize	ArvDWORD	in

description

item	description
etype	indicates an orlando enum: ORLRET: OrlRet ORLBOARDTYPE: OrlBoardType ORLSERIALSRC: OrlSerialSrc ORLBAUDRATE: OrlBaudRate ORLCAMSCANTYPE: OrlCamScanType ORLCAMCOLORTYPE: OrlCamColorType ORLCAMNUMTAPS: OrlCamNumTaps ORLNBITSPIXEL: OrlNBitsPixel ORLTRIGGERSRC: OrlTriggerSrc ORLLEVELSRC: OrlLevelSrc ORLACQMEMMODE: OrlAcqMemMode ORLINPUTPIN: OrlInputPin ORLOUTPUTPIN: OrlOutputPin ORLEDGE: OrlEdge ORLTIMERID: OrlTimerId ORLSWTRIG: OrlSWTrig ORLBMTYPE: OrlBMType ORLFUNCTYPE: OrlFunc ARVLOGLEV: ArvLogLev ORLMODULE: OrlModule ORLPORT: OrlPort ORLVIDEOTIME: OrlVideoTime ORLVIDEOGAINMODE: OrlVideoGainMode ORLVIDEOBCSMODE: OrlVideoBCSMode ORLFIELDARRANGE: OrlFieldArrange ORLPIXELFORMAT: OrlPixelFormat ORLSTREAMSTATE: OrlStreamState ORLFONTTYPE: OrlFontType ORLFRMANNOTTYPE: OrlFrmAnnotType ORLOVLCOLOR: OrlOvlColor

item	description
	ORLMODSTATE: OrlModState ORLTXTALIGN: OrlTxtAlign ORLANTRIGSRC: OrlAnTrigSrc ORLFUSIONTYPE: OrlFusionType
itemofenum	number of an item in enum indicated by etype
strbuff	buffer to contain the returned string should be allocated by caller
buffsize	size of strbuff in bytes

This function can be used to convert orlando enumeration values to the stringable version. You have to pass the enum type and a value to this function and the string will be returned.

example

```
SOrlBoardInfo     binfo;
ArvCHAR     str[256];

/* retrieve information of orlando */

ORLINITSTRUCT( binfo );
OrlFunc( h, ORLF_GETBOARDINFO, &binfo );

/* convert binfo.btype (OrlBoardType) to string */
OrlEnumToStr( ORLBOARDTYPE, (ArvSDWORD)binfo.btype, strbuff, 256 );

/* to output: "Boardtype: ORL_BOARDT_CL_PCIE" */
printf( "Boardtype: %s\n", strbuff );
```

10 OrlFunc reference

Next chapters describes the various functions of OrlFunc. See also OrlFunc at page 30.

10.1 Orlando CameraLink

board initialization

UART/Serial

video acquisition

video buffering

input/output

timer

10.2 Orlando Analog

board initialization

video modules

overlay & annotation

host buffers

input/output

video streams (obsolete)

10.3 Board initialization

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_INIT`

parameter

struct SOrlInit

item	type	direction
cbfunc	OrlCbFunc	in
cbparm	ArvPVOID	in
opt	ArvDWORD	in

description

item	description
cbfunc	callback function should be of type OrlCbFunc: `ArvDWORD callbackfunction( PSOrlCbParm parm )`
cbparm	user parameter may set to NULL
opt	callback options: • ORL_INITOPT_POST_CB callback function will be called if board will be closed (OrlClose) This can be used to free resources related to the callback function.

This function initializes the orlando board. It also set the orlando callback function OrlCbFunc.

example

see callback function OrlCbFunc, page 38.

10.3.1 callback function OrlCbFunc

```
ArvDWORD (*OrlCbFunc)( PSOrlCbParm parm )
```

parameter
struct SOrlCbParm

item	type	direction
inttype	OrlInterrType	out
int_not_handled_yet	ArvDWORD	out
cbparm	ArvPVOID	out
struct frame_grab		
○ num_interr	ArvDWORD	out
struct frame_host		
○ num_interr	ArvDWORD	out
struct message		
○ mess	ArvDWORD	out
struct clvideo		
○ num_interr	ArvDWORD	out
struct gp0		
○ num_interr	ArvDWORD	out
struct gp1		
○ num_interr	ArvDWORD	out
opt	ArvDWORD	out

description

item	description
inttype	Bitwise parameter which indicates which interrupt(s) are occured • ORL_INT_FRAME_GRAB one Video Element is grabbed and stored in on board memory struct frame_grab • ORL_INT_FRAME_HOST one Video Element is transferred from orlando to a host buffer struct frame_host • ORL_INT_MESSAGE A message of the board. This may be an error struct message • ORL_INT_CLVIDEO a camera is connected or disconnected, or the pixel clock of the camera is changed struct clvideo

item	description
	• ORL_INT_GP0 General Purpose input 0 gives an interrupt, see page 81 struct gp0 • ORL_INT_GP1 General Purpose input 1 gives an interrupt, see page 81 struct gp1
int_not_handled_yet	Number of interrupts that are not handled yet by the callback function
cbparm	User parameter. Initialized by Board initialization, SOrlInit.parm item.
struct frame_grab	
num_interr	number of ORL_INT_FRAME_GRAB interrupts occured after the previous call of this callback function
struct frame_host	
num_interr	number of ORL_INT_FRAME_HOST interrupts occured after the previous call of this callback function
struct message	
mess	Valid if inttype bit ORL_INT_MESSAGE is set bitwise parameter which contains the message code • ORL_ERR_FPGA_VIDFIFO_OV Input Video-FIFO overflow • ORL_ERR_ACQ_VIDFIFO_OV Acquisition Video-FIFO overflow • ORL_ERR_HOST_VIDFIFO_OV Host Video-FIFO overflow • ORL_MESS_ACQ_VIDFIFO_FULL Message indicates the on board buffers are filled
struct clvideo	
num_interr	number of ORL_INT_CLVIDEO interrupts occured after the previous call of this callback function, see Get CameraLink video information 54
struct gp0	
num_interr	number of ORL_INT_FRAME_GP0 interrupts occured after the previous call of this callback function
struct gp1	
num_interr	number of ORL_INT_FRAME_GP1 interrupts occured after the previous call of this callback function
opt	normally '0' is set to ORL_INITOPT_POST_CB if: - ORLF_INIT opt is set to ORL_INITOPT_POST_CB - board is closing (freeing resources)

The callback function is called when the orlando board interrupts the orlando-application. The parameter gives information about the interrupt (inttype). It is recommended to hold the callback function as short as possible.

example

```
ArvDWORD    NumGrabbed = 0;

ArvDWORD CBFunc( PSOrlCbParm parm )
{
```

```
      if ( parm->int_not_handled_yet > 0 )
      {
            printf( "not handled interrupts: %lu.\n",
                                    parm->int_not_handled_yet );
      }
      if ( parm->inttype == ORL_INT_FRAME_GRAB )
      {
            NumGrabbed += parm->frame_grab.num_interr;
            printf( "%lu frames grabbed\n", NumGrabbed );
      }
      else if ( parm->inttype == ORL_INT_FRAME_HOST )
      {
            parm->frame_host
      }
      else if ( parm->inttype == ORL_INT_MESSAGE )
      {
            printf( "Error/Message occured: 0x%x.\n",
                  parm->message.mess );
      }
      else if ( parm->inttype == ORL_INT_CLVIDEO )
      {
            parm->clvideo
      }
      else if ( parm->inttype == ORL_INT_GP0 )
      {
            parm->gp0
      }
      else if ( parm->inttype == ORL_INT_GP1 )
      {
            parm->gp1
      }
} /* CBFunc */



/* init of callback function CBFunc */

SOrlInit    init;
ORLINITSTRUCT( init );

init.cbfunc = CBFunc;
init.parm = NULL;
OrlFunc( h, ORLF_INIT, &init );
```

10.4 Retrieve board information

supported boards

CL	AN
Y	Y

OrlFuncType

`ORLF_GETBOARDINFO`

parameter

struct SOrlBoardInfo

item	type	direction
btype	OrlBoardType	out
board_version	ArvDWORD	out
id	ArvDWORD	out
td	ArvDWORD	out
fhii	ArvDWORD	out
fb	ArvDWORD	out
fsii	ArvDWORD	out
fsiiv	ArvDWORD	out
dsp_id	ArvDWORD	out
dsp_revid	ArvDWORD	out
dhii	ArvDWORD	out
dsii	ArvDWORD	out
dsiiv	ArvDWORD	out
str	ArvPCHAR	in/out
buffsize	ArvDWORD	in/out

description

item	description
• btype	board type • ORL_BOARD_CL_PCIE orlando PeX-CL PCI Express CameraLink Base • ORL_BOARD_CL_PC104 orlando 104-BASE24 PC104*plus* CameraLink Base • ORL_BOARDT_ANALOG_PCIE orlando PeX-AN • ORL_BOARDT_ANALOG_PC104 orlando 104-AN
board_version	version of the PCB 1: Revision A 2: Revision B 3: Revision C etc
dsp_id	0x4: Ti DSP C64x
dsp_revid	0x801: Ti DSP 64xx silicon rev 1
str	string which is the stringable version of this struct

item	description
	should be allocated by user if str is NULL, no string is returned
buffsize	size of str buffer if buffsize is '0' and str is NULL this value will set to the required buffsize
	other struct items are for internal use

example

```
SOrlBoardInfo     info;

ORLINITSTRUCT( info );

/* get the size for str buffer */
OrlFunc( h, ORLF_GETBOARDINFO, &info );

info.str = (ArvPCHAR)malloc( info.buffsize );

/* str buffer will be filled */
OrlFunc( h, ORLF_GETBOARDINFO, &info );

printf( "orlando board: %s", info.str );

free( info.buffsize );
```

10.5 Firmware directory

supported boards

CL	AN
Y	Y

OrlFuncType

`ORLF_GET_FIRMWARE_DIR`
`ORLF_SET_FIRMWARE_DIR`

parameter

struct SOrlFirmwareDir

item	type	direction
dirstr	ArvPCHAR	in/out
buffsize	ArvDWORD	in/out

see also

Firmware programming 44

description

item	description
dirstr	Buffer which contains the directory of the orlando firmware files. This buffer should be allocated by the caller
buffsize	Size (in bytes) of the dirstr buffer. If using ORLF_GET_FIRMWARE_DIR and buffsize is '0' and dirstr is NULL then buffsize will return the number of bytes of the current firmware directory string.

This function should be called to retrieve or change the current firmware directory.
The caller should allocate enough room for buffer dirstr.
Function ORLF_SET_FIRMWARE_DIR changes the firmware directory.
ORLF_GET_FIRMWARE_DIR can be used to retrieve the directory. If using ORLF_GET_FIRMWARE_DIR and buffsize is '0', the buffsize will return the number of bytes of the current firmware directory string.
The default firmware directory is: "../firmware/".

Under Windows this directory is stored in the Registry:
HKCU\Software\ARVOO Engineering\Orlando\fwdir
The value after installation is: "\Program Files\ARVOO\Orlando\firmware\"

Under other OSs this directory is stored in file \opt\arvdrv.conf, section [Orlando], item fwdir.

The default directory is: "/opt/arvoo/orlando/firmware/"

Example

see page 44.

10.6 Firmware programming

supported boards

CL	AN
Y	N

OrlFuncType
ORLF_PROGRAM_FIRMWARE

parameter
struct SOrlProgramFirmware

item	type	direction
opt	ArvDWORD	in

see also
Firmware directory 43

description

item	description
opt	bitwise parameter which contains firmware upload options • ORL_OPT_DSPEMUL_MODE ORL_OPT_ONLY_F used for testing

After opening en initializing of the orlando board the orlando firmware should be uploaded to the board. The orlando library searches in de firmware directory for the firmware files and upload these to the board.

example

```
SOrlFirmwareDir   fwdir;
SOrlProgramFirmware     fwpr;

/* retrieve current firware directory */
ORLINITSTRUCT( fwdir );
OrlFunc( h, ORLF_GET_FIRMWARE_DIR, &fwdir );

fwdir.dirstr = (ArvPCHAR)malloc( fwdir.buffsize );
OrlFunc( h, ORLF_GET_FIRMWARE_DIR, &fwdir );
printf( "firware directory: %s\n", fwdir.dirstr );
free( fwdir.dirstr );

/* program firmware */
OrlFunc( h, ORLF_PROGRAM_FIRMWARE, &fwpr );
```

10.7 Change logging

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_GET_LOG`
`ORLF_SET_LOG`

parameter

struct SOrlLog

item	type	direction
system	ArvDWORD	in
filename	ArvCHAR[MAX_LOG_FILENAMELENGTH+1]	in/out
level	ArvLogLev	in/out
flags	ArvDWORD	in/out

description

item	description
system	• log type: • 0: orlando library logging 1: system (low level) logging
filename	buffer which contains the log filename, eg: "orlando.log"
level	log threshold • LEV_MESS all information (needed for support by ARVOO) • LEV_WARN log warnings and errors • LEV_ERR log errors • LEV_LOG_OFF logging disabled
flags	Bitwise parameter. No flags in use: 0 • ARV_LOGFLAG_DISABLE_FLUSH Normally the log files doesn't use file caches. When set this flag the file caches will be used. This gives beter performance.

These functions control the use of generating the orlando log file. There are two log-types:

- system logging: logs low level functions
- orlando logging: logs orlando library functions

The filename of system and orlando logging should be the same.

Under normal circumstances, you should set the log level to LEV_LOG_OFF. When you need to create a log file for ARVOO support you should change the log level to LEV_MESS and send the log file to support@arvoo.com.

10.8 Set serial route

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_SET_SERIAL_SETTINGS`

parameter

struct SOrlSerialSett

item	type	direction
cl_sertc	OrlSerialSrc	in
uart_to_host	OrlSerialSrc	in
rs232_out	OrlSerialSrc	in

see also

Read/write UART 49

description

item	description
cl_sertc	source of CameraLink Serial To Camera (SerTC) signal • ORL_SERSRC_CL_SERTFG data from Camera To Frame Grabber • ORL_SERSRC_UART_TFG data from host To Frame Grabber • ORL_SERSRC_RS232_TFG data from RS232 To Frame Grabber • ORL_SERSRC_NONE no data source
uart_to_host	source of Uart Rx signal
rs232_out	source of RS232 Tx signal

The orlando board has three serial 'outputs':

- Camera Link SerTC (serial data to camera)
- On board UART RX (serial data to host, function ORLF_READ_UART)
- RS-232 connector, TX pin (serial data to external device)

This function should be called to select a serial source for these outputs, see next figure.

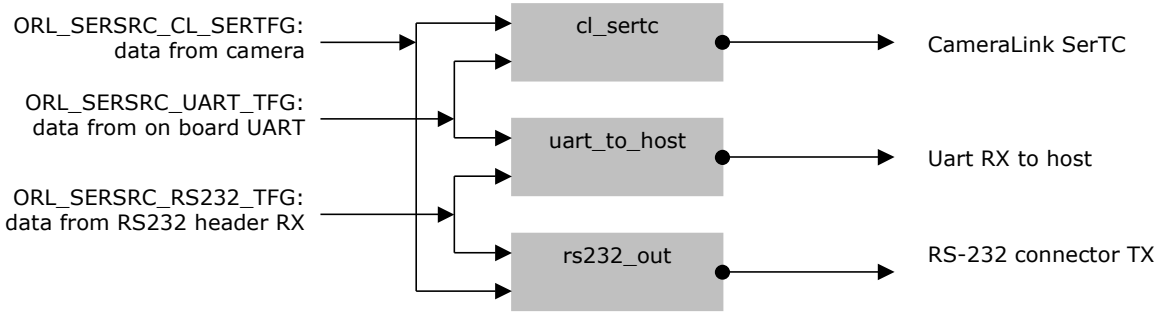


Figure 21: Routing of serial communication

Function ORLF_SET_SERIAL_SETTINGS selects the source (=serial input) of the serial outputs.

serial sources (OrlSerialSrc):
- ORL_SERSRC_CL_SERTFG: Camera Link SerTFG (serial data from camera)
- ORL_SERSRC_UART_TFG: on board UART data from host (by function ORLF_WRITE_UART)
- ORL_SERSRC_RS232_TFG: serial data from RS232 connector
- ORL_SERSRC_NONE can be used if an serial output isn't used

example

```
/* in this example
- UART Tx is connected to CameraLink SerTC output
- CameraLink input (SerTFG) is connected to UART Rx
- RS232 port is disconnected
*/

SOrlSerialSett    ser;

ORLINITSTRUCT( ser );
ser.cl_sertc = ORL_SERSRC_UART_TFG;
ser.uart_to_host = ORL_SERSRC_CL_SERTFG;
ser.rs232_out = ORL_SERSRC_NONE;

OrlFunc( h, ORLF_SET_SERIAL_SETTINGS, &ser );
```

10.9 UART settings

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_SET_UART_SETTINGS`

parameter

struct SOrlUartSett

item	type	direction
baudrate	OrlBaudRate	in

see also

Set serial route 46
Read/write UART 49

description

item	description
baudrate	baudrate of the on board UART: • ORL_BR_9600: 9600 baud • ORL_BR_19200: 19k2 baud • ORL_BR_38400: 38k4 baud • ORL_BR_57600: 57k6 baud • ORL_BR_115200: 115k2 baud • ORL_BR_230400: 230k4 baud • ORL_BR_460800: 460k8 baud • ORL_BR_921600: 921k6 baud

The baudrate of the on board UART of the orlando can be changed by this function.
The other RS-232 protocol settings are static:

- 8 bits per char
- No parity
- 1 Stop bit
- No handshaking

These settings are conforming the Camera Link specification.

10.10 Read/write UART

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_READ_UART`
`ORLF_WRITE_UART`

parameter

struct SOrlUartReadWrite

item	type	direction
buffer	ArvPVOID	in/out
nbytes_to_trans	ArvDWORD	in
nbytes_trans	ArvDWORD	out
timeout	ArvDWORD	in
err	ArvDWORD	out

see also

Set serial route 46
UART settings 48
Retrieve UART bytes received 51
Purge UART buffers 53

description

item	description
buffer	pointer to data buffer ORLF_WRITE_UART transmit data ORLF_READ_UART received data
nbytes_to_trans	number of bytes to transfer (request)
nbytes_trans	number of bytes that was transmitted/received
timeout	timeout of function in milliseconds
err	UART bitwise error code • ORL_UART_ERR_NONE No error occured • ORL_UART_ERR_PE RS232 parity error • ORL_UART_ERR_FE RS232 frame error • ORL_UART_ERR_OR_HW Hardware buffer overrun • ORL_UART_ERR_OR_SW Software buffer overrun • ORL_UART_ERR_TO Timeout • ORL_UART_ERR_DRV Error in UART command processing

This function should be used to read or write data to the on board UART. The UART routes the data to a device, which is selected by: Set serial route 46.

Item buffer should be initialized by the caller. It should be large enough to contain 'nbytes_to_trans' data.
If item err contains an error, the return value of the function isn't ORL_RET_SUCCESS.

example

```
/*    example of write data to UART (UART Tx)
receive UART example: see Retrieve UART bytes received 51
*/

/* UART will transmit "Hello, world!" */
OrlRet                  ret;
SOrlUartReadWrite       wr;
ArvBYTE                 buff[32];

ORLINITSTRUCT( wr );
strcpy( buff, "Hello, world!" );
wr.buffer = buff;
wr.nbytes_to_trans = strlen( buff );

ret = OrlFunc( h, ORLF_WRITE_UART, &wr );
if ( ret == ORL_RET_SUCCESS )
{
      printf( "%lu bytes transmitted.\n", wr.nbytes_trans );
}
```

10.11 Retrieve UART bytes received

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_UART_GET_BYTES_AVAIL`

parameter

struct SOrlUartReadWrite

item	type	direction
buffer	ArvPVOID	-
nbytes_to_trans	ArvDWORD	-
nbytes_trans	ArvDWORD	out
timeout	ArvDWORD	-
err	ArvDWORD	-

see also

Read/write UART 49
Purge UART buffers 53

description

item	description
buffer	not used
nbytes_to_trans	not used
nbytes_trans	Number of bytes which in the internal UART receive buffer
timeout	not used
err	• not used

This function should be called to retrieve the number of bytes received by the on board UART buffer. The serial data can be retrieved by ORLF_READ_UART function.

example

```
SOrlUartReadWrite       avail, rd;
ORLINITSTRUCT( avail );
ORLINITSTRUCT( rd );
rd.timeout = 50; /* milliseconds */

/* retrieve the num of bytes which are received already*/
OrlFunc( h, ORLF_UART_GET_BYTES_AVAIL, &avail );

if ( avail.nbytes_trans > 0 )
{
/* create buffer to store rx data */
rd.buffer = malloc( avail.nbytes_trans );

/* read these bytes */
rd.nbytes_to_trans = avail.nbytes_trans;
OrlFunc( h, ORLF_READ_UART, &rd );
```

```
printf( "%lu bytes received.\n", rd.nbytes_trans );

/* free receive buffer */
free( rd.buffer );
rd.buffer = NULL;
}
```

10.12 Purge UART buffers

supported boards

CL	AN
Y	N

OrlFuncType
`ORLF_PURGE_UART`

parameter
struct SOrlNullStruct

description
This function purges the read and write buffers of the on board UART.

example
```
SOrlNullStruct    null;
ORLINITSTRUCT( null );

OrlFunc( h, ORLF_PURGE_UART, &null );
```

10.13 Get CameraLink video information

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_GET_CLVIDEO_INFO`

parameter

struct SOrlCLVideoInfo

item	type	direction
clk_100khz	ArvDWORD	out
nstrb	ArvDWORD	out
nlines	ArvDWORD	out

see also

callback function OrlCbFunc 38

description

item	description
clk_100khz	Contains the pixel clock of the camera in 100KHz units. A value of 0 means: no pixel clock detected • '457' means: 45.7 MHz
nstrb	• The detected number of CameraLink Strobes per line. You should multiply this value with the number of taps to get the number of pixels per line
nlines	• The detected number of lines per frame

This function gives timing information of Camera Link video acquired by the orlando. This information is retrieved from the Camera Link interface of the Orlando, before Region Of Interest, pixel formatting etc.

10.14 Initialize video acquisition

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_INIT_VIDEO_ACQ`

parameter

struct SOrlVideoAcqSett

item	type	direction
cam_scantype	OrlCamScanType	in
cam_coltype	OrlCamColorType	in
cam_ntaps	OrlCamNumTaps	in
cam_nbits_pixel	OrlNBitsPixel	in
roi	SOrlRoi struct	in
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in/out
○ height	ArvDWORD	in
start_trig	OrlTriggerSrc	in
stop_trig	OrlTriggerSrc	in
flags	ArvDWORD	in
acq_bytes_line	ArvDWORD	out
buff_size_pages	ArvDWORD	out

see also

Get CameraLink video information 54
Initialize video buffers 59
Start/stop video acquisition 64

description

item	description
cam_scantype	Scanning type of Camera Link camera: • ORL_CAM_SCANTYPE_LINE line scan • ORL_CAM_SCANTYPE_AREA area scan
cam_coltype	Color type of camera: • ORL_CAM_COLORTYPE_MONOCHROME monochrome camera (black/white) • ORL_CAM_COLORTYPE_RGB RGB camera if using ORL_CAM_COLORTYPE_RGB the cam_ntaps should be ORL_CAM_NTAP_1 and cam_nbits_pixel should be ORL_NBITS_PIXEL_RGB24

item	description
cam_ntaps	Number of Camera Link taps (channels) of camera • ORL_CAM_NTAP_1 one tap (one pixel per Strobe) • ORL_CAM_NTAP_2 two taps (two pixels per Strobe) • ORL_CAM_NTAP_3 three taps (three pixels per Strobe)
cam_nbits_pixel	Number of bits per pixel of camera • ORL_NBITS_PIXEL_8 8 bit/pixel • ORL_NBITS_PIXEL_10 10 bit/pixel • ORL_NBITS_PIXEL_12 12 bit/pixel • ORL_NBITS_PIXEL_14 14 bit/pixel • ORL_NBITS_PIXEL_16 16 bit/pixel • ORL_NBITS_PIXEL_RGB24 24 bit/pixel (RGB)
roi	Region of Interest
○ xoffs	Number of pixels to skip (horizontal) Max: ORL_MAX_ROI_X = 65520
○ yoffs	Number of lines to skip (vertical) Max: ORL_MAX_ROI_Y = 65520
○ width	Number of valid pixels per line to acquire (horizontal) Max: ORL_MAX_ROI_W = 65534
○ height	Number of valid lines per frame to acquire (vertical) Max: ORL_MAX_ROI_H = 65534

item	description
start_trig	Start trigger of video acquisition • ORL_TRIGSRC_SW No trigger • ORL_TRIGSRC_LVAL_RISING • ORL_TRIGSRC_LVAL_FALLING Rising/falling edge of Camera Link LVAL signal • ORL_TRIGSRC_FVAL_RISING • ORL_TRIGSRC_FVAL_FALLING Rising/falling edge of Camera Link FVAL signal • ORL_TRIGSRC_TIMER0_RISING • ORL_TRIGSRC_TIMER0_FALLING Rising/falling edge of Timer 0 output • ORL_TRIGSRC_TIMER1_RISING • ORL_TRIGSRC_TIMER1_FALLING Rising/falling edge of Timer 0 output • ORL_TRIGSRC_GPIN0_RISING • ORL_TRIGSRC_GPIN0_FALLING Rising/falling edge of General Purpose 0 input • ORL_TRIGSRC_GPIN1_RISING • ORL_TRIGSRC_GPIN1_FALLING Rising/falling edge of General Purpose 1 input
stop_trig	same triggers as start_trig, and: • ORL_TRIGSRC_END_OF_ROI Acquisitions stops when the end of the Region Of Interest is reached stop_trig should be not equal to start_trig
flags	acquisition bitwise flags • ORL_ACQ_FLAG_USE_DVAL If set the video acquisition acquires the CameraLink DVAL signal of the camera. Otherwise DVAL signal will be ignored • ORL_ACQ_FLAG_AUTO_PITCH If set the roi.width variable is changed to get a line length of a multiply of 4 bytes. This is needed for displaying captures or saving as BMP format
acq_bytes_line	Number of bytes per line
buff_size_pages	Number of memory-pages (4096 bytes) needed for a complete video frame

This function should be called to initialize the Camera Link video acquisition:

- Camera Link video format
- Region Of Interest
- Start and/or stop trigger

This function may not be called if the orlando board is acquiring video data.

Next cam_coltype/cam_ntaps/cam_nbits_pixel are possible (Camera Link Base):

cam_coltype	cam_ntaps	cam_nbits
ORL_CAM_COLORTYPE_MONOCHROME	ORL_CAM_NTAP_1	ORL_NBITS_PIXEL_8

		ORL_NBITS_PIXEL_10
		ORL_NBITS_PIXEL_12
		ORL_NBITS_PIXEL_14
		ORL_NBITS_PIXEL_16
	ORL_CAM_NTAP_2	ORL_NBITS_PIXEL_8
		ORL_NBITS_PIXEL_10
		ORL_NBITS_PIXEL_12
	ORL_CAM_NTAP_3	ORL_NBITS_PIXEL_8
ORL_CAM_COLORTYPE_RGB	ORL_CAM_NTAP_1	ORL_NBITS_PIXEL_RGB24

example

```
SOrlVideoAcqSett  acq;
ORLINITSTRUCT( acq );

/* initialize for camera:
area scan, monochrome
10 bit per pixel, 2 taps
1024 x 1024 (w x h)
triggers not used
ignore DVAL signal
*/
acq.cam_scantype = ORL_CAM_SCANTYPE_AREA;
acq.cam_coltype = ORL_CAM_COLORTYPE_MONOCHROME;
acq.cam_nbits_pixel = ORL_NBITS_PIXEL_10;
acq.cam_ntaps = ORL_CAM_NTAP_2;
acq.roi.xoffs = 0UL;
acq.roi.yoffs = 0UL;
acq.roi.width = 1024;
acq.roi.height = 1024;
acq.start_trig = ORL_TRIGSRC_SW;
acq.stop_trig = ORL_TRIGSRC_SW;
acq.flags = 0UL;

OrlFunc( h, ORLF_INIT_VIDEO_ACQ, &acq );

printf( "each line holds %lu bytes\n", acq.acq_bytes_line );
printf( "a complete frame uses %lu pages (= %lu bytes)\n",
      acq.buff_size_pages, acq.buff_size_pages * ORL_PAGE_SIZE_BYTES );
```

10.15 Initialize video buffers

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_INIT_VIDEO_BUFF`

parameter

struct SOrlVideoBuffSett

item	type	direction
vid_elmnnt_size_pages	ArvDWORD	in
acq_fifo_depth_ve	ArvDWORD	in
acq_fifo_thresh_ve	ArvDWORD	in
acq_fifo_mode	OrlAcqMemMode	in
hbuff_elmnt_size_pages	ArvDWORD	in
hbuff_size_ve	ArvDWORD	in
host_fifo_depth_hbuffs	ArvDWORD	in

see also

SW trig ACQMEM_CIRCULAR 63
callback function OrlCbFunc 38

description

item	description
vid_elmnnt_size_pages	The size of a Video Element in memory-pages. One page is 4096 bytes (= ORL_PAGE_SIZE_BYTES)
acq_fifo_depth_ve	Acquisition FIFO depth in Video Elements
acq_fifo_thresh_ve	Acquisition FIFO threshold in Video Elements number of VE which are hold in FIFO
acq_fifo_mode	Acquisition buffer mode: • ORL_ACQMEM_FIFO Buffer is ordered as a FIFO • ORL_ACQMEM_CIRCULAR_STOP_AT_SWTRIG Buffer is ordered as a Circular buffer. After SWTrig all empty buffers will be filled • ORL_ACQMEM_CIRCULAR_STOP_AT_GP0 • ORL_ACQMEM_CIRCULAR_STOP_AT_GP1
hbuff_elmnt_size_pages	Size of a Video Element at host (number of pages)
hbuff_size_ve	Size of a host buffer in Video Elements
host_fifo_depth_hbuffs	Size of host FIFO in number of host buffers

This function should be called to initialize the on board video buffers and the buffers in system memory of the host computer.

Each on board buffer is called a Video Element (VE). A VE contains data of one completely video frame or a part of video frame.

The on board VEs will be used as FIFO or ring buffer. This specified by acq_fifo_mode. In the ORL_ACQMEM_FIFO mode, the VEs works like a FIFO memory. Acquired VE will be put in the FIFO. If there are more then 'acq_fifo_thresh_ve' in the FIFO then the oldest VE will be transferred to the host buffer. The FIFO depth is given by item acq_fifo_depth_ve, see Figure 22.

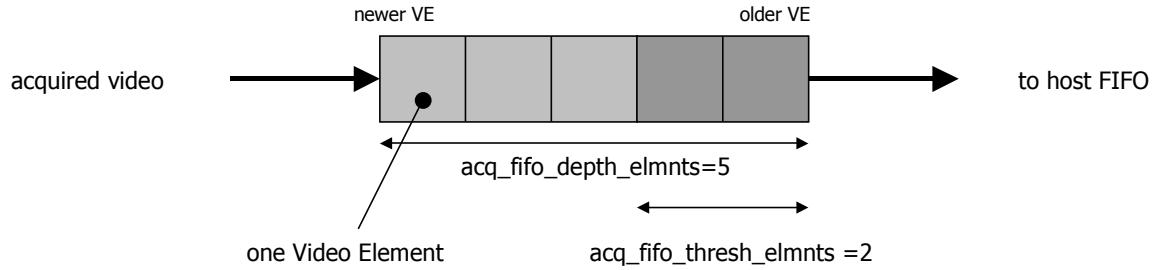


Figure 22 ORL_ACQMEM_FIFO mode

In the ORL_ACQMEM_CIRCULAR_xxx mode, the VEs are ordered in as a ring. The ring consists of acq_fifo_depth_ve buffers, see Figure 23. The acquired VE will overwrite the oldest VE in the ring and so on. When acquiring video, no video data is transferred to the host.
After circular stop trigger, defined by acq_fifo_mode, the ring will hold acq_fifo_thresh_ve VEs and fill the rest of the ring with VE after the moment of triggering, see Figure 24.

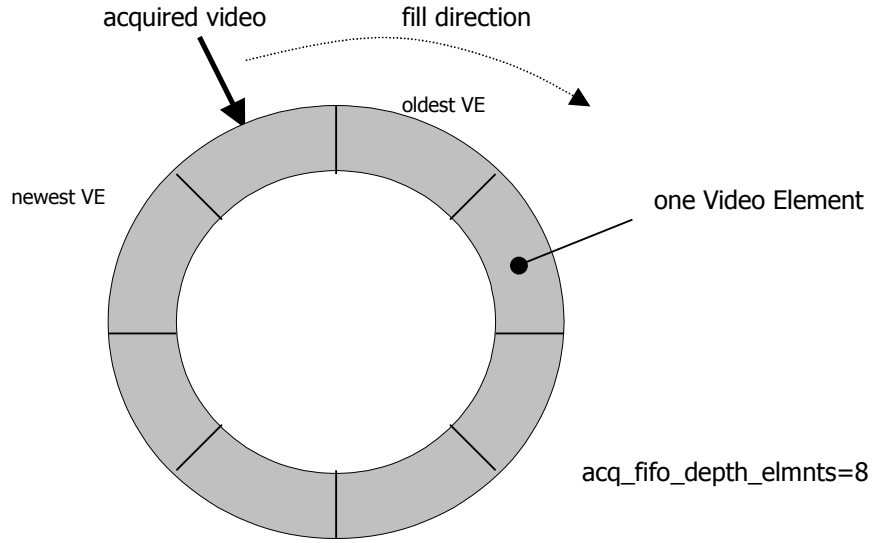


Figure 23 ORL_ACQMEM_CIRCULAR_xxx mode

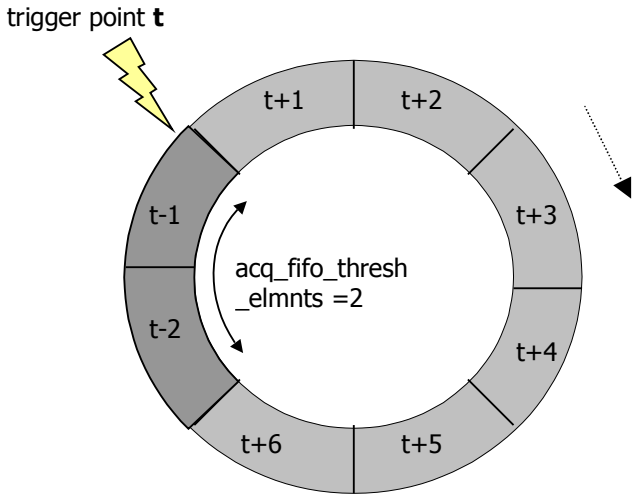


Figure 24 ORL_ACQMEM_CIRCULAR_xxx after stop trigger

There are three circular stop trigger types (indicated by acq_fifo_mode):

- ORL_ACQMEM_CIRCULAR_STOP_AT_SWTRIG:
  triggering by function ORLF_ACQ_STOP_SWTRIG, see SW trig ACQMEM_CIRCULAR, 63.
- ORL_ACQMEM_CIRCULAR_STOP_AT_GP0:
  Triggering by General Purpose 0 interrupt, see Set input interrupt 81
- ORL_ACQMEM_CIRCULAR_STOP_AT_GP1:
  Triggering by General Purpose 1 interrupt, see Set input interrupt 81

Items hbuff_elmnt_size_pages, hbuff_size_ve and host_fifo_depth_hbuffs controls the size of the video FIFO at host.
Item hbuff_elmnt_size_pages is the size of a host VE in pages. This value may the same as vid_elmnnt_size_pages.
Item hbuff_size_ve is the size of one host buffer, given in VE.
Item host_fifo_depth_hbuffs is the depth of the FIFO in host buffers. See Figure 25.

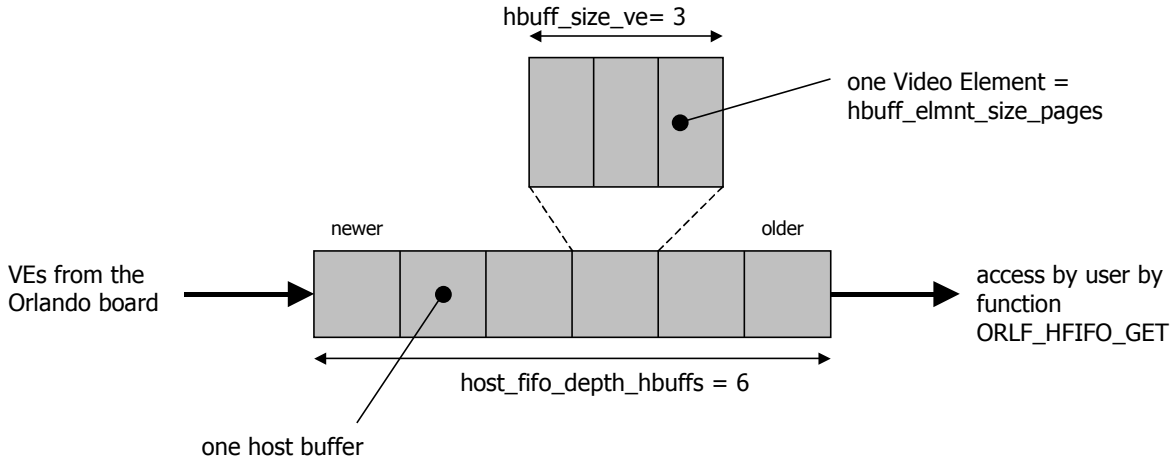


Figure 25 Host FIFO

example

```
/* Note: buff_size_pages is retrieved of example Initialize video
acquisition 55. This is the size of a complete frame in pages.

The on board VEs are ordered as a FIFO with a depth of 20 VEs
The Host FIFO will have a depth of 20 host buffers
*/

SOrlVideoBuffSett       buffsett;
ArvDWORD                buff_size_pages;/* see note above */

ORLINITSTRUCT( buffsett );

/* size of a VE: */
buffsett.vid_elmnt_size_pages = buff_size_pages;

/* FIFO on board */
buffsett.acq_fifo_depth_ve = 20;
buffsett.acq_fifo_thresh_ve = 0;/* no threshold */
buffsett.acq_fifo_mode = ORL_ACQMEM_FIFO;

/* FIFO at host */
buffsett.hbuff_elmnt_size_pages = buffsett.vid_elmnnt_size_pages;
buffsett.hbuff_size_ve = 1;/* host buffer has a size of one VE */
buffsett.host_fifo_depth_hbuffs = 20;/* FIFO depth */

OrlFunc( H, ORLF_INIT_VIDEO_BUFF, &buffsett );
```

10.16 SW trig ACQMEM_CIRCULAR

supported boards

CL	AN
Y	N

OrlFuncType
`ORLF_ACQ_STOP_SWTRIG`

parameter
struct SOrlNullStruct

see also
Initialize video buffers 59

description
This function should be called to give the software trigger if the acquisition mode (SOrlVideoBuffSett.acq_fifo_mode) is ORL_ACQMEM_CIRCULAR_STOP_AT_SWTRIG.

10.17 Start/stop video acquisition

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_ACQUIRE_ENABLE`

parameter

struct SOrlAcquireEnable

item	type	direction
start	ArvDWORD	in

see also

Initialize video acquisition 55
Initialize video buffers 59

description

item	description
start	0: disable video acquisition 1: enable video acquisition After enable the acquisition module it waits for 'start_trig' is valid, see Initialize video acquisition

After Initialize video acquisition and Initialize video buffers the video acquisition may be enabled by this function.
After calling this function with parameter start == 1, the acquire process will wait on the start trigger: struct SOrlVideoAcqSett item start_trig, see 55.

10.18 Fill levels of video buffers

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_GET_VIDEO_BUFF_FILL`

parameter

struct SOrlVideoBuffFillLevel

item	type	direction
acq_fifo_fill_ve	ArvDWORD	out
pci_fifo_fill_ve	ArvDWORD	out
host_fifo_fill_hbuffs	ArvDWORD	out

see also

Initialize video buffers 59
Purge video buffers 66

description

item	description
acq_fifo_fill_ve	number of Video Elements (VEs) in the on board FIFO
pci_fifo_fill_ve	number of VEs in the on board FIFO which are waiting for transfer to host
host_fifo_fill_hbuffs	number of filled buffers of the host FIFO

This function gives information about the fill levels of the video FIFOs.
(acq_fifo_fill_ve + pci_fifo_fill_ve) equals to the number of VEs in the on board FIFO.
host_fifo_fill_hbuffs gives the fill level of the host video FIFO.

example

```
SOrlVideoBuffFillLevel  levs;

ORLINITSTRUCT( levs );
OrlFunc( H, ORLF_GET_VIDEO_BUFF_FILL, &levs );
```

10.19 Purge video buffers

supported boards

CL	AN
Y	N

OrlFuncType
`ORLF_PURGE_VIDEO_BUFF`

parameter
struct SOrlNullStruct

see also
Fill levels of video buffers 65

description
This function should be called to purge the on board video FIFO and the host video FIFO. After calling this function, the FIFOs are empty.

example

```
SOrlNullStruct    null;

ORLINITSTRUCT( null );
OrlFunc( H, ORLF_PURGE_VIDEO_BUFF, &null );
```

10.20 Move VE to host

supported boards

CL	AN
Y	N

OrlFuncType
`ORLF_VE_TO_HOST`

parameter
struct SOrlNullStruct

see also
Dequeque host buffer 68

description
This function should be called to dequeue one Video Element from the on board FIFO, transfer it to the host and enqueue it to the host FIFO.
This function is normally used when the on board VEs are ordered as a ring (see Initialize video buffers 59). When acquiring is ready, you should call ORLF_VE_TO_HOST to retrieve the on board VEs.

10.21 Dequeque host buffer

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_HFIFO_GET`

parameter

struct SOrlHostBuffInfo

item	type	direction
buffernr	ArvDWORD	out
buffer	ArvPVOID	out
numbytes	ArvDWORD	out
w	ArvDWORD	out
h	ArvDWORD	out
pitch	ArvDWORD	out
bitspix	OrlNBitsPixel	out
coltype	OrlCamColorType	out
nbytespix	ArvDWORD	out
cont_type	VidElType	out
cont_flags	VidElFlags	out
cont_nbytes	ArvDWORD	out
cont_framecnt	ArvDWORD	out
cont_nstrb	ArvDWORD	out
cont_nlines	ArvDWORD	out
num_in_hfifo	ArvDWORD	out
format	OrlPixelFormat	out

see also

Get CameraLink video information 54
Initialize video buffers 59
Fill levels of video buffers 65
Move VE to host 67
Mark host buffer as empty 74

description

item	description
information about the buffer	
buffernr	The index of the host buffer. The first host buffer has index '0'.
buffer	Pointer to the buffer
numbytes	Size of the buffer (bytes)
w	Number of pixels per line (x direction)
h	Number of lines per frame (y direction)
pitch	Number of bytes per line

item	description
bitspix	Number of bits per pixel: • ORL_NBITS_PIXEL_8 • ORL_NBITS_PIXEL_10 • ORL_NBITS_PIXEL_12 • ORL_NBITS_PIXEL_14 • ORL_NBITS_PIXEL_16 • ORL_NBITS_PIXEL_RGB24
coltype	Color type: • ORL_CAM_COLORTYPE_MONOCHROME • ORL_CAM_COLORTYPE_RGB
nbytespix	Number of bytes per pixel
information about the video element(s)	
cont_type	Bitwise parameter. Gives the type of VEs which are placed in this buffer: • ORL_ACQ_TYPE_EMPTY VE doesn't contain data • ORL_ACQ_TYPE_FIRST The VE of the begin of a video frame • ORL_ACQ_TYPE_MIDDLE A VE which is not at the begin or at the end of a video frame • ORL_ACQ_TYPE_LAST ORL_ACQ_TYPE_COMPLETED Indicates that the end (last line) of a video frame is placed in the buffer • ORL_ACQ_TYPE_CORRUPTED Indicates that the acquired data was corrupted
cont_flags	Bitwise parameter • ORL_VIDFLAG_ERR_HBUFF_OVL Host buffer overflow The acquired video frame contains more data then the size of this host buffer. The rest of the video frame data is purged.
cont_nbytes	number of bytes placed in this buffer so far
cont_framecnt	Video frame index. Each acquired video frame gets its own index. The index-counter is reset to zero when the DSP is booted by ORLF_PROGRAM_FIRMWARE.
cont_nstrb	number of CameraLink Strobe of a video line
cont_nlines	number of video lines copied ti this buffer, so far
information about the host FIFO	
num_in_hfifo	number of buffers placed in the host FIFO

item	description
format	Format of pixels • ORL_PIXFRMT_Y 8-bit/pixel • ORL_PIXFRMT_Y10 10-bit/pixel • ORL_PIXFRMT_Y12 12-bit/pixel • ORL_PIXFRMT_Y14 14-bit/pixel • ORL_PIXFRMT_Y16 16-bit/pixel • ORL_PIXFRMT_RGB888 24-bit/pixel RGB

This function is called to dequeue a host buffer from the host FIFO. Struct SOrlHostBuffInfo gives information about the dequeued host buffer and the content of the buffer (cont_xxx items). This function return ORL_RET_NOT_FOUND if the host FIFO is empty.
A host buffer is intended to hold a complete video frame. A video frame may build up of one or more Video Elements (VEs) (see Initialize video buffers 59). Therefore, a host buffer contains one or more VEs. Item cont_type gives information about the VEs of the host buffer, see Figure 28: 3 VEs placed in one host buffer.
If a host buffer is dequeued from the FIFO it can't be filled by the orlando, so the video data is frozen. After calling ORLF_HFIFO_PUT_EMPTY the buffer can be used by the orlando board again.

Figure 26: Acquired video frame

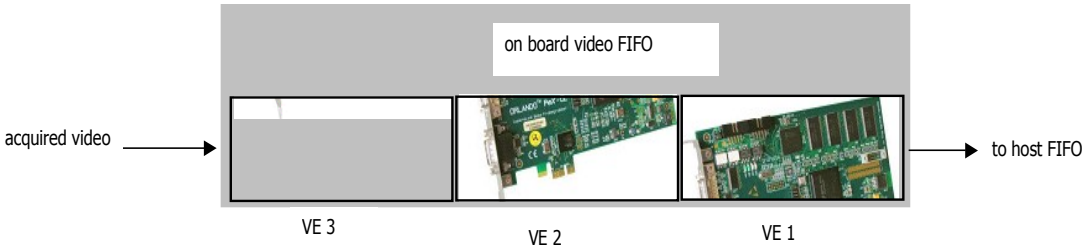
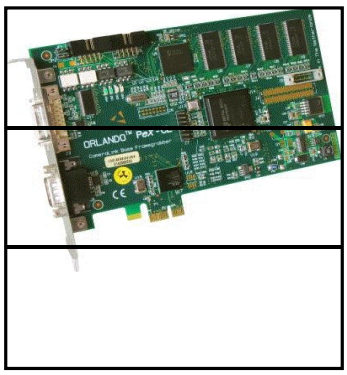


Figure 27: Video frame in on board video FIFO as 3 VEs



VE 1:
ORL_ACQ_TYPE_FIRST

VE 2:
ORL_ACQ_TYPE_MIDDLE

VE 3: ORL_ACQ_TYPE_LAST
ORL_ACQ_TYPE_COMPLETED

Figure 28: 3 VEs placed in one host buffer

example
see example Mark host buffer as empty 74.

10.22 Retrieve host buffer info

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_GET_HBUFFINFO`

parameter

struct SOrlHostBuffInfo

item	type	direction
buffernr	ArvDWORD	in
buffer	ArvPVOID	out
numbytes	ArvDWORD	out
w	ArvDWORD	out
h	ArvDWORD	out
pitch	ArvDWORD	out
bitspix	OrlNBitsPixel	out
coltype	OrlCamColorType	out
nbytespix	ArvDWORD	out
format	OrlPixelFormat	out

see also

Initialize video buffers 59

description

item	description
buffernr	The index of the host buffer. The first host buffer is indicated by index '0'.
buffer	Pointer to the allocated buffer
numbytes	Size of the buffer (bytes)
w	Number of pixels per line (x direction)
h	Number of lines per frame (y direction)
pitch	Number of bytes per line
bitspix	Number of bits per pixel: • ORL_NBITS_PIXEL_8 • ORL_NBITS_PIXEL_10 • ORL_NBITS_PIXEL_12 • ORL_NBITS_PIXEL_14 • ORL_NBITS_PIXEL_16 • ORL_NBITS_PIXEL_RGB24
coltype	Color type: • ORL_CAM_COLORTYPE_MONOCHROME • ORL_CAM_COLORTYPE_RGB
nbytespix	Number of bytes per pixel

item	description
format	Pixel format • ORL_PIXFRMT_Y • ORL_PIXFRMT_Y10 • ORL_PIXFRMT_Y12 • ORL_PIXFRMT_Y14 • ORL_PIXFRMT_Y16 • ORL_PIXFRMT_RGB888

This function should be called to get information of the allocated host buffers (Initialize video buffers 59).
This function does not dequeue the buffer from the host FIFO.

10.23 Mark host buffer as empty

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_HFIFO_PUT_EMPTY`

parameter

struct SOrlHostBuffEmpty

item	type	direction
buffernr	ArvDWORD	in

see also

Move VE to host 67
Dequeque host buffer 68

description

item	description
buffernr	index of a host buffer of the host FIFO

This function should be called to mark a dequeued host buffer as empty. This means that it can be used for new Video Elements from the on board buffer.

example

```
OrlRet      ret;
SOrlHostBuffInfo  hinfo;
SOrlHostBuffEmpty hempty;

ORLINITSTRUCT( hinfo );
ORLINITSTRUCT( hempty );

/* dequeue a buffer from host FIFO */
ret = OrlFunc( H, ORLF_HFIFO_GET, &hinfo );
if ( ret != ORL_RET_SUCCESS )
      return;

/*    here: processing of the host buffer.
      Use hinfo.buffer pointer to access the buffer.
 */

/* buffer processing is ready, so give it back */
hempty.buffernr = hinfo.buffernr;
OrlFunc( H, ORLF_HFIFO_PUT_EMPTY, &hempty );
```

10.24 Save host buffer as file

supported boards

CL	AN
Y	Y

OrlFuncType

`ORLF_SAVE_HOSTBUFF`

parameter

struct SOrlSaveFile

item	type	direction
filename	ArvPCHAR	in
filenamelength	ArvDWORD	in
hostbuffernr	ArvDWORD	in
bmtype	OrlBMType	in
quality	ArvDWORD	in

see also

description

item	description
filename	string which contains the file name
filenamelength	length of the file name
hostbuffernr	index of the host buffer to save
bmtype	image file type • ORL_BM_BMP Windows BMP file • ORL_BM_RAW Raw data, no conversion • ORL_BM_BMP_UD Windows BMP file, negative Height • ORL_BM_JPEG JPEG file, use item 'quality'.
quality	only when bmtype is ORL_BM_JPEG indicates the quality of the JPEG image. The range is 1..100: 1 is worst, 100 is best JPEG quality.

Orlando CL

bmtype	supports (OrlNBitsPixel):
ORL_BM_BMP ORL_BM_BMP_UD	ORL_NBITS_PIXEL_8 .. ORL_NBITS_PIXEL_16 ORL_NBITS_PIXEL_RGB24
ORL_BM_RAW	ORL_NBITS_PIXEL_8 .. ORL_NBITS_PIXEL_16 ORL_NBITS_PIXEL_RGB24
ORL_BM_JPEG	ORL_NBITS_PIXEL_8 ORL_NBITS_PIXEL_RGB24

bmtype	supports (OrlPixelFormat):
ORL_BM_BMP ORL_BM_BMP_UD	ORL_PIXFRMT_Y ORL_PIXFRMT_RGB565 ORL_PIXFRMT_RGB888
ORL_BM_RAW	all ORL_PIXFRMT_xxxx
ORL_BM_JPEG	ORL_PIXFRMT_YCBCR422 ORL_PIXFRMT_Y ORL_PIXFRMT_RGB888

example

```
OrlRet      ret;
SOrlHostBuffInfo  hinfo;
SOrlSaveFile      savefile;
ArvCHAR           filename[32] = "orl.bmp";

ORLINITSTRUCT( hinfo );
ORLINITSTRUCT(savefile );

/* dequeue a buffer from host FIFO */
ret = OrlFunc( H, ORLF_HFIFO_GET, &hinfo );
if ( ret != ORL_RET_SUCCESS )
      return;
savefile.filename = filename;
savefile.filenamelength = strlen( savefile.filename );
savefile.hostbuffernr = hinfo.buffernr;
savefile.bmtype = ORL_BM_BMP;
ret = OrlFunc( H, ORLF_SAVE_HOSTBUFF, &savefile );
```

10.25 Get GP input levels

supported boards

CL	AN
Y	Y

OrlFuncType

`ORLF_GET_INPUT`

parameter

struct SOrlInput

item	type	direction
inp	OrlInputPin	in
level	ArvDWORD	out

see also

Set input interrupt 81

description

item	description
inp	indicates a General Purpose input pin • ORL_INP_GPIN0 General Purpose input 0 • ORL_INP_GPIN1 General Purpose input 1
level	the state of the General Purpose input pin A value of '0' indicates low level, '1' indicates high level

This function returns the state of a General Purpose input pin.

example

```
SOrlInput   inp;

ORLINITSTRUCT( inp );

/* reads the level of General Purpose Input 0 */
inp.inp = ORL_INP_GPIN0;
OrlFunc( H, ORLF_GET_INPUT, &inp );
printf( "GP input 0 level: %s.\n", inp.level ? "HIGH" : "LOW" );
```

10.26 Set output signals

supported boards

CL	AN
Y	Y

OrlFuncType
`ORLF_SET_OUTPUT`

parameter
struct SOrlOutput

item	type	direction
outp	OrlOutputPin	in
src	OrlLevelSrc	in

see also
Get GP input levels 77
Set timer 83

description

Orlando CL model item	description
outp	A output pin: • ORL_OUTP_GPOUT0 General Purpose output 0 • ORL_OUTP_GPOUT1 General Purpose output 1 • ORL_OUTP_CC1* CameraLink Camera Control 1 • ORL_OUTP_CC2* CameraLink Camera Control 2 • ORL_OUTP_CC3* CameraLink Camera Control 3 • ORL_OUTP_CC4* CameraLink Camera Control 4
src	The source of the output pin: • ORL_LEVSRC_LOGIC_LOW • ORL_LEVSRC_LOGIC_HIGH Sets the output to Low or High level. • ORL_LEVSRC_LVAL • ORL_LEVSRC_LVAL_INV The output is driven by the (inverted) CameraLink LVAL signal. • ORL_LEVSRC_FVAL • ORL_LEVSRC_FVAL_INV The output is driven by the (inverted) CameraLink LVAL signal. • ORL_LEVSRC_TIMER0 • ORL_LEVSRC_TIMER1 The output is driven by the output of Timer 0/1. • ORL_LEVSRC_TIMER0_INV

Orlando CL model	
item	description
	• ORL_LEVSRC_TIMER1_INV The output is driven by the inverted output of Timer 0/1. • ORL_LEVSRC_GPIN0 • ORL_LEVSRC_GPIN1 The output is driver by General Purpose input 0/1. • ORL_LEVSRC_GPIN0_INV • ORL_LEVSRC_GPIN1_INV The output is driver by inverted General Purpose input 0/1 • ORL_LEVSRC_ROI Sets the output when the video acquire module process the Region Of Interest • ORL_LEVSRC_ROI_INV Sets the output when the video acquire modules process the outside the Region Of Interest range

Orlando AN model	
item	description
outp	A output pin: • ORL_OUTP_GPOUT0 General Purpose output 0 • ORL_OUTP_GPOUT1 General Purpose output 1
src	The source of the output pin: • ORL_LEVSRC_LOGIC_LOW • ORL_LEVSRC_LOGIC_HIGH Sets the output to Low or High level. • ORL_LEVSRC_GPIN0 • ORL_LEVSRC_GPIN1 The output is driver by General Purpose input 0/1. • ORL_LEVSRC_GPIN0_INV • ORL_LEVSRC_GPIN1_INV The output is driver by inverted General Purpose input 0/1 • ORL_LEVSRC_ADC0_HSYNC ORL_LEVSRC_ADC0_HSYNC_INV (inverted) Horizontal sync of input module 0. • ORL_LEVSRC_ADC0_VSYNC ORL_LEVSRC_ADC0_VSYNC_INV (inverted) Vertical sync of input module 0. • ORL_LEVSRC_ADC1_HSYNC ORL_LEVSRC_ADC1_HSYNC_INV (inverted) Horizontal sync of input module 1. • ORL_LEVSRC_ADC1_VSYNC ORL_LEVSRC_ADC1_VSYNC_INV (inverted) Vertical sync of input module 1.

This function configures the output signals of the orlando board. There are two types of outputs:

- Camera Link CCx signals
- General Purpose outputs

Each output can set to a fixed level (ORL_LEVSRC_LOGIC_LOW or ORL_LEVSRC_LOGIC_HIGH) or can be driven by an (inverted) input signal:

- Camera Link LVAL or FVAL
- output of the onboard timer, see page 83
- General Purpose inputs
- Acquiring module

example

```
SOrlOutput  outp;

ORLINITSTRUCT( outp );

/* connect GPIN1 input to CC1 output: */
outp.outp = ORL_OUTP_CC1;
outp.src = ORL_LEVSRC_GPIN1;
OrlFunc( H, ORLF_SET_OUTPUT, &outp );
```

10.27 Set input interrupt

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_SET_INPUT_INTERR`

parameter

struct SOrlInpInterr

item	type	direction
inp	OrlInputPin	in
edge	OrlEdge	in

see also

Get GP input levels 77
callback function OrlCbFunc 38

description

item	description
inp	Indication of General Purpose Input pin: • ORL_INP_GPIN0 • ORL_INP_GPIN0
edge	The edge which will generate an interrupt: • ORL_EDGE_NONE Disables interrupt generation • ORL_EDGE_FALLING Interrupt at falling edge of GP input • ORL_EDGE_RISING Interrupt at rising edge of GP input • ORL_EDGE_BOTH Interrupt at falling and rising edges.

This function should be called to receive interrupts on level changes of a General Purpose input pins.

example

```
SOrlInpInterr     interr;
ORLINITSTRUCT( interr );

/* Falling edge of a signal at GP input 0 will generate an interrupt */
interr.inp = ORL_INP_GPIN0;
interr.edge = ORL_EDGE_FALLING;

OrlFunc( H, ORLF_SET_INPUT_INTERR, &interr );

/* ------ */

/* if an interrupt occurs, the callback function will be called.
this function is installed by ORLF_INIT, page 37 */
ArvDWORD CBFunc( PSOrlCbParm parm )
{

      /* here: other callback handling... */
      if ( parm->inttype == ORL_INT_GP0 )
      {
            /* GP input 0 interrupt occured */
      }
      else if ( parm->inttype == ORL_INT_GP1 )
      {
            /* GP input 1 interrupt occured */
      }
} /* CBFunc */
```

10.28 Set timer

supported boards

CL	AN
Y	N

OrlFuncType
`ORLF_SET_TIMER`

parameter
struct SOrlTimer

item	type	direction
timerid	OrlTimerId	in
start_trig	OrlTriggerSrc	in
stop_trig	OrlTriggerSrc	in
opt	ArvDWORD	in
low_cycles	ArvDWORD	in
high_cycles	ArvDWORD	in

see also
Set timer software trigger 87
Set output signals78
Initialize video acquisition 55

description

item	description
timerid	Id of timer • ORL_TIMERID_0 Timer 0 • ORL_TIMERID_1 Timer 1
start_trig	Timer start trigger • ORL_TRIGSRC_SW Trigger by calling function ORLF_TIMER_SWTRIG, see Set timer software trigger 87 • ORL_TRIGSRC_LVAL_RISING • ORL_TRIGSRC_LVAL_FALLING Trigger by rising/falling edge of CameraLink LVAL signal • ORL_TRIGSRC_FVAL_RISING • ORL_TRIGSRC_FVAL_FALLING Trigger by rising/falling edge of CameraLink FVAL signal • ORL_TRIGSRC_TIMER0_RISING • ORL_TRIGSRC_TIMER0_FALLING Trigger by rising/falling edge of Timer 0 output. • ORL_TRIGSRC_TIMER1_RISING • ORL_TRIGSRC_TIMER1_FALLING Trigger by rising/falling edge of Timer 1 output. • ORL_TRIGSRC_GPIN0_RISING

item	description
	• ORL_TRIGSRC_GPIN0_FALLING Trigger by rising/falling edge of General Purpose input 0 • ORL_TRIGSRC_GPIN1_RISING • ORL_TRIGSRC_GPIN1_FALLING Trigger by rising/falling edge of General Purpose input 1 • ORL_TRIGSRC_START_OF_ROI Trigger if the acquire module gets the begin of the ROI • ORL_TRIGSRC_END_OF_ROI Trigger if the end of the ROI is reached
stop_trig	Timer stop trigger Same triggers as start_trig. Only used if timer generates pulses continuously.
opt	Timer options, bitwise • ORL_TIMEROPT_DISABLE Disable timer. • ORL_TIMEROPT_ENABLE Enable timer. • ORL_TIMEROPT_MODE_ONESHOT Timer generates one pulse. • ORL_TIMEROPT_MODE_CONT Timer generates pulses continuous. • ORL_TIMEROPT_SEQ_LOWHIGH Timer starts with low level. • ORL_TIMEROPT_SEQ_HIGHLOW Timer starts with high level
low_cycles	Number of cycles of the low level period The cycle counter runs at 66MHz, so one cycle is about 15 ns. The range is: min: 0 = 1 cycle = 15.15 ns max 2^{32} = 2^{32} + 1 cycles = 65.075 s defines: ORL_TIMER_CYCLE_FREQ: frequency of timer (Hz) ORL_TIMER_CYCLE_TIME: one period of timer (s) ORL_TMER_MAX_CYCLES: max value of low_cycles ORL_TIMER_MIN_TIME: min period (s) ORL_TIMER_MAX_TIME: max period (s)
high_cycles	Number of cycles of the high level period See low_cycles

The orlando has two on board timers, which can be used for triggering various outputs. Timer 0 and timer 1 are identical.
A timer will run if the start_trigger occurs.

If the mode is continuous (opt = ORL_TIMEROPT_MODE_CONT), the timer generates a frequency specified by low_cycles and high_cycles. It can be stopped by the stop_trigger. After the stop_trigger was occurred, the timer will continue with the current pulse and the timer stops. Disabling a timer (ORL_TIMEROPT_DISABLE) will stop the timer directly.

If the mode is one shot (opt = ORL_TIMEROPT_MODE_ONESHOT), the timer generates one pulse.

The frequency or pulse length depends on the low and high period. A period (in seconds) should be converted to cycles. The formula is:

cycles = period(s) * ORL_TIMER_CYCLE_FREQ.

Eg: period should be 12 ms:
0.012 * ORL_TIMER_CYCLE_FREQ (= 66E6 Hz) -1 = 791999 cycles

This value should be used in items low_cycles and high_cycles.

Next figures shows timer output with various values of item opt.

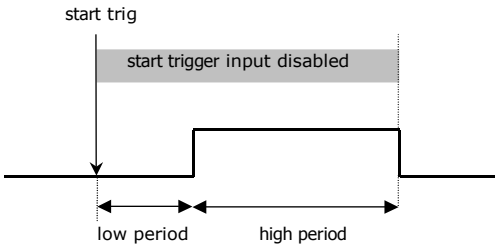


Figure 29 Timer output: starts with low, one pulse (MODE_ONESHOT, SEQ_LOWHIGH)

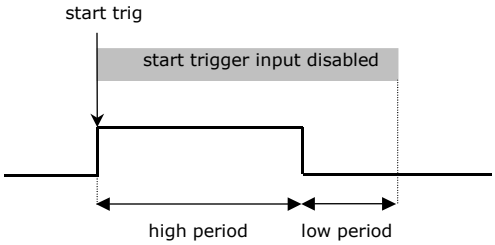


Figure 30 Timer output: starts with high, one pulse (MODE_ONESHOT, SEQ_HIGHLOW)

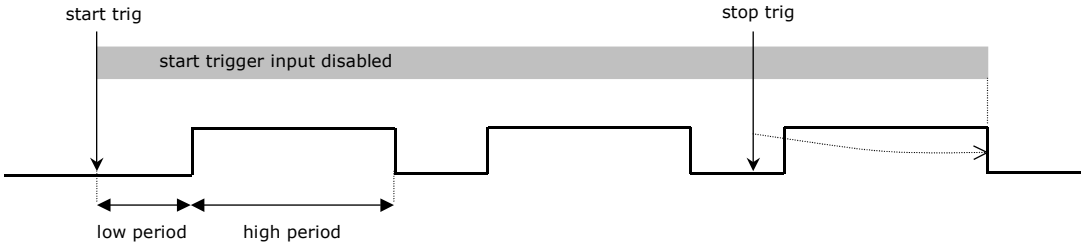


Figure 31 Timer output: starts with low, runs continuously until stop trigger (MODE_CONT, SEQ_LOWHIGH)

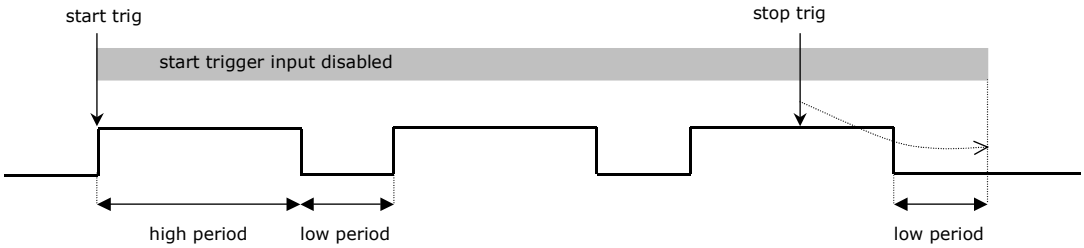


Figure 32 Timer output: starts with high, runs continuously until stop trigger (MOD_CONT, SEQ_ HIGHLOW)

example

```
/* Creates a timer which generates a frequency of 7 KHz with a duty
cycle of 50% */

SOrlTimerSWTrig   trig;
SOrlTimer   timer;
ArvDWORD    cycles;
ORLINITSTRUCT( trig );
ORLINITSTRUCT( timer );

timer.timerid = ORL_TIMERID_0;
timer.start_trig = ORL_TRIGSRC_SW;
timer.stop_trig = ORL_TRIGSRC_SW;
timer.opt = ORL_TIMEROPT_ENABLE | ORL_TIMEROPT_MODE_CONT |
                                          ORL_TIMEROPT_SEQ_LOWHIGH;
cycles = ORL_TIMER_CYCLE_FREQ/7000;
cycles = (cycles / 2) - 1;
timer.low_cycles = cycles;
timer.high_cycles = cycles;
OrlFunc( H, ORLF_SET_TIMER, &timer );

/* start timer by Software trigger */
trig.timerid = timer.timerid;
trig.trig = ORL_SWTRIG_START;
OrlFunc( H, ORLF_TIMER_SWTRIG, &trig );
```

10.29 Set timer software trigger

supported boards

CL	AN
Y	N

OrlFuncType

`ORLF_TIMER_SWTRIG`

parameter

struct SOrlTimerSWTrig

item	type	direction
timerid	OrlTimerId	in
trig	OrlSWTrig	in

see also

Set timer 83

description

item	description
timerid	Id of timer: • ORL_TIMERID_0 Timer 0 • ORL_TIMERID_1 Timer 1
trig	Software trigger type • ORL_SWTRIG_START Start trigger. • ORL_SWTRIG_STOP Stop trigger

This function is the software trigger for the timer. The timer start_trig or stop_trig should set to ORL_TRIGSRC_SW.

example

See Set timer 83

10.30 Create analog input module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_MODULE_IN`

parameter

struct SOrlModInSett

item	type	direction
module	OrlModule	in
port	OrlPort	in
vtime	OrlVideoTime	in
gain_mode	OrlVideoGainMode	in
gain_y_cvbs	ArvDWORD	in
gain_c	ArvDWORD	in
bcs_mode	OrlVideoBCSMode	in
brightness	ArvDWORD	in
contrast	ArvSDWORD	in
saturation	ArvSDWORD	in
hue	ArvSDWORD	in
flags	ArvDWORD	in
roi	struct SOrlRect	
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in
○ height	ArvDWORD	in
scale_width	ArvDWORD	in
scale_height	ArvDWORD	in

see also

Enable/disable a module 118
Destroy an module 119
Initialize/release module interrupts 114
Retrieve module information 120

description

item	description
module	input module of stream • ORL_MODULE_VIDEO_IN0 • ORL_MODULE_VIDEO_IN1

item	description
port	video input port of the specified input module • ORL_PORT_CVBS0: Composite input 0 • ORL_PORT_CVBS1: Composite input 1 • ORL_PORT_CVBS2: Composite input 2 • ORL_PORT_CVBS3: Composite input 3 • ORL_PORT_CVBS4: Composite input 4 • ORL_PORT_YC0: S-Video input 0 • ORL_PORT_YC1: S-Video input 1 • ORL_PORT_RGB0: RGB input
vtime	video timing of the input source (camera) • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video • ORL_VTIME_SECAM: SECAM (50 Hz) video
gain_mode	on board video gain mode • ORL_VGAIN_AUTO automatic video gain • ORL_VGAIN_FIXED fixed gain (use gain_y_cvbs and gain_c) • ORL_VGAIN_FIXED_DEFAULTS fixed gain, 0dB
gain_y_cvbs	gain level of composite video or Y part of S-Video, depends on port. range: 0..511: -3dB..+6dB only if gain_mode is ORL_VGAIN_FIXED
gain_c	gain levels of C part of S-Video range: 0..511: -3dB..+6dB only if port is ORL_PORT_YC*x* and gain_mode is ORL_VGAIN_FIXED
bcs_mode	Bright/Contr/Satur/Hue mode • ORL_VBCS_NORMAL Bright/Contr/Satur/Hue can be changed by next parameters • ORL_VBCS_DEFAULTS Bright/Contr/Satur/Hue are set to default values (ITU levels)
brightness	Luminance brightness control range: 0..255: dark..bright default: 128
contrast	Luminance contrast control range: -128..127: -2(inverse)..+1.984 default: 68 (1.063)
saturation	Chrominance saturation control range: -128..127: -2(inverse)..+1.984 default: 64 (1.0)
hue	Chrominance hue control range: -128..127: -180°..+178.6° default: 0 (not avail if port ORL_PORT_RGB0 is used)

item	description
flags	bitwise flags • ORL_FLAG_MIRROR video lines will be mirrored • ORL_FLAG_POWER_DOWN ADC of input will be set to power down mode • ORL_FLAG_INV_FID odd-even field detection will be inverted
roi	
○ xoffs	number of pixels which will be skipped range: 2..720, even value
○ yoffs	number of lines which will be skipped
○ width	number of pixels/line to acquire even value max: ORL_PAL_FRAME_WIDTH/ ORL_NTSC_FRAME_WIDTH
○ height	number of lines/ field to acquire max: ORL_PAL_FIELD_HEIGHT/ ORL_NTSC_FIELD_HEIGHT
scale_width	number of pixels/line after video scaler, even value downscaling possible, total upscale factor limited to about 1.17x of whole field max: ORL_PAL_FRAME_WIDTH/ ORL_NTSC_FRAME_WIDTH
scale_height	number of lines/frame after video scaler, even value. downscaling possible, total upscale factor limited to about 1.17x of whole fiel max: ORL_PAL_FIELD_HEIGHT/ ORL_NTSC_FIELD_HEIGHT

This function creates an analog video input module. This module acquires the incoming analog video and digitizes it.
The resulting digital video data can be used as source for other modules.

The Region Of Interest (ROI) is the region of an input image that should be acquired. This can be the complete image or a part of it.
After the ROI the image can be scaled. See next figure.

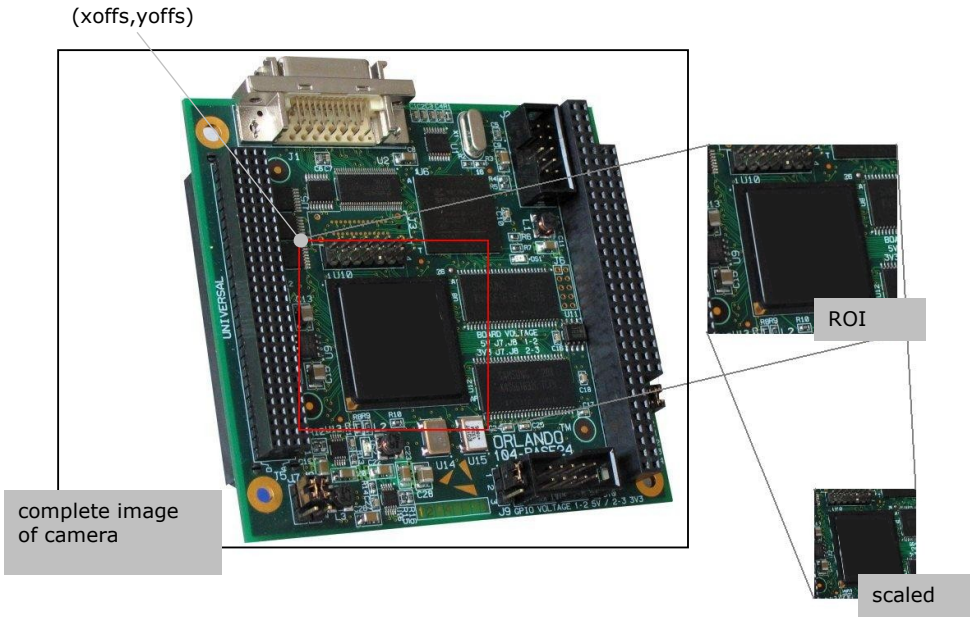


Figure 33 Complete image, Region Of Interest, video scaler

The ROI and the scaled dimensions are video field sizes.
After creation the state of the module is ORL_MODSTATE_INIT, see Retrieve module information on page 120.

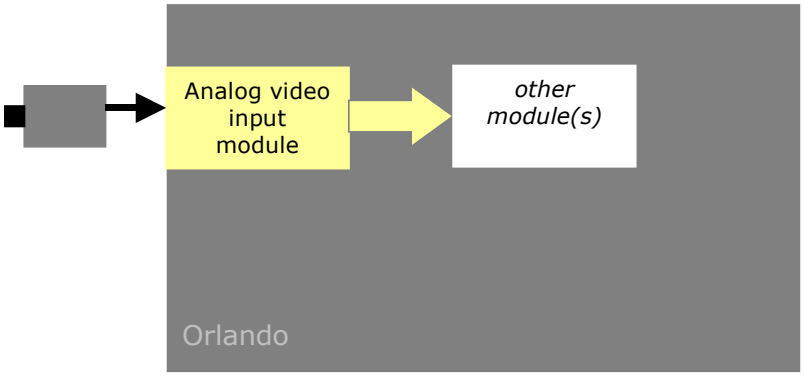


Figure 34 place of input module on the Orlando board

example

```
/* creates analog input module 0 */

SOrlModInSett     sett;
ORLINITSTRUCT(sett);

sett.module_in = ORL_MODULE_VIDEO_IN0;
sett.port = ORL_PORT_CVBS0;/* composite video */
sett.vtime = ORL_VTIME_PAL;/* 50Hz PAL */
sett.gain_mode = ORL_VGAIN_FIXED_DEFAULTS;
sett.bcs_mode = ORL_VBCS_DEFAULTS;
sett.flags = 0x0;

/* acquire full field */
```

```
sett.roi.xoffs = 2;
sett.roi.yoffs = 17;
sett.roi.width = ORL_PAL_WIDTH;
sett.roi.height = ORL_PAL_FIELD_HEIGHT;

/* scale to 1/2 */
sett.scale_width_out = sett.roi.width/2;
sett.scale_height_out = sett.roi.height/2;

OrlFunc( H, ORLF_INIT_MODULE_IN, &sett );
```

10.31 Create analog output module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_MODULE_OUT`

parameter

struct SOrlModOutSett

item	type	direction
module_out	OrlModule	in
module_src	OrlModule	in
pos_x_src	ArvDWORD	in
pos_y_src	ArvDWORD	in
bg_color_out	ArvDWORD	in
port	OrlPort	in
vtime	OrlVideoTime	in
flags	ArvDWORD	in
outp_width	ArvDWORD	in
outp_height	ArvDWORD	in

see also

Enable/disable a module 118
Destroy an module 119
Initialize/release module interrupts 114
Retrieve module information 120

description

item	description
module_out	analog output module which will be created • ORL_MODULE_VIDEO_OUT0 • ORL_MODULE_VIDEO_OUT1
module_src	video source of this module, eg ORL_MODULE_VIDEO_IN1
pos_x_src	x-offset of the video source (#pixels) in the result
pos_y_src	y-offset of the video source (#lines) in the result
bg_color_out	background color this color is used in the non-video region at the output see ORL_PIXFRMT_RGB565 171.
port	video port of the specified output module • ORL_PORT_CVBS0: Composite output 0 • ORL_PORT_CVBS1: Composite output 1 • ORL_PORT_YC0: S-Video output 0 • ORL_PORT_CVBS_DUAL: Composite output 0 and 1 • ORL_PORT_CVBS_YC: Composite output 1 and S-Video output 0 • ORL_PORT_RGB0_SYNC: RGB (with sync) output

item	description
vtime	Generated analog video output timing • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video
flags	Video output options, bitwise parameter (OR-ed) • ORL_FLAG_POWER_DOWN: sets the Digital Analog Coverter to power down mode • ORL_FLAG_NO_COLOR Switches the color information at the video output off
outp_width	number of pixels/line which will be displayed: ORL_PAL_WIDTH/ORL_NTSC_WIDTH
outp_height	number of lines/field which will be displayed: ORL_PAL_FRAME_HEIGHT or ORL_NTSC_FRAME_HEIGHT

This function creates a video output module. The Orlando has two video output modules: ORL_MODULE_VIDEO_OUT0 and ORL_MODULE_VIDEO_OUT1. The output module converts the on board digital video to analog video.

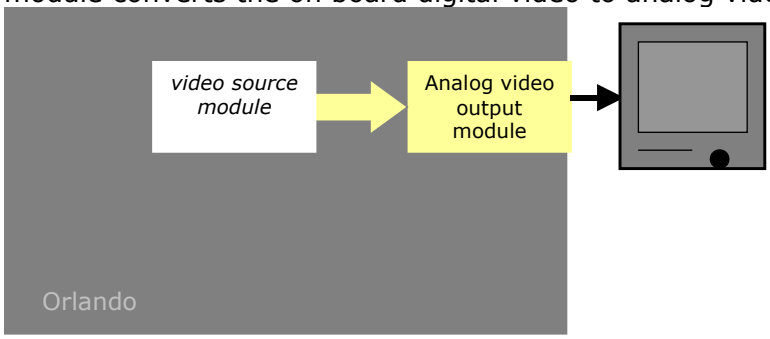


Figure 35 place of the output module on the Orlando board

Parameter module_src is the video source of this module. The source module should be initialized before creating the output module.
pos_x_src and pos_y_src is the offset of the source in the resulting image. The source region should fit in the output region (outp_width and outp_height) of this output module.

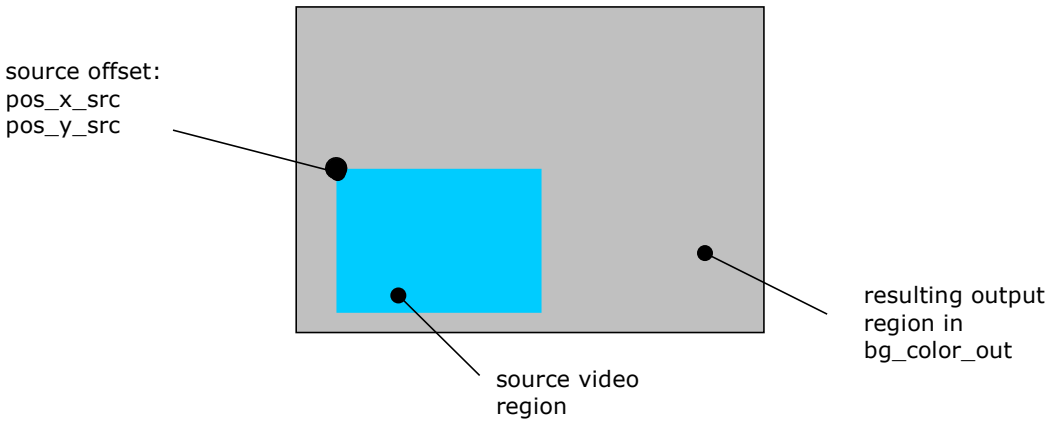


Figure 36 source video in output

After creating the module state will be: ORL_MODSTATE_INIT.

example

```
/* create a output module with host-to-board module as video source */
SOrlModOutSett    sett;
ORLINITS(sett);
/* name of output module */
sett.module_out = ORL_MODULE_VIDEO_OUT0;
/* video source is host */
sett.module_src = ORL_MODULE_HOST2BRD0;
/* no source offset */
sett.pos_x_src = 0;
sett.pos_y_src = 0;
sett.bg_color_out = ORL_YCBCR_BLACK;
/* composite video out, NTSC
sett.port = ORL_PORT_CVBS0;
sett.vtime =  ORL_VTIME_NTSC;
sett.flags = 0UL;
OrlFunc( H, ORLF_INIT_MODULE_OUT, &sett );
```

10.32 Create video fusion module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_VMODULE_FUSION`

parameter

struct SOrlModFusionSett

item	type	direction
module_fusion	OrlModule	in
module_src0	OrlModule	in
pos_x_src0	ArvDWORD	in
pos_y_src0	ArvDWORD	in
module_src1	OrlModule	in
pos_x_src1	ArvDWORD	in
pos_y_src1	ArvDWORD	in
bg_color_out	ArvDWORD	in
module_sync	OrlModule	in
fusion_type	OrlFusionType	in
key_color	SOrlColorRange	
○ start0	ArvDWORD	in
○ end0	ArvDWORD	in
○ start1	ArvDWORD	in
○ end1	ArvDWORD	in
○ start2	ArvDWORD	in
outp_width	ArvDWORD	in
outp_height	ArvDWORD	in
flags	ArvDWORD	in

see also

Enable/disable a module 118
Destroy an module 119
Retrieve module information 120

description

item	description
module_fusion	name of the video fusion module which should be created. ORL_VMODULE_FUSIONx
module_src0	video source 0 module (background)
pos_x_src0	x-offset of the video source 0 (#pixels) in the result
pos_y_src0	y-offset of the video source 0 (#pixels) in the result
module_src1	video source 1 module (foreground)
pos_x_src1	same as pos_x_src0 for source 1
pos_y_src1	same as pos_y_src0 for source 1

item	description
bg_color_out	background color this color is used in the non-video region at the output see ORL_PIXFRMT_RGB565 171.
module_sync	indicates the module which starts the fusion process should be one of the source modules
fusion_type	• ORL_FUSION_IN1_OVER_IN0 Fuse source 0 and source 1. Overlapping regions will result in source 1. • ORL_FUSION_IN1_OVER_IN0_KEYING Same as ORL_FUSION_IN1_OVER_IN0. Source 1 uses chroma key (see parameter key_color)
key_color	only used if fusion_type is ORL_FUSION_IN1_OVER_IN0_KEYING. Otherwise set all items to 0. Holds the chroma key of input 1. As a pixel value within the limits of the chroma key, it will be transparant. Next items shows the limits.
○ start0	Y limit low. Min 16 (black)
○ end0	Y limit high. Should be >= start0 and <= 235
○ start1	Cb limit low. Min 16
○ end1	Cb limit high. Should be >= start1 and <= 240
○ start2	Cr limit low. Min 16
○ end2	Cr limit high. Should be >= start1 and <= 240
outp_width	width (#pixels) of resulting image
outp_height	height (#lines/field) of resulting image
flags	0 or • ORL_FLAG_NOT_WAIT_FOR_NSYNC By default the fusion module waits until both sources has generated an image before the fusion starts. If this flag is set the fusion module waits only for module_sync.

This function creates a fusion module. A fusion module merges two video streams: module_src0 and module_src1. The source modules should be initialized before the fusion module is created.

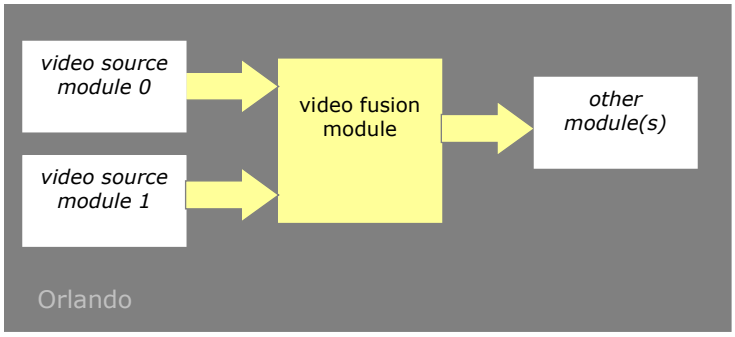


Figure 37 fusion module

With parameters pos_x_src and pos_y_src the position of the video sources in the fusion-output stream are set.
The source regions should fit in the output region (outp_width and outp_height) of the fusion module.

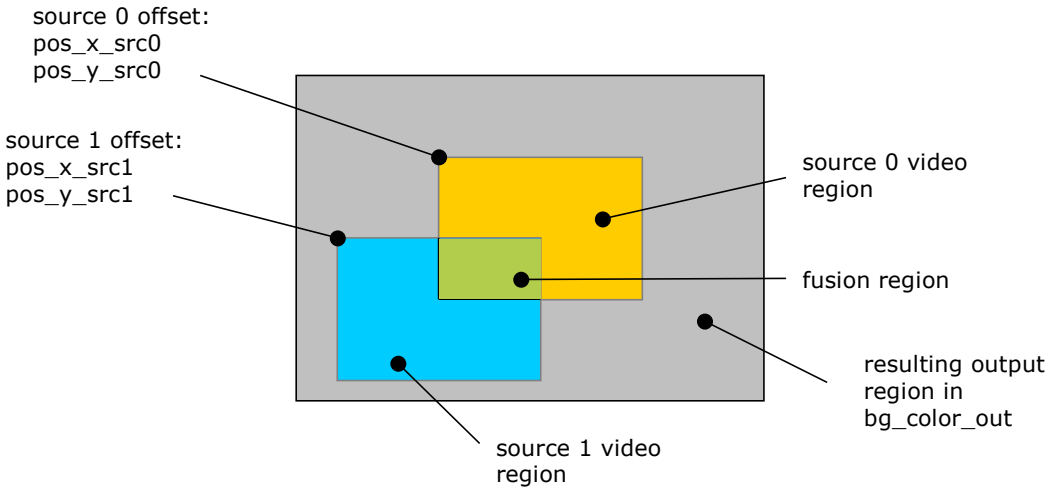


Figure 38 result of fusion module

The result of fusion region depends on items fusion_type and key color. If the fusion_type is ORL_FUSION_IN1_OVER_IN0 source 0 will be overlapped by source 1. Key_color isn't used.
If fusion_type is ORL_FUSION_IN1_OVER_IN0_KEYING the fusion region depends on video source 0, source 1 and key_color. Source 1 overlaps source 0. If a pixel of source 1 falls in the key_color range the pixel of source 1 will be replaced by source 0 (known as chroma keying or color keying). The key_color constists of three ranges: Y, Cb and Cr. The pixel will be transparent if it falls between the 3 ranges.

After creating the module state will be ORL_MODSTATE_INIT.

example

```
/* creates a fusion module.
      The video sources ORL_MODULE_VIDEO_IN1 and ORL_VMODULE_OVL_TEXT0
      should be created already */


SOrlColorRange    key_range;
SOrlModFusionSett sett;

ORLINITSTRUCT(key_color_range);
ORLINITSTRUCT W(sett);

/* create color key: only black pixels will be transparent */
key_color_range.start0 = 0x10; /* y */
key_color_range.end0 = 0x11; /* y */
key_color_range.start1 = 0x80; /* cb */
key_color_range.end1 = 0x81; /* cb */
key_color_range.start2 = 0x80; /* cr */
key_color_range.end2 = 0x81; /* cr */


/* name of fusion module */
sett.module_fusion = ORL_VMODULE_FUSION0;
```

```
/* video source 0: analog input 1 module */
sett.module_src0 = ORL_MODULE_VIDEO_IN1;
/* no offset: */
sett.pos_x_src0 = 0;
sett.pos_y_src0 = 0;

/* video source 1: text overlay module */
sett.module_src1 = ORL_VMODULE_OVL_TEXT0;
/* no offset: */
sett.pos_x_src1 = 0;
sett.pos_y_src1 = 0;

/* background color: black */
sett.bg_color_out = YCBCR_BLACK;
/* source 0 syncs the fusion output */
sett.module_sync = sett.module_src0;

/* fusion uses keying */
sett.fusion_type = ORL_FUSION_IN1_OVER_IN0_KEYING;
sett.key_color = key_color_range;

sett.outp_width = ORL_PAL_WIDTH;
sett.outp_height = ORL_PAL_FIELD_HEIGHT;
sett.flags = ORL_FLAG_NOT_WAIT_FOR_NSYNC;

OrlFunc( H, ORLF_INIT_VMODULE_FUSION, &sett );
```

10.33 Create overlay text module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_VMODULE_TEXT_OVL`

parameter

struct SOrlModTxtOverlaySett

item	type	direction
module_ovl	OrlModule	in
outp_width	ArvDWORD	in
outp_height	ArvDWORD	in

see also

Enable/disable a module 118
Destroy an module 119
Retrieve module information 120
Add static text to text overlay on page 101
Add counter to text overlay 103
Remove item from text overlay 106

description

item	description
module_ovl	name of the video fusion module which should be created. ORL_VMODULE_OVL_TEXTx
outp_width	width (pixels) of the resulting video (max 720)
outp_height	height (lines) of the resulting field video (max 242/288)

This module holds texts, which can be used as overlay for video. The output of this module should be a fusion module, which fuses the video with the overlay characters.
After creating an overlay text module, text items can be added and removed from the text overlay. The state will be ORL_MODSTATE_INIT.

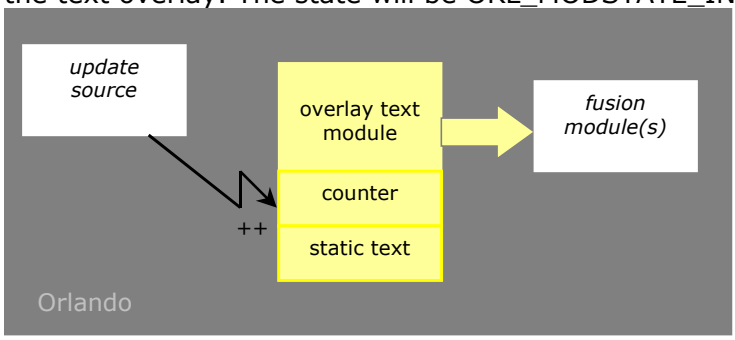


Figure 39 Overlay text module with two items

example

see Add static text to text overlay 101

10.33.1 Add static text to text overlay

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_ADDOVL_STAT_TEXT`

parameter

struct SOrlOvlStaticText

item	type	direction
module_ovl	OrlModule	in
handle	OrlOvlItemHandle	in/out
txt_format	SOrlOvlTxtFrmt	
○ in_overlay_video	ArvDWORD	in
○ posx	ArvDWORD	in
○ posty	ArvDWORD	in
○ align	OrlTxtAlign	in
○ font	OrlFontType	in
○ font_color	OrlOvlColor	in
○ bg_color	OrlOvlColor	in
○ txt_width	ArvDWORD	out
○ txt_height	ArvDWORD	out
text_str	ArvCHAR[ORL_MAX_OVLIT_TEXT_LEN+1]	in

see also

Create overlay text module 100
Add counter to text overlay 103
Remove item from text overlay 106

description

item	description
module_ovl	Overlay text module which should hold a new text item
handle	new item: set to ORL_INVALID_OVL_ITEM_HANDLE. After calling this function this item is set to a new item handle change item: set to the text item which should be changed
txt_format	
○ in_overlay_video	indicates if the text should be displayed in video: 0: false 1: true
○ posx	horizontal offset in overlay of the text, in #pixels of left side. Depends on 'align'
○ posty	vertical offset to the top of the text in overlay (0..241/287). In #lines/field of top of overlay.

item	description
○ align	horizontal alignment: • ORL_TXT_ALIGN_LEFT left aligned text • ORL_TXT_ALIGN_RIGHT right aligned text
○ font	used font: • ORL_STDFONT_8X8 • ORL_STDFONT_16X16 • ORL_STDFONT_24X24
○ font_color	color of the text
○ bg_color	color of the text background
○ txt_width	resulting text width in pixels
○ txt_height	resulting text height in lines/field
text_str	the text string

This function adds or changes a static text item to a text overlay module. After creating a static text, an item handle is returned. This handle should be used when the text should be changed or deleted. The resulting text width and height are returned too.
This function should be called after the *complete* stream is created and enabled.

example

```
/*
      Creates a overlay text module. A text item will be added
*/
SOrlModTxtOverlaySett   sett;
SOrlOvlStaticText       txt_item;

ORLINITSTRUCT(sett);
ORLINITSTRUCT(txt_item);

/* creates an overlay text module with NTSC dimensions */
sett.module_ovl = ORL_VMODULE_OVL_TEXT0;
sett.outp_width = ORL_NTSC_WIDTH;
sett.outp_height = ORL_NTSC_FIELD_HEIGHT;
OrlFunc( H, ORLF_INIT_VMODULE_TEXT_OVL, &sett );

/* create text item */
txt_item.module_ovl = sett.module_ovl;
txt_item.handle = ORL_INVALID_OVL_ITEM_HANDLE;
strcpy( txt_item.text_str, "overlay" );

/* create format of text item */
txt_item.txt_format.in_overlay_video = 1;
txt_item.txt_format.posx = 600;
txt_item.txt_format.posty = 100;
txt_item.txt_format.align = ORL_TXT_ALIGN_RIGHT;
txt_item.txt_format.font = ORL_STDFONT_8X8;
txt_item.txt_format.font_color = ORL_OVL_COLOR_WHITE;
txt_item.txt_format.bg_color = ORL_OVL_COLOR_BLACK;

/* create & enable modules of stream here */

/* add text item to overlay module */
```

```
OrlFunc( H, ORLF_ADDOVL_STAT_TEXT, &txt_item );

/* returns:
      txt_item.txt_format.txt_width: 112
      txt_item.txt_format.txt_height:8
*/
```

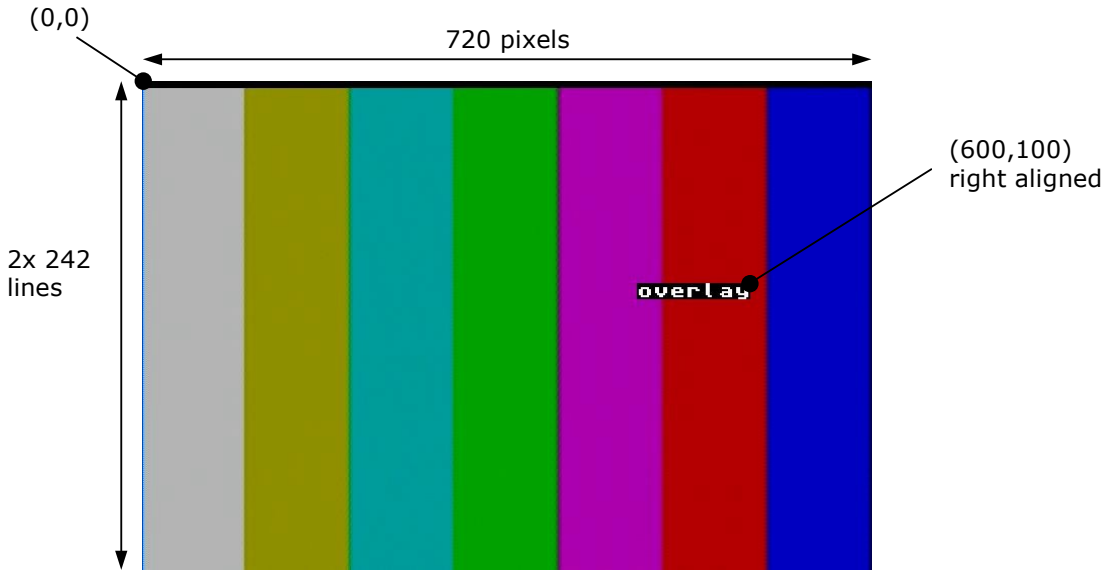


Figure 40 Result of add static text

10.33.2 Add counter to text overlay

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_ADDOVL_STAT_COUNTER`

parameter

struct SOrlOvlCounter

item	type	direction
module_ovl	OrlModule	in
handle	OrlOvlItemHandle	in/out
txt_format	SOrlOvlTxtFrmt	
○ in_overlay_video	ArvDWORD	in
○ posx	ArvDWORD	in
○ posty	ArvDWORD	in
○ align	OrlTxtAlign	in
○ font	OrlFontType	in
○ font_color	OrlOvlColor	in
○ bg_color	OrlOvlColor	in
○ txt_width	ArvDWORD	out
○ txt_height	ArvDWORD	out

item	type	direction
format	OrlFrmAnnotType	in
value	SOrlFrmAnnotValue	
○ val0	ArvDWORD	in
○ val1	ArvDWORD	in
○ val2	ArvDWORD	in
○ val3	ArvDWORD	in
update_src	OrlAnTrigSrc	in

see also
Create overlay text module 100
Add static text to text overlay 101
Remove item from text overlay 106

description

item	description
module_ovl	Overlay text module which should hold a new text item
handle	new item: set to ORL_INVALID_OVL_ITEM_HANDLE. After calling this function this item is set to a new item handle change item: set to the text item which should be changed
txt_format	
○ in_overlay_video	indicates if the text should be displayed in video: 0: false 1: true
○ posx	horizontal offset in overlay of the text. In #pixels of left side. Depends on 'align'
○ posty	vertical offset to the top of the text in overlay In #lines/field of top side. (0..241/287)
○ align	horizontal alignment: • ORL_TXT_ALIGN_LEFT left aligned text • ORL_TXT_ALIGN_RIGHT right aligned text
○ font	used font: • ORL_STDFONT_8X8 • ORL_STDFONT_16X16 • ORL_STDFONT_24X24
○ font_color	color of the text
○ bg_color	color of the text background
○ txt_width	resulting text width in pixels
○ txt_height	resulting text height in lines/field

item	description
format	annotation type • ORL_ANNOT_COUNTER counter value, start with '0', next: 1, 2,... • ORL_ANNOT_FRM_COUNTER frame counter, format: [hh]:[mm]:[ss].[ff] hh: hours, range 00.. >1000 mm: minutes, range: 00..59 ss: seconds, range: 00..59 ff: frames, range: 00..24 (PAL), 00..29 (NTSC)
value	
○ val0	if ORL_ANNOT_COUNTER: start value of the counter if ORL_ANNOT_FRM_COUNTER: start value of hours
○ val1	only if ORL_ANNOT_FRM_COUNTER start value of minutes
○ val2	only if ORL_ANNOT_FRM_COUNTER start value of seconds
○ val3	only if ORL_ANNOT_FRM_COUNTER start value of frames
update_src	item which increments the counter value • ORL_TRIG_VIDIN0_FRM • ORL_TRIG_VIDIN1_FRM video input x has received a video frame • ORL_TRIG_HOST2BRD0 • ORL_TRIG_HOST2BRD1 • ORL_TRIG_HOST2BRD2 • ORL_TRIG_HOST2BRD3 • ORL_TRIG_HOST2BRD4 • ORL_TRIG_HOST2BRD5 module host-to-board has received an image

This function adds or changes a counter to an overlay text module. The position, color, value etc can be set.
If a counter is successfully added, this function returns an item handle (handle) that can be used for changing or deleting it. The width and height of the counter is returned too.
The complete video stream should be initialized and enabled before calling this function.

example

```
/* creates a overlay text module. A frame-counter will be added on
position (40,210). The counter will be incremented if video input 1 has
received a video frame
*/
SOrlModTxtOverlaySett   sett;
SOrlOvlCounter          txt_item;

ORLINITSTRUCT(sett);
ORLINITSTRUCT(txt_item);

/* creates an overlay text module with NTSC dimensions */
sett.module_ovl = ORL_VMODULE_OVL_TEXT0;
```

```
sett.outp_width = ORL_NTSC_WIDTH;
sett.outp_height = ORL_NTSC_FIELD_HEIGHT;
OrlFunc( H, ORLF_INIT_VMODULE_TEXT_OVL, &sett );

/* create counter item */
txt_item.module_ovl = sett.module_ovl;
txt_item.handle = ORL_INVALID_OVL_ITEM_HANDLE;
txt_item.value.val0 = txt_item.value.val1 = txt_item.value.val2 =
txt_item.value.val3 = 0;
txt_item.update_src = ORL_TRIG_VIDIN1_FRM;
txt_item.format = ORL_ANNOT_FRM_COUNTER;

/* create format of text item */
txt_item.txt_format.in_overlay_video = 1;
txt_item.txt_format.posx = 40;
txt_item.txt_format.posty = 210;
txt_item.txt_format.align = ORL_TXT_ALIGN_LEFT;
txt_item.txt_format.font = ORL_STDFONT_8X8;
txt_item.txt_format.font_color = ORL_OVL_COLOR_TRANS;
txt_item.txt_format.bg_color = ORL_OVL_COLOR_BLACK;

/* create & enable modules of stream here */

/* add text item to overlay module */
OrlFunc( H, ORLF_ADDOVL_STAT_TEXT, &txt_item );

/* returns:
      txt_item.txt_format.txt_width: 176
      txt_item.txt_format.txt_height:8
*/
```

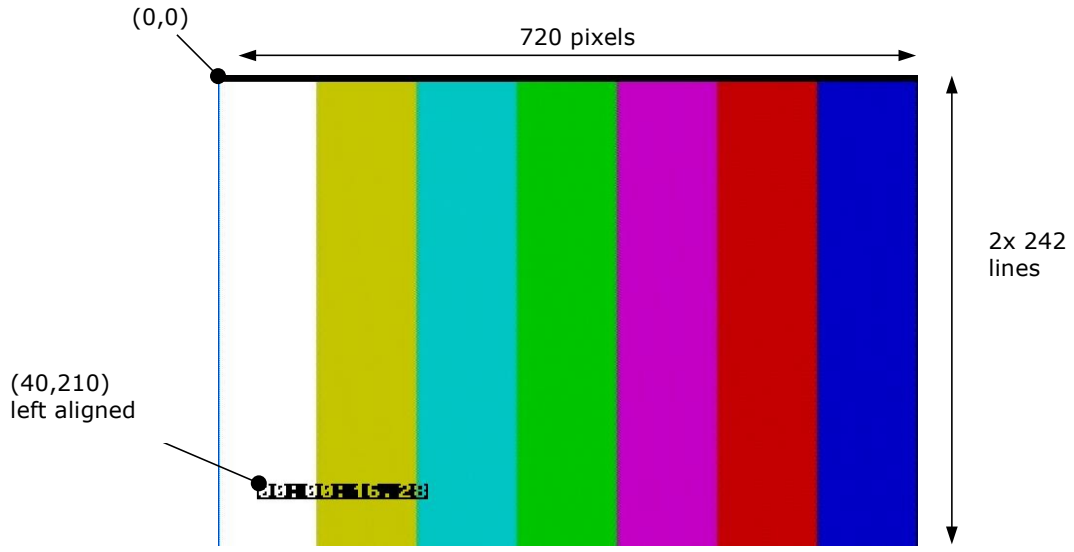


Figure 41 Result of adding a counter to overlay

10.33.3 Remove item from text overlay

supported boards

CL AN

N	Y

OrlFuncType
`ORLF_REMOVEOVL_ITEM`

parameter
struct SOrlOvlItem

item	type	direction
module_ovl	OrlModule	in
handle	OrlOvlItemHandle	in

see also
Create overlay text module 100
Add static text to text overlay 101
Add counter to text overlay 103

description

item	description
module_ovl	name of overlay text module (ORL_VMODULE_OVL_TEXTx)
handle	handle of an item in the overlay text module

Call this function if an item in the overlay text module should be removed.

10.34 Create board-to-host module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_MODULE_BRD2HOST`

parameter

struct SOrlModHostSett

item	type	direction
module_host	OrlModule	in
module_src	OrlModule	in
pos_x_src	ArvDWORD	in
pos_y_src	ArvDWORD	in
bg_color_out	ArvDWORD	in
fieldarrange_host	OrlFieldArrange	in
format_host	OrlPixelFormat	in
width	ArvDWORD	in
height	ArvDWORD	in
host_numplanes	ArvDWORD	out
host_pl0	SOrlPlaneSize	
○ buff_size	ArvDWORD	out
○ npages	ArvDWORD	out
host_pl1	SOrlPlaneSize	
same as host_pl0		
host_pl2	SOrlPlaneSize	
same as host_pl0		

see also

Create host-to-board module 111
Destroy an module 119
Initialize/release module interrupts 114
Retrieve module information 120

description

item	description
module_host	name of the board-to-host module which should be created (ORL_MODULE_BRD2HOSTx)
module_src	video source of this module
pos_x_src	x-offset of the video source (#pixels) in the result
pos_y_src	y-offset of the video source (#lines) in the result
bg_color_out	background color this color is used in the non-video region at the output see ORL_PIXFRMT_RGB565 171.

item	description
fieldarrange_host	how video fields are placed in buffers • ORL_FLD_INTERLACED odd and even lines are interlaced • ORL_FLD_DUAL odd and even lines not interlaced ORL_FLD_SINGLE only one video field per buffer
format_host	output of the color space convertor • ORL_PIXFRMT_YCBCR422 YCbCr 4:2:2 format, 16 bit/pixel, 1 plane • ORL_PIXFRMT_YCBCR422P 3 planes: plane0: Y plane1: Cb plane2: Cr • ORL_PIXFRMT_Y Y only, 8 bit/pixel, 1 plane • ORL_PIXFRMT_RGB565 16-bit RGB format, 1 plane • ORL_PIXFRMT_RGB888 24-bit RGB format (packed), 1 plane See also Pixel formats in memory 170
width	number of pixels/line, multiply of 8 max: ORL_PAL_FRAME_WIDTH ORL_NTSC_FRAME_WIDTH
height	number of lines/field even value max: ORL_PAL_FIELD_HEIGHT ORL_NTSC_FIELD_HEIGHT
host_numplanes	returns number of image planes
host_pl0	
○ buff_size	returns needed buffer size of plane 0
○ npages	returns needed number of host memory pages
host_pl1	
	same as host_pl0 of plane 1
host_pl2	
	same as host_pl0 of plane 2

This function creates a board-to-host module. This module receives video of another module, convert it to 'fieldarrange_host' and 'format_host'. Function ORLF_FRM_TRANSF (see: Transfer host frame buffer 130) transfers an image from this module to a host buffer. Such host buffer should be allocated by ORLF_ALLOC_HBUFF (123).
The video source module should be initialized before creating this host module. After successful creation the state will be ORL_MODSTATE_INIT.

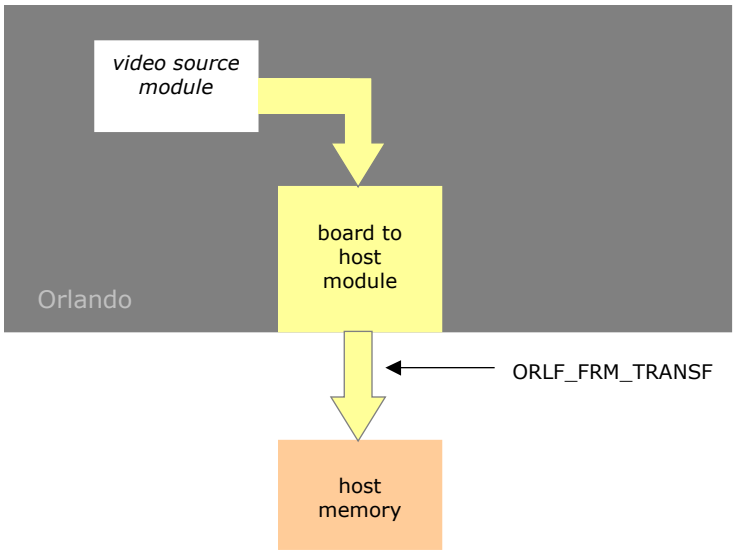


Figure 42 Board-to-host module

example

```
/*
      creates a board-to-host module. The video source is
      ORL_MODULE_VIDEO_IN. This host module is able to generate full
      frame (PAL), RGB 16-bit/pixel images.
*/

/* here: create ORL_MODULE_VIDEO_IN0... */

SOrlModHostSett   sett;
ORLINITSTRUCT( sett );
sett.module_host = ORL_MODULE_BRD2HOST0;
sett.module_src = ORL_MODULE_VIDEO_IN0;
sett.pos_x_src = 0;
sett.pos_y_src = 0;
sett.bg_color_out = YCBCR_BLACK;
sett.fieldarrange_host = ORL_FLD_INTERLACED;
sett.format_host = ORL_PIXFRMT_RGB565;
sett.width = ORL_PAL_WIDTH;
sett.height = ORL_PAL_FIELD_HEIGHT;

OrlFunc( H, ORLF_INIT_MODULE_BRD2HOST, &sett );
```

10.35 Create host-to-board module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_MODULE_HOST2BRD`

parameter

struct S

item	type	direction
module_host	OrlModule	in
fieldarrange_host	OrlFieldArrange	in
format_host	OrlPixelFormat	in
width	ArvDWORD	in
height	ArvDWORD	in
host_numplanes	ArvDWORD	out
host_pl0	SOrlPlaneSize	
○ buff_size	ArvDWORD	out
○ npages	ArvDWORD	out
host_pl1	SOrlPlaneSize	
same as host_pl0		
host_pl2	SOrlPlaneSize	
same as host_pl0		

see also

Create board-to-host module 108
Destroy an module 119
Initialize/release module interrupts 114
Retrieve module information 120

description

item	description
module_host	name of the host-to-board module which should be created (ORL_MODULE_HOST2BRDx)
fieldarrange_host	how video fields are placed in buffers • ORL_FLD_INTERLACED odd and even lines are interlaced • ORL_FLD_DUAL odd and even lines not interlaced • ORL_FLD_SINGLE only one video field per buffer

item	description
format_host	input of the color space convertor • ORL_PIXFRMT_YCBCR422 YCbCr 4:2:2 format, 16 bit/pixel, 1 plane • ORL_PIXFRMT_YCBCR422P 3 planes: plane0: Y plane1: Cb plane2: Cr • ORL_PIXFRMT_Y Y only, 8 bit/pixel, 1 plane • ORL_PIXFRMT_RGB565 16-bit RGB format, 1 plane • ORL_PIXFRMT_RGB888 24-bit RGB format (packed), 1 plane See also Pixel formats in memory 170
width	number of pixels/line, multiply of 8 max: ORL_PAL_FRAME_WIDTH ORL_NTSC_FRAME_WIDTH
height	number of lines/field even value max: ORL_PAL_FIELD_HEIGHT ORL_NTSC_FIELD_HEIGHT
host_numplanes	returns number of image planes
host_pl0	
○ buff_size	returns needed buffer size of plane 0
○ npages	returns needed number of host memory pages
host_pl1	
	same as host_pl0 of plane 1
host_pl2	
	same as host_pl0 of plane 2

This module receives images from the host which will be converted to the on board video format. The host-to-board modules can be used as video source for other modules (e.g. ORL_MODULE_VIDEO_OUT0).
Parameters fieldarrange_host, format_host, width and height describe the properties of host images.
Function ORLF_FRM_TRANSF (see: Transfer host frame buffer 130) transfers an image from a host buffer to this module. Such host buffer should be allocated by ORLF_ALLOC_HBUFF (123).
If the host-to-board module has received a new image an interrupt is generated.

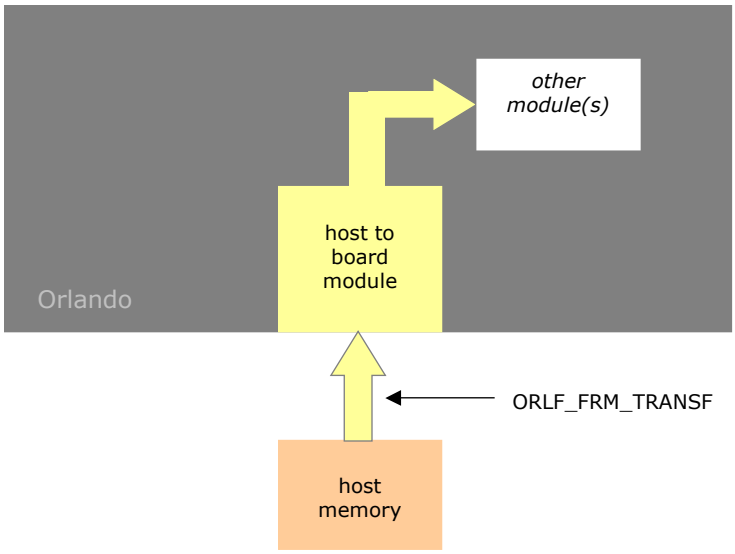


Figure 43Host-to-board module

example

```
/* creates a host-to-board module which is able to receive grey (8-bit)
100x100 images from the host.
*/

SOrlModHostSett sett;
ORLINITSTRUCT(sett);

sett.module_host = ORL_MODULE_HOST2BRD0;
sett.fieldarrange_host = ORL_FLD_INTERLACED;
sett.format_host = ORL_PIXFRMT_Y;
sett.width = 100;
sett.height = 100;

OrlFunc( H, ORLF_INIT_MODULE_HOST2BRD, &sett );
```

10.36 Initialize/release module interrupts

supported boards

CL	AN
N	Y

OrlFuncType
`ORLF_INIT_INTERR_MODULE`

parameter
struct SOrlModInitInterr

item	type	direction
module	OrlModule	in
interr_mask	OrlModInterrType	in

see also
Wait for modules interrupts 116

description

item	description
module	module which should generate interrupts, next modules are supported: ORL_MODULE_VIDEO_INx ORL_MODULE_VIDEO_OUTx ORL_MODULE_BRD2HOSTx ORL_MODULE_HOST2BRDx Other modules are not able to generate interrupts
interr_mask	Interrupt mask. If mask is '0' no new interrupts will be generated. Next (bitwise) masks are possible: ORL_MODINT_FRM_IN ORL_MODINT_FRM_RDY_FOR_OUT ORL_MODINT_FRM_OUT ORL_MODINT_FRM_SKIPPED

This function enables (and disables) interrupt generation of a module. Next matrix shows which interrupts a module generates.

	ORLAN_MODINT_xxx			
module type	FRM_IN	FRM_RDY_FOR_OUT	FRM_OUT	FRM_SKIPPED
VIDEO_IN	●			
VIDEO_OUT			●	
BRD2HOST		●	●	●
HOST2BRD	●			●

ORL_MODULE_VIDEO_INx generates an interrupt if a compelete frame is acquired. This gives 25/30 (PAL/NTSC) interrupts per second.

ORL_MODULE_VIDEO_OUTx generates an interrupt if a frame is send to the analog output. Gives 25/30 interrupts/sec.

ORL_MODULE_BRD2HOSTx generates three types of interrupts:
ORL_MODINT_FRM_RDY_FOR_OUT if an image is ready for transfer to host

ORL_MODINT_FRM_OUT: image is transferred to host
ORL_MODINT_FRM_SKIPPED: an (on board) image will be removed if it isn't transferred to host.

ORL_MODULE_HOST2BRDx generates two types of interrupts:
ORL_MODINT_FRM_IN: image is received from host
ORL_MODINT_FRM_SKIPPED: on board image is skipped because host send to many images (buffer overflow).

example

```
SOrlModInitInterr interr;
ORLINITSTRUCT(interr);

/* enable 2 interr.types of board-to-host module 0 */
interr.module = ORL_MODULE_BRD2HOST0;
interr.interr_mask = ORL_MODINT_FRM_RDY_FOR_OUT |
                     ORL_MODINT_FRM_SKIPPED;
OrlFunc( H, ORLF_INIT_INTERR_MODULE, &interr );

/* disable interrupts */
interr.interr_mask = 0UL;
OrlFunc( H, ORLF_INIT_INTERR_MODULE, &interr );
```

10.37 Wait for modules interrupts

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_WAIT_INTERR_MODULE`

parameter

struct SOrlModWaitInterr

item	type	direction
stopped	ArvDWORD	out
num_interr	ArvDWORD [NUM_ORL_MODULE] [ORL_NUM_MODULE_INTERR]	out

see also

Initialize/release module interrupts 114

description

item	description
stopped	returns: 0: interrupt process is running 1: interrupt process is stopped due eg. error
num_interr	returns number of interrupts occured after last call of ORLF_WAIT_INTERR_MODULE

After initializing module interrupts this function can be called. It waits until one or more modules have generated an interrupt.
Parameter num_interr gives the number of interrupts occurred after the last call.
num_interr is a two dimensional array:
num_interr[ORL_MODULE_xxx][ORLAN_MODINIT_xxxx].

example

```
/* Waits until an interrupt occur. Shows all modules and interrupt
counters. Should be called after modules and interrupts are initialized
*/
void InterrHandler()
{
      OrlRet      ret = ORL_RET_SUCCESS;
      int         imod, iint;
      SOrlModWaitInterr       waitinterr;
      do {
            ORLINITSTRUCT(waitinterr);
            /* blocks until interrupt occur */
            ret = OrlFunc( H, ORLF_WAIT_INTERR_MODULE, &waitinterr );

            /* walk trough all possible modules and interrupt types */
            for ( imod = 0; imod < NUM_ORL_MODULE; imod++ )
```

```
            {
                  printf( "module: %d\n", imod );
                  for ( iint = 0; iint < ORL_NUM_MODULE_INTERR; iint++ )
                  {
                        printf( "%d: %lu interrupts",
                              iint, waitinterr[imod][iint] );
                  }/* iint */
            }/* imod */
      } while( ret == ORL_RET_SUCCESS && !waitinterr.stopped );
}
```

10.38 Enable/disable a module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_MODULE_ENABLE`

parameter

struct SOrlModule

item	type	direction
module	OrlModule	in
enable	ArvDWORD	in

see also

Create analog input module 88
Create analog output module93
Create video fusion module 96
Create overlay text module 100
Create board-to-host module 108
Create host-to-board module 111
Destroy an module 119
Retrieve module information 120

description

item	description
module	Name of a created module which should be enabled or disabled.
enable	0: disable, stop with processing of video 1: enable, process video

This function enables or disables a module. After enabling a module is able to receive images and process these.
After enabling the state of the module will be ORL_MODSTATE_ENABLED, after disable: ORL_MODSTATE_INIT.

example

```
/* enable module ORL_MODULE_VIDEO_OUT1 */
SModule     smod;
ORLINITSTRUCT(smod);

smod.module = ORL_MODULE_VIDEO_OUT1;
smod.enable = 1;

OrlFunc( H, ORLF_MODULE_ENABLE, &smod );
```

10.39 Destroy an module

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_MODULE_RELEASE`

parameter

struct SOrlModule

item	type	direction
module	OrlModule	in

see also

Create analog input module 88
Create analog output module93
Create video fusion module 96
Create overlay text module 100
Create board-to-host module 108
Create host-to-board module 111
Enable/disable a module 118
Retrieve module information 120

description

item	description
module	Name of a created module which should be destroyed.

This function destroys a created module. After destroying the module can't be used and doesn't generate interrupts. The module state will be ORL_MODSTATE_NOT_INIT.

Note:
All modules that use images of the destroy-module will be destroyed too!

10.40 Retrieve module information

supported boards

CL	AN
N	Y

OrlFuncType

`ORL_GET_MODULE_INFO`

parameter

struct SOrlModInfo

item	type	direction
module	OrlModule	in
state	OrlModState	out

see also

Enable/disable a module 118
Destroy an module 119

description

item	description
module	name of module to retrieve the state
state	returns the state of the module: ORL_MODSTATE_UNKNOWN State can't be retrieved ORL_MODSTATE_NOT_INIT Module isn't created (eg released) ORL_MODSTATE_INIT Module is created but not enabled ORL_MODSTATE_GOTO_STOP Temporary state while going from state _ENABLED to _INIT ORL_MODSTATE_ENABLED Module is created and is enabled

This function retrieves the current state of a give module.

10.41 Get analog video information

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_GET_VIDEOIN_STAT`

parameter

struct SOrlVideoInputStat

item	type	direction
module	OrlModule	in
decoder_stat	ArvDWORD	out

see also

Initialize board interrupts 132

description

item	description
module	input module to retrieve analog video status: • ORL_MODULE_VIDEO_IN0 analog video input 0 • ORL_MODULE_VIDEO_IN1 analog video input 1
decoder_stat	returned video status (bitwise): • ORL_ADC_STAT_DCSTD[1:0} Detected color standard: 00: no color 01: NTSC 10: PAL 11: SECAM • ORL_ADC_STAT_WIPA white peak loop is activated • ORL_ADC_STAT_GLIMB gain value for luminance is limited (bottom) • ORL_ADC_STAT_GLIMT gain value for luminance is limited (top) • ORL_ADC_STAT_SLTCA slow time constant active • ORL_ADC_STAT_HLCK horizontal frequency not locked • ORL_ADC_STAT_RDCAP ready for capture • ORL_ADC_STAT_COPRO copy protected source detected • ORL_ADC_STAT_COLSTR macrovision color stripe burst • ORL_ADC_STAT_TYPE3 macrovision color stripe burst type 3

item	description
	• ORL_ADC_STAT_FIDT detected field frequency 60Hz (otherwise 50Hz) • ORL_ADC_STAT_HLVLN horizontal and vertical loops not locked • ORL_ADC_STAT_INTL interlaced video detected (otherwise non-interlaced)

This function can be called to retrieve which type of video is connected to the analog input of the orlando board. The input module should be initialized first.

example

```
SOrlVideoInputStat      stat;
ArvDWORD                dcstd = 0;
ORLINITSTRUCT( stat );

stat.module = ORL_MODULE_VIDEO_IN0;
OrlFunc( H, ORLF_GET_VIDEOIN_STAT, &stat );
if ( (stat.decoder_stat & ORL_ADC_STAT_HLCK ) )/* check HLCK bit */
{
      printf( "No video detected.\n" );
}
else
{
      printf( "Video detected:\n" );
      if ( stat.decoder_stat &  ORL_ADC_STAT_FIDT )
      {     /* 60 Hz */
            printf( "60Hz field rate, " );
      }
      else
      {
            printf( "50Hz field rate, " );
      }

      /* display colour standard */
      dcstd = stat.decoder_stat &
            (ORL_ADC_STAT_DCSTD1|ORL_ADC_STAT_DCSTD0);

      switch( dcstd )
      {
      case 1: printf( "NTSC colour\n" );break;
      case 2: printf( "PAL colour\n" );break;
      case 3: printf( "SECAM colour\n" );break;
      default: /* no colour */
            printf( "monochrome\n" );break;
      }
}
```

10.42 Allocate host buffer

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_ALLOC_HBUFF`

parameter

struct SOrlAllocPlHostBuff

item	type	direction
numplanes	ArvDWORD	in
plane0	struct SOrlPlaneBuff	
○ sz	struct SOrlPlaneSize	
○ • buff_size	ArvDWORD	in
○ pbuff	ArvPVOID	out
plane1	struct SOrlPlaneBuff	
(same items as plane0)		
plane2	struct SOrlPlaneBuff	
(same items as plane0)		
buffernr	ArvDWORD	out

see also

Free host buffer 128
Retrieve host plane buffer info 125
Transfer host frame buffer130

description

item	description
numplanes	Number of planes needed by image; 1 or 3. If numplanes is 1 only plane0 is valid.
plane0	
○ sz	
○ • buff_size	requested size of plane 0 (bytes)
○ pbuff	returned pointer to host buffer
plane1	
(same items as plane0)	
plane2	
(same items as plane0)	
buffernr	returned buffer index

This function allocates one host buffer that can be used to store frames of video streams or overlay data. The number of planes and the size of the needed host buffer can be copied from the returned host_pl*x* items of the stream initialize structs (e.g. SOrlVideoInputToHostStream.host_pl0.buff_size).
Returned item 'buffernr' should be used in other host buffer functions (e.g. deallocate host buffer). If more host buffers are needed, this function should be called again. Up to 32 (HOST_MAX_CAPT_BUFFS) can be allocated.

example

```
SOrlAllocPlHostBuff     hbuff, free_hbuff;
OrlRet                  ret;
ORLINITSTRUCT( hbuff );

/* create a buffer with 1 plane of 720kbyte
    these values can be copied of init-stream functions
*/
hbuff.numplanes = 1;
hbuff.plane0.sz.buff_size = 720000;
ret = OrlFunc( H, ORLF_ALLOC_HBUFF, &hbuff );
if ( ret == ORL_RET_SUCCESS )
{
      printf( "Buffer alloc'd: buffernr: %lu, ptr: %p\n",
            hbuff.buffernr, hbuff.plane0.pbuff );
}


/* free host buffer */
ORLINITSTRUCT( free_hbuff );
free_hbuff.buffernr = hbuff.buffernr;
OrlFunc( H, ORLF_FREE_HBUFF, &free_hbuff );
```

10.43 Retrieve host plane buffer info

supported boards

CL	AN
N	Y

OrlFuncType
`ORLF_GET_PL_HBUFFINFO`

parameter
struct SOrlPlHostBuffInfo

item	type	direction
buffernr	ArvDWORD	in
hstream	OrlStrHandle	out
numplanes	ArvDWORD	out
plane0	struct SOrlPlaneBuff	
○ sz	struct SOrlPlaneSize	
○ • buff_size	ArvDWORD	out
○ • npages	ArvDWORD	out
○ • horbytes	ArvDWORD	out
○ • bytes_used	ArvDWORD	out
○ pbuff	ArvPVOID	out
plane1	struct SOrlPlaneBuff	out
(same items as plane0)		
plane2	struct SOrlPlaneBuff	out
(same items as plane0)		
width_pix	ArvDWORD	out
height	ArvDWORD	out
fieldarrange	OrlFieldArrange	out
format	OrlPixelFormat	out
annotstr	ArvCHAR[ORL_ANNOT_MAX_NUM_CHARS]	out

see also
Allocate host buffer 123
Pixel formats in memory 170

description

item	description
buffernr	requested host buffer index
hstream	video stream handle which has accessed this buffer
numplanes	number of image planes: 1: plane0 3: plane0, plane1 and plane2
plane0	plane0 properties
○ • buff_size	size of buffer in bytes

item	description
○ • npages	number of physical host pages
○ • horbytes	number of bytes per video line (is set after buffer is read or written)
○ • bytes_used	number of bytes used by video stream (is set after buffer is read or written)
○ pbuff	pointer to image buffer
plane1	plane1 properties
(same items as plane0)	
plane2	plane2 properties
(same items as plane0)	
width_pix	number of pixels per video line
height	number of lines per video frame
fieldarrange	odd/even field arrangement in buffer • ORL_FLD_INTERLACED odd and even lines are interlaced • ORL_FLD_DUAL odd and even lines not interlaced • ORL_FLD_SINGLE only one video field
format	pixel format • ORL_PIXFRMT_YCBCR422 YCbCr 4:2:2 format, 16 bit/pixel • ORL_PIXFRMT_YCBCR422P plane0: Y plane1: Cb plane2: Cr • ORL_PIXFRMT_Y Y only, 8 bit/pixel • ORL_PIXFRMT_RGB565 16-bit RGB format • ORL_PIXFRMT_RGB888 24-bit RGB format (packed)
annotstr (16 bytes)	contains the annotation value of the video frame as string

This function retrieves information of an allocated host buffer. Some parameters are set after a video stream has accessed the buffer.

example

```
/*
      Retrieves info of a host buffer. If format is Y8 all pixels will
      be scanned.
*/
SOrlPlHostBuffInfo      hbuff;
ArvDWORD                x,y;
ArvBYTE                 *ptr;
ORLINITSTRUCT( hbuff );

hbuff.buffernr = 0; /* requested buffer index */
OrlFunc( H, ORLF_GET_PL_HBUFFINFO, &hbuff );

if ( hbuff.format == ORL_PIXFRMT_Y )
{
      /* 1 byte per pixel, 1 plane */
      for ( y = 0; y < hbuff.height; y++ )
      {
            /* set ptr to begin of line */
            ptr = hbuff.plane0.pbuff;
            ptr = ptr + (y * hbuff.plane0.sz.hor_bytes);

            for ( x = 0; x < hbuff.width_pix; x++ )
            {
                  /* *ptr contains data of one pixel */
                  ptr++;

            }
      }
}
```

10.44 Free host buffer

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_FREE_HBUFF`

parameter

struct SOrlAllocPlHostBuff

item	type	direction
buffernr	ArvDWORD	in

see also

Allocate host buffer 123

description

item	description
buffernr	buffer number which should be free'd

This function frees the resources used by host buffer 'buffernr'. After freeing the buffer it shouldn't be used by video frame data.

example

see Allocate host buffer 123

10.45 Load host buffer from file

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_LOAD_HOSTBUFF`

parameter

struct SOrlLoadFile

item	type	direction
filename	ArvPCHAR	in
filenamelength	ArvDWORD	in
hostbuffernr	ArvDWORD	in
bmtype	OrlBMType	in

see also

Save host buffer as file 75

description

item	description
filename	string which contains the file name
filenamelength	length of the file name
hostbuffernr	index of the host buffer to load (destination)
bmtype	image file type • ORL_BM_BMP Windows BMP file • ORL_BM_JPEG JPEG file

Loads a BMP or JPEG file and stores the data in a host buffer.

example

```
SOrlLoadFile      ldfile;
ORLINITSTRUCT( ldfile );

ldfile.filename = "orlando.jpg";
ldfile.filenamelength = strlen( ldfile.filename );
ldfile.hostbuffernr = hbuffix;/* host buffer index of alloc'd host
buffer */
ldfile.bmtype = ORL_BM_JPEG;/* JPEG file */
OrlFunc( H, ORLF_LOAD_HOSTBUFF, &ldfile );
```

10.46 Transfer host frame buffer

supported boards

CL	AN
N	Y

OrlFuncType
`ORLF_FRM_TRANSF`

parameter
struct SOrlFrmTransf

item	type	direction
hstream	OrlStrHandle	in
buffernr	ArvDWORD	in

see also
Allocate host buffer 123
Create board-to-host module 108
Create host-to-board module 111
Get transfer rate 131

description

item	description
hstream	indicates the video stream which has an input or output to host
buffernr	buffer number of an allocated host buffer

This function should be used to transfer video frames from the orlando board to the host buffers or from a host buffer to the orlando board. It depends on the video stream if the host buffer is the source or destination of the transfer:

- input to host stream (137): on board output buffer is source, host buffer is destination
- host to output stream (143): host buffer is source, on board input buffer is destination

The size of the destination buffer should be equal or larger then the source buffer.

10.47 Get transfer rate

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_GET_PCI_STAT`

parameter

struct SOrlPCIStat

item	type	direction
wr_mb_s	ArvDWORD	out
rd_mb_s	ArvDWORD	out
master_abort_count	ArvDWORD	out
target_abort_count	ArvDWORD	out

see also

Transfer host frame buffer 130

description

item	description
wr_mb_s	measured board to host rate of last transferred frame, in MByte/s
rd_mb_s	measured host to board rate of last transferred frame, in MByte/s
master_abort_count	Number of PCI Master aborts
target_abort_count	Number of PCI Target aborts

This functions returns information about the transfer rate of the orlando board and the host. The transfer rates are calculated on board after a video frame is transferred from/to the orlando board.

10.48 Initialize board interrupts

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_INTERR_BOARD`

parameter

struct SOrlInitInterrBoard

item	type	direction
interr_mask	OrlInterrType	in
gp0_edge	OrlEdge	in
gp1_edge	OrlEdge	in

see also

Initialize/release module interrupts 114
Wait on board interrupt 134

description

item	description
interr_mask	Interrupt mask. If mask is '0', interrupts are disabled. Next items can be OR-ed: • ORL_INT_GP0 interrupt if a 'gp0_edge' is detected at GP input 0 • ORL_INT_GP1 interrupt if a 'gp1_edge' is detected at GP input 1 • ORL_INT_ADC0 interrupt if a video source is (dis)connected at analog input module 0 • ORL_INT_ADC1 interrupt if a video source is (dis)connected at analog input module 1
gp0_edge	specifies which edge generates an interrupt, if interr_mask has ORL_INT_GP0 bit • ORL_EDGE_NONE disabled (no edge) • ORL_EDGE_FALLING falling edge • ORL_EDGE_RISING rising edge • ORL_EDGE_BOTH rising and falling edge
gp1_edge	specifies which edge generates an interrupt, if interr_mask has ORL_INT_GP1 bit • ORL_EDGE_NONE disabled (no edge) • ORL_EDGE_FALLING

item	description
	falling edge • ORL_EDGE_RISING rising edge • ORL_EDGE_BOTH rising and falling edge

example

```
void Init()
{
      SOrlInitInterrBoard    interr;
      ORLINITSTRUCT( interr );

      /* interr of GP0 and ADC 0 */
      interr.interr_mask = ORL_INT_GP0 | ORL_INT_ADC0;
      interr.gp0_edge = ORL_EDGE_FALLING;
      OrlFunc( H, ORLF_INIT_INTERR_BOARD, &interr );

      /* here: create an thread which handles the stream interrupts.
            CreateThread( ..., BoardInterrHandling, ... );
      */
}

/* thread function which handles interrupts of an orlando board */
void BoardInterrHandling()
{
      SOrlWaitInterrBoard    wait;
      ORLINITSTRUCT( wait );

      do {
            /* wait until an interrupt occur */
            OrlFunc( H, ORLF_WAIT_INTERR_BOARD, &wait );

            /* check if interrupt handling is stopped */
            if ( !wait.stopped )
            {
                  /* check which interrupts were occured
                     between two ORLF_WAIT_INTERR_BOARD calls
                  */

                  printf( "#GP0 edges: %lu\n", wait.num_gp0_in );
                  if ( wait.num_adc0 )
                  {
                        /* retrieve ADC status */

                        SOrlVideoInputStat     stat;
                        ORLINITSTRUCT( stat );
                        stat.module = ORL_MODULE_VIDEO_IN0;
                        OrlFunc( H, ORLF_GET_VIDEOIN_STAT, &stat );
                        /* todo: check stat */
                  }
            }
      } while ( !wait.stopped );
}
```

10.49 Wait on board interrupt

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_WAIT_INTERR_BOARD`

parameter

struct SOrlWaitInterrBoard

item	type	direction
stopped	ArvDWORD	out
num_gp0_in	ArvDWORD	out
num_gp1_in	ArvDWORD	out
num_adc0	ArvDWORD	out
num_adc1	ArvDWORD	out

see also

Initialize board interrupts 132

description

item	description
stopped	indicates if interrupt handling is stopped: 0: interrupt handling is running 1: interrupt handling is stopped or not initialized, don't call ORLF_WAIT_INTERR_BOARD again
num_gp0_in	number of interrupts at General Purpose input 0, between two calls
num_gp1_in	number of interrupts at General Purpose input 1, between two calls
num_adc0	number of interrupts of video ADC0, between two calls
num_adc1	number of interrupts of video ADC1, between two calls

Before calling this function, the board interrupts should be initialized.
This function blocks until a board interrupt occurs. The num_xxx parameters indicate how many times an interrupt was occurred between two function calls.

example

see page 132

11 Obsolete

The Orlando AN stream functions are obsolete. These functions are replaced by 'modules'.

11.1 Programming Guide AN model video-stream

This chapter describes the functions of the Orlando Analog (AN).

11.1.1 Overview

Next figure shows the functional block diagram of the Orlando AN.

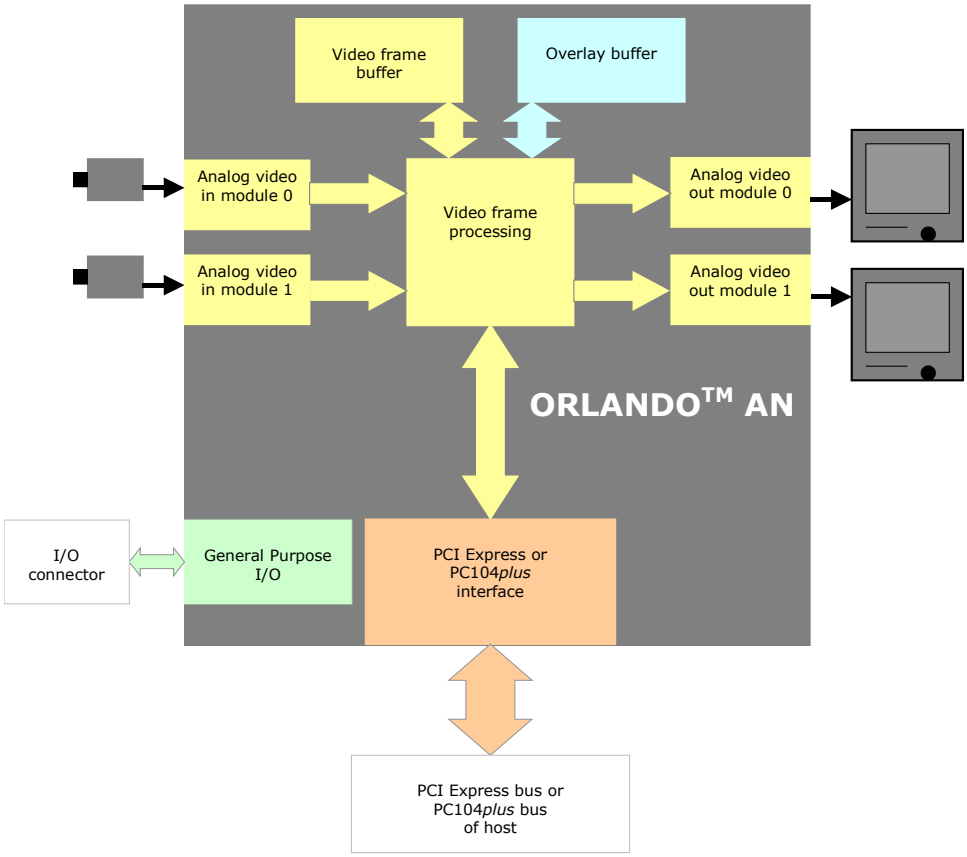


Figure 44 Functional block diagram

11.1.2 Analog video input

For video acquisition the Orlando AN has two the same independent video input modules. Both modules consist of (see Figure 45):

- Video multiplexer. Each input module has more video inputs. The video multiplexer selects one video input port.
- Analog Digital Converter (ADC). Converts analog video signals of a camera to digital video data.

- Gain. Amplifies the video signal. The amplifier can be set to a fixed value or to automatic gain.
- Color control. Used for brightness, contrast, saturation and hue settings.
- Region Of Interest (ROI). To grab a part of a full video frame.
- Video scaler. For downscaling or upscaling of the video frames.

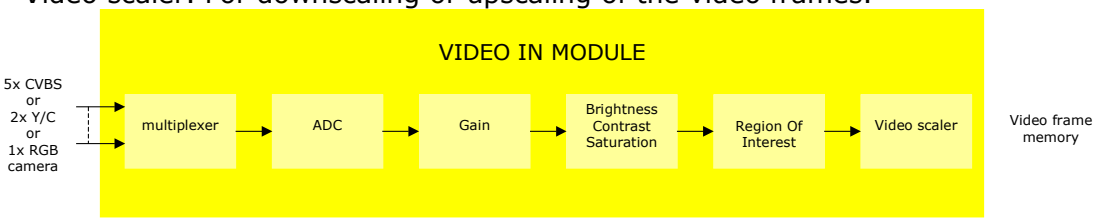


Figure 45 Analog video input module

The Region Of Interest (ROI) is the region of an input frame that should be acquired. This can be the complete input frame or a part of it.
The analog input modules of the orlando have also a video scaler. This can be used to downscale (and upscale) the video frames. See next figure.

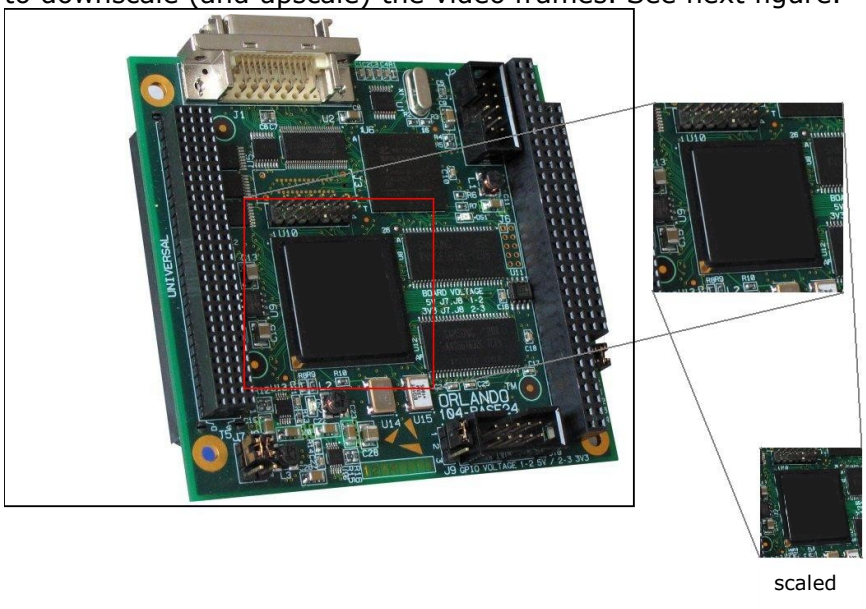


Figure 46 Complete frame, Region Of Interest, video scaler

11.1.3 Video frame processing

The video frame processing module retrieves the video frames from the Video In Module or from the host. Video processing consists of one or more tasks (see Figure 47):

- Annotate the video frame: Gives each frame sequential number.
- Color space converting, e.g. from YCbCr to RGB.
- Add full frame overlay to the video.
  The frame annotation value can be displayed in overlay data.
  Overlay data can be set by host.

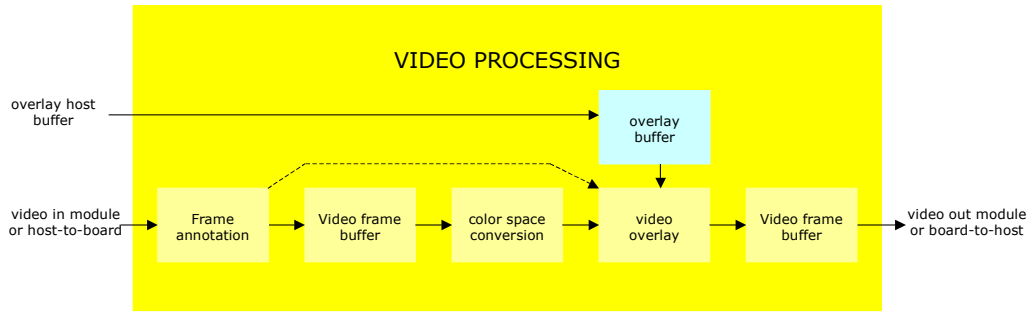


Figure 47 Video frame processing

11.1.4 Analog video output

The Orlando has two the same analog video output modules. These independent modules convert the digital video data to analog video signal. This video signal can be used to display the video frames on a monitor (see Figure 48).

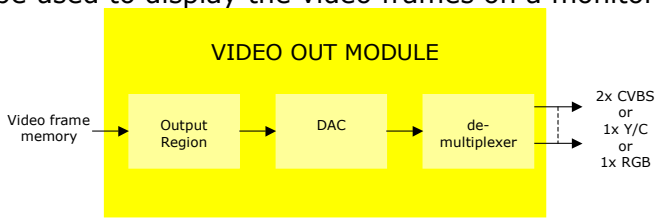


Figure 48 Analog video output module

11.1.5 Host-Orlando board interface

The host-Orlando interface (see Figure 49) is used for:

- board control, eg. initialize a video stream
- transfer video frames from the video processing module of the Orlando board to the host
- transfer video frames from the host to the video processing module of the Orlando board
- transfer overlay data to the Orlando board

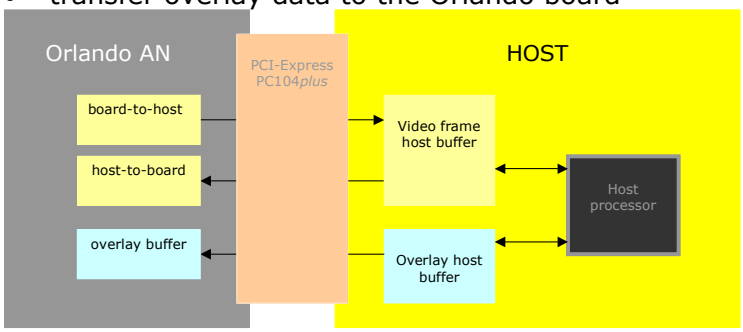


Figure 49 Host-Orlando board interface

11.2 Functions

11.2.1 Initialize video input to host stream

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_VIDEO_TO_HOST_STREAM`

parameter

struct SOrlVideoInputToHostStream

item	type	direction
input	struct SOrlVideoInputSett	
○ module	OrlModule	in
○ port	OrlPort	in
○ vtime	OrlVideoTime	in
○ gain_mode	OrlVideoGainMode	in
○ gain_y_cvbs	ArvDWORD	in
○ gain_c	ArvDWORD	in
○ bcs_mode	OrlVideoBCSMode	in
○ brightness	ArvDWORD	in
○ contrast	ArvSDWORD	in
○ saturation	ArvSDWORD	in
○ hue	ArvSDWORD	in
○ flags	ArvDWORD	in
roi	struct SOrlRect	
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in
○ height	ArvDWORD	in
scale_width	ArvDWORD	in
scale_height	ArvDWORD	in
fieldarrange	OrlFieldArrange	in
format	OrlPixelFormat	in
inp_buff_size	ArvDWORD	out
outp_buff_size	ArvDWORD	out
host_numplanes	ArvDWORD	out
host_pl0	struct SOrlPlaneSize	
○ buff_size	ArvDWORD	out
○ npages	ArvDWORD	out
○ hor_bytes	ArvDWORD	out
host_pl1	struct SOrlPlaneSize	
(same as host_pl0)		
host_pl2	struct SOrlPlaneSize	
(same as host_pl0)		
hstream	OrlStrHandle	out

see also
Initialize host to video output stream 143
Initialize video input to output stream 147
Video acquisition 24
Pixel formats in memory 170

description

item	description
input	
○ module	input module of stream • ORL_MODULE_VIDEO_IN0 • ORL_MODULE_VIDEO_IN1
○ port	video input port of the specified input module • ORL_PORT_CVBS0: Composite input 0 • ORL_PORT_CVBS1: Composite input 1 • ORL_PORT_CVBS2: Composite input 2 • ORL_PORT_CVBS3: Composite input 3 • ORL_PORT_CVBS4: Composite input 4 • ORL_PORT_YC0: S-Video input 0 • ORL_PORT_YC1: S-Video input 1 • ORL_PORT_RGB0: RGB input
○ vtime	video timing of the input source (camera) • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video • ORL_VTIME_SECAM: SECAM (50 Hz) video
○ gain_mode	on board video gain mode • ORL_VGAIN_AUTO automatic video gain • ORL_VGAIN_FIXED fixed gain (use gain_y_cvbs and gain_c) • ORL_VGAIN_FIXED_DEFAULTS fixed gain, 0dB
○ gain_y_cvbs	gain level of composite video or Y part of S-Video, depends on port. range: 0..511: -3dB..+6dB only if gain_mode is ORL_VGAIN_FIXED
○ gain_c	gain levels of C part of S-Video range: 0..511: -3dB..+6dB only if port is ORL_PORT_YC*x* and gain_mode is ORL_VGAIN_FIXED
○ bcs_mode	Bright/Contr/Satur/Hue mode • ORL_VBCS_NORMAL Bright/Contr/Satur/Hue can be changed by next parameters • ORL_VBCS_DEFAULTS Bright/Contr/Satur/Hue are set to default values (ITU levels)
○ brightness	Luminance brightness control range: 0..255: dark..bright default: 128
○ contrast	Luminance contrast control range: -128..127: -2(inverse)..+1.984

item	description
	default: 68 (1.063)
○ saturation	Chrominance saturation control range: -128..127: -2(inverse)..+1.984 default: 64 (1.0)
○ hue	Chrominance hue control range: -128..127: -180°..+178.6° default: 0 (not avail if port ORL_PORT_RGB0 is used)
○ flags	bitwise flags • ORL_FLAG_MIRROR video lines will be mirrored • ORL_FLAG_POWER_DOWN ADC of input will be set to power down mode • ORL_FLAG_INV_FID odd-even field detection will be inverted
roi	
○ xoffs	number of pixels which will be skipped range: 2..720, even value
○ yoffs	number of lines which will be skipped
○ width	number of pixels/line to acquire even value max: ORL_PAL_FRAME_WIDTH/ ORL_NTSC_FRAME_WIDTH
○ height	number of lines/frame (field) to acquire even value max: ORL_PAL_FRAME_HEIGHT/ ORL_NTSC_FRAME_HEIGHT
scale_width	number of pixels/line after video scaler, even value downscaling possible, total upscale factor limited to about 1.17x max: ORL_PAL_FRAME_WIDTH/ ORL_NTSC_FRAME_WIDTH
scale_height	number of lines/frame after video scaler, even value. downscaling possible, total upscale factor limited to about 1.17x max: ORL_PAL_FRAME_HEIGHT/ ORL_NTSC_FRAME_HEIGHT
fieldarrange	how video fields are placed in buffers • ORL_FLD_INTERLACED odd and even lines are interlaced • ORL_FLD_DUAL odd and even lines not interlaced • ORL_FLD_SINGLE only one video field per buffer
format	output of the color space convertor • ORL_PIXFRMT_YCBCR422 YCbCr 4:2:2 format, 16 bit/pixel, 1 plane • ORL_PIXFRMT_YCBCR422P 3 planes: plane0: Y plane1: Cb

item	description
	plane2: Cr • ORL_PIXFRMT_Y Y only, 8 bit/pixel, 1 plane • ORL_PIXFRMT_RGB565 16-bit RGB format, 1 plane • ORL_PIXFRMT_RGB888 24-bit RGB format (packed), 1 plane
inp_buff_size	returns number of bytes of an input buffer (on board)
outp_buff_size	returns number of bytes of an output buffer (on board)
host_numplanes	returns number of planes which is needed for a host buffer
host_pl0	
○ buff_size	returns needed host buffer size of plane 0
○ npages	returns needed physical memory pages
○ hor_bytes	returns number of bytes per line
host_pl1	
○ buff_size	id for plane1
○ npages	
host_pl2	
○ buff_size	id for plane2
○ npages	
hstream	returns handle of the video stream if video stream is successfully created

This function creates and initializes a video stream from one analog input module of the orlando board to the PCI Express/PC104 interface (host) (see Figure 50).

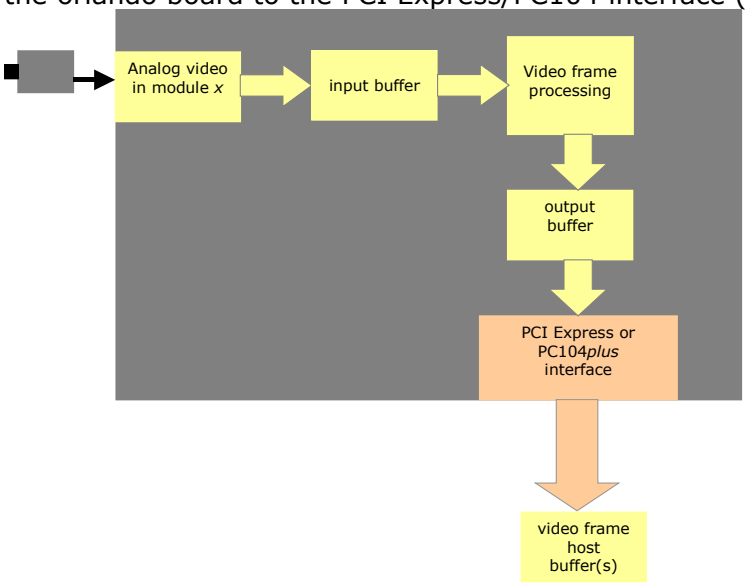


Figure 50 analog video in to host stream

The video data is acquired by the video input module, which has various settings:

- input port of the video multiplexer
- video timing
- gain
- brightness, contrast saturation

- Region Of Interest
- Video scaler

Next the video data will be converted in the Video frame processing module. The video output format of this module can be changed by item 'format'.

If this function returns successfully, parameter hstream contains a handle of the newly created stream. The stream state is set to ORL_STRSTATE_INIT (see Get video stream info 153).
After a stream is initialized you should create stream buffers for it, see Allocate on board buffers 158 and Allocate host buffer 123.

The resulting frames can be transferred from the on board output buffers to the host (page 130). After a transfer, the output buffer can be used for new frames. If there are not enough output buffers, the oldest frame will be overwritten by a new frame. This will be indicated by a 'skipped frame' (ORL_STRINT_FRM_SKIPPED) video stream interrupt (see Initialize stream interrupts 155).

example

```
/* initialize a video input to host stream, NTSC video timing */

SOrlVideoInputToHostStream   in2host;
ORLINITSTRUCT( in2host );

/* initialize analog video input */
in2host.input.module = ORL_MODULE_VIDEO_IN0;/* input module 0 */
in2host.input.port = ORL_PORT_CVBS0; /* use composite input 0 */
in2host.input.vtime = ORL_VTIME_NTSC;
in2host.input.gain_mode = ORL_VGAIN_FIXED_DEFAULTS;/* fixed gain at
default values */
/* in2host.input.gain_y_cvbs = */
/* in2host.input.gain_c = */
in2host.input.bcs_mode = ORL_VBCS_DEFAULTS;/* Bright./Contr./Satur. at
default values */
/* in2host.input.brightness = */
/* in2host.input.contrast = */
/* in2host.input.saturation = */
/* in2host.input.hue = */
in2host.input.flags = 0;

/* initialize Region Of Interest and Scaler output dimensions */
in2host.roi.xoffs = 2; /* skip #pixels */
in2host.roi.yoffs = 34;/* skip #lines */
in2host.roi.width = ORL_NTSC_WIDTH;/* #pixels/line */
in2host.roi.height = ORL_NTSC_FRAME_HEIGHT;/* #lines/frame */

in2host.scale_width = in2host.roi.width; /* don't use scaler */
in2host.scale_height = in2host.roi.height;

in2host.fieldarrange = ORL_FLD_INTERLACED;/* interlaced frames */
in2host.format = ORL_PIXFRMT_RGB888; /* 24-bit/pixel RGB */

ret = OrlFunc( H, ORLF_INIT_VIDEO_TO_HOST_STREAM, &in2host );
```

11.2.2 Initialize host to video output stream

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_HOST_TO_VIDEO_STREAM`

parameter

struct SOrlHostToVideoOutputStream

item	type	direction
output	struct SOrlVideoOutputSett	
○ module	OrlModule	in
○ port	OrlPort	in
○ vtime	OrlVideoTime	in
○ flags	ArvDWORD	in
outp_region	struct SOrlRect	
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in
○ height	ArvDWORD	in
fieldarrange	OrlFieldArrange	in
format	OrlPixelFormat	in
inp_buff_size	ArvDWORD	out
outp_buff_size	ArvDWORD	out
host_numplanes	ArvDWORD	out
host_pl0	struct SOrlPlaneSize	out
○ buff_size	ArvDWORD	out
○ npages	ArvDWORD	out
○ hor_bytes	ArvDWORD	out
host_pl1	struct SOrlPlaneSize	
(same as host_pl0)		
host_pl2	struct SOrlPlaneSize	
(same as host_pl0)		
hstream	OrlStrHandle	out

see also

Initialize video input to host stream 137
Initialize video input to output stream 147
Pixel formats in memory 170

description

item	description
output	
○ module	analog output module of video stream • ORL_MODULE_VIDEO_OUT0 • ORL_MODULE_VIDEO_OUT1
○ port	video port of the specified output module • ORL_PORT_CVBS0: Composite output 0 • ORL_PORT_CVBS1: Composite output 1 • ORL_PORT_YC0: S-Video output 0 • ORL_PORT_CVBS_DUAL: Composite output 0 and 1 • ORL_PORT_CVBS_YC: Composite output 1 and S-Video output 0 • ORL_PORT_RGB0_SYNC: RGB (with sync) output
○ vtime	Generated analog video output timing • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video
○ flags	Video output options, bitwise parameter (OR-ed) • ORL_FLAG_POWER_DOWN: sets the Digital Analog Coverter to power down mode • ORL_FLAG_NO_COLOR Switches the color information at the video output off
outp_region	
○ xoffs	number of pixels to skip at the output
○ yoffs	number of lines to skip at the output
○ width	number of pixels/line of the on board input buffer (and which is displayed) max: ORL_PAL_WIDTH/ORL_NTSC_WIDTH
○ height	number of lines/frame of the on board input buffer (and which is displayed) max: ORL_PAL_FRAME_HEIGHT/ ORL_NTSC_FRAME_HEIGHT
fieldarrange	how the video fields are placed in the input buffers • ORL_FLD_INTERLACED odd and even lines are interlaced • ORL_FLD_DUAL odd and even lines not interlaced • ORL_FLD_SINGLE only one video field per buffer
format	pixel format in the input buffer: • ORL_PIXFRMT_YCBCR422 YCbCr 4:2:2 format, 16 bit/pixel, 1plane • ORL_PIXFRMT_YCBCR422P 3 planes: plane0: Y plane1: Cb plane2: Cr • ORL_PIXFRMT_Y Y only, 8 bit/pixel, 1plane • ORL_PIXFRMT_RGB565

item	description
	16-bit RGB format, 1plane ORL_PIXFRMT_RGB888 24-bit RGB format (packed) , 1plane
inp_buff_size	returns the on board input buffer size of one frame
outp_buff_size	returns the on board output buffer size of one frame
host_numplanes	returns number of planes which is needed for host buffers
host_pl0	
○ buff_size	returns needed host buffer size of plane 0
○ npages	returns needed physical memory pages
○ hor_bytes	returns the number of byter per video line
host_pl1	
(same as host_pl0)	
host_pl2	
(same as host_pl0)	
hstream	returns handle of the video stream if video stream is successfully created

This function creates and initializes a video stream from the PCI Express/PC104 interface (host memory) to the analog video output module of the orlando board (see Figure 51).

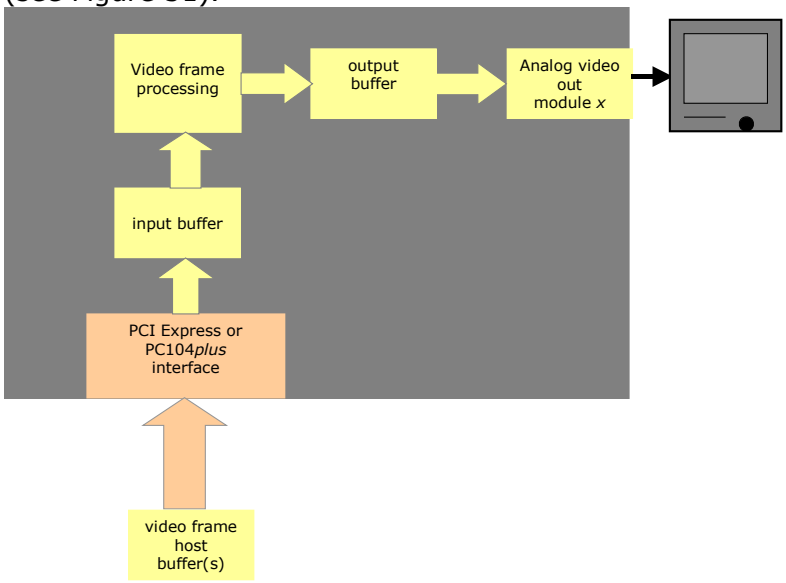


Figure 51 host to analog video out stream

The input of the video stream is the host buffer. A frame (image) should be transferred from host to the onboard input buffer.

The properties of the input image(s) should be set in next items:

- format: pixel format
- outp_region.width: number of pixels per line
- outp_region.height: number of lines
- fieldarrange

After the stream is created, parameter hstream contains the handle of the video stream. The stream state will be: ORL_STRSTATE_INIT, see Get video stream info 153.
The host_numplanes and host_pl*x* output parameters return the size of the needed host buffer(s). The host buffers can be allocated by function: Allocate host buffer 123. The on board buffers can be created by Allocate on board buffers 158. Frame transfer between host buffer and on board buffers can be done by: Transfer 130.

After a frame of the host is received it will be converted in the video frame processing module to YCbCr format. Next it will be set to analog video output module, which has next settings:

- video output port
- video timing

The analog video output shows always the latest video frame.
The input and output modules of the orlando can be used by only one video stream.

example

```
SOrlHostToVideoOutputStream  host2out;
ORLINITSTRUCT( host2out );

/* initialize analog video output */
host2out.output.module = ORL_MODULE_VIDEO_OUT0;/* output module 0 */
host2out.output.port = ORL_PORT_CVBS0; /* composite output 0 */
host2out.output.vtime = ORL_VTIME_PAL;/* video timing PAL (CCIR601) */
host2out.output.flags = 0UL;

/* set video output region at full frame */
host2out.outp_region.xoffs = 0;
host2out.outp_region.yoffs = 0;
host2out.outp_region.width = ORL_PAL_WIDTH;/* #pixels/line */
host2out.outp_region.height = ORL_PAL_FRAME_HEIGHT;/* #lines/frame */

/* set format of the (input) host buffer */
host2out.fieldarrange = ORL_FLD_INTERLACED;/* interlaced frame */
host2out.format = ORL_PIXFRMT_RGB888;/* RBB 24-bit input */

OrlFunc( H, ORLF_INIT_HOST_TO_VIDEO_STREAM, &host2out );
```

11.2.3 Initialize video input to output stream

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_VIDEO_IN_TO_OUT_STREAM`

parameter

struct SOrlInpToOutputStream

item	type	direction
input	struct SOrlVideoInputSett	
○ module	OrlModule	in
○ port	OrlPort	in
○ vtime	OrlVideoTime	in
○ gain_mode	OrlVideoGainMode	in
○ gain_y_cvbs	ArvDWORD	in
○ gain_c	ArvDWORD	in
○ bcs_mode	OrlVideoBCSMode	in
○ brightness	ArvDWORD	in
○ contrast	ArvSDWORD	in
○ saturation	ArvSDWORD	in
○ hue	ArvSDWORD	in
○ flags	ArvDWORD	in
output	struct SOrlVideoOutputSett	
○ module	OrlModule	in
○ port	OrlPort	in
○ vtime	OrlVideoTime	in
○ flags	ArvDWORD	in
roi	struct SOrlRect	
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in
○ height	ArvDWORD	in
outp_region	struct SOrlRect	
○ xoffs	ArvDWORD	in
○ yoffs	ArvDWORD	in
○ width	ArvDWORD	in
○ height	ArvDWORD	in
inp_buff_size	ArvDWORD	out
outp_buff_size	ArvDWORD	out
hstream	OrlStrHandle	out

see also
Initialize video input to host stream 137
Initialize host to video output stream 143
Video acquisition 24

description

item	description
input	
○ module	input module of stream • ORL_MODULE_VIDEO_IN0 • ORL_MODULE_VIDEO_IN1
○ port	video input port of the specified input module • ORL_PORT_CVBS0: Composite input 0 • ORL_PORT_CVBS1: Composite input 1 • ORL_PORT_CVBS2: Composite input 2 • ORL_PORT_CVBS3: Composite input 3 • ORL_PORT_CVBS4: Composite input 4 • ORL_PORT_YC0: S-Video input 0 • ORL_PORT_YC1: S-Video input 1 • ORL_PORT_RGB0: RGB input
○ vtime	video timing of the input source (camera) • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video • ORL_VTIME_SECAM: SECAM (50 Hz) video
○ gain_mode	on board video gain mode • ORL_VGAIN_AUTO automatic video gain • ORL_VGAIN_FIXED fixed gain (use gain_y_cvbs and gain_c) • ORL_VGAIN_FIXED_DEFAULTS fixed gain, 0dB
○ gain_y_cvbs	gain level of composite video or Y part of S-Video, depends on port. range: 0..511: -3dB..+6dB only if gain_mode is ORL_VGAIN_FIXED
○ gain_c	gain levels of C part of S-Video range: 0..511: -3dB..+6dB only if port is ORL_PORT_YC*x* and gain_mode is ORL_VGAIN_FIXED
○ bcs_mode	Bright/Contr/Satur/Hue mode • ORL_VBCS_NORMAL Bright/Contr/Satur/Hue can be changed by next parameters • ORL_VBCS_DEFAULTS Bright/Contr/Satur/Hue are set to default values (ITU levels)
○ brightness	Luminance brightness control range: 0..255: dark..bright default: 128
○ contrast	Luminance contrast control range: -128..127: -2(inverse)..+1.984 default: 68 (1.063)
○ saturation	Chrominance saturation control

item	description
	range: -128..127: -2(inverse)..+1.984 default: 64 (1.0)
○ hue	Chrominance hue control range: -128..127: -180°..+178.6° default: 0 (not if port ORL_PORT_RGB0 is used)
○ flags	bitwise flags • ORL_FLAG_MIRROR video lines will be mirrored • ORL_FLAG_POWER_DOWN ADC of input will be set to power down mode • ORL_FLAG_INV_FID odd-even field detection will be inverted
output	
○ module	analog output module of video stream • ORL_MODULE_VIDEO_OUT0 • ORL_MODULE_VIDEO_OUT1
○ port	video port of the specified output module • ORL_PORT_CVBS0: Composite output 0 • ORL_PORT_CVBS1: Composite output 1 • ORL_PORT_YC0: S-Video output 0 • ORL_PORT_CVBS_DUAL: Composite output 0 and 1 • ORL_PORT_CVBS_YC: Composite output 1 and S-Video output 0 • ORL_PORT_RGB0_SYNC: RGB (with sync) output
○ vtime	Generated analog video output timing • ORL_VTIME_NTSC: NTSC (60 Hz) video • ORL_VTIME_PAL: PAL (50 Hz) video
○ flags	Video output options, bitwise parameter (OR-ed) • ORL_FLAG_POWER_DOWN: sets the Digital Analog Coverter to power down mode • ORL_FLAG_NO_COLOR Switches the color information at the video output off
roi	
○ xoffs	number of pixels which will be skipped range: 2..720, even value
○ yoffs	number of lines which will be skipped
○ width	number of pixels/line to acquire even value max: ORL_PAL_WIDTH/ORL_NTSC_WIDTH
○ height	number of lines/frame (field) to acquire even value max: ORL_PAL_FRAME_HEIGHT/ ORL_NTSC_FRAME_HEIGHT
outp_region	
○ xoffs	number of pixels to skip at the output
○ yoffs	number of lines to skip at the output
○ width	number of pixels/line of the input buffer (and which is displayed) max: ORL_PAL_WIDTH/ORL_NTSC_WIDTH

item	description
○ height	number of lines/frame of the input buffer (and which is displayed) max: ORL_PAL_FRAME_HEIGHT/ ORL_NTSC_FRAME_HEIGHT
inp_buff_size	returns the on board input buffer size of one frame
outp_buff_size	returns the on board output buffer size of one frame
hstream	returns the stream handle if stream is successfully created

After calling this function, a video stream is created from an input module to an output module of the orlando board (see Figure 52).

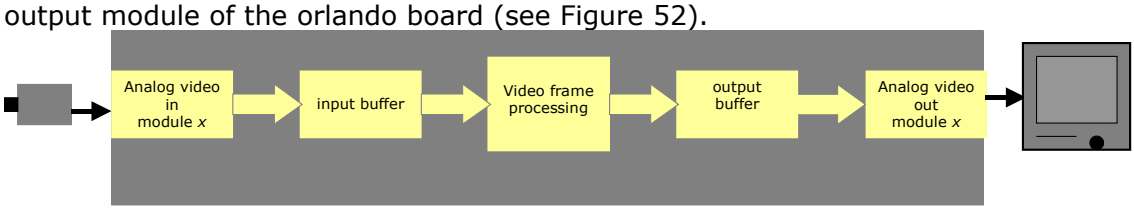


Figure 52 analog video in to video out stream

If this function was successful, parameter hstream contains the stream handle. The state of the stream is: ORL_STRSTATE_INIT, see Get video stream info 153.
Parameter roi sets the Region of Interest at the video input. Parameter outp_region sets the dimensions of the video frame at the output and the output of the video scaler.
It is permitted to use PAL video at the input and use NTSC video at the output or vice versa. This can be done by items input.vtime and output.vtime.
You should create on board buffers before using this stream (Allocate on board buffers 158).

example

```
SOrlInpToOutputStream        in2out;
ORLINITSTRUCT( in2out );

/* initialize analog video input */
in2out.input.module = ORL_MODULE_VIDEO_IN1; /* use input module 1 */
in2out.input.port = ORL_PORT_CVBS0; /* use composite input 0 */
in2out.input.vtime = ORL_VTIME_PAL;
in2out.input.gain_mode = ORL_VGAIN_FIXED_DEFAULTS;/* fixed gain at
default values */
/* in2out.input.gain_y_cvbs = */
/* in2out.input.gain_c = */
in2out.input.bcs_mode = ORL_VBCS_DEFAULTS;/* Bright./Contr./Satur. at
default values */
/* in2out.input.brightness = */
/* in2out.input.contrast = */
/* in2out.input.saturation = */
/* in2out.input.hue = */
in2out.input.flags = 0;

/* initialize Region Of Interest and Scaler output dimensions */
in2out.roi.xoffs = 2; /* skip #pixels */
in2out.roi.yoffs = 34;/* skip #lines */
in2out.roi.width = ORL_PAL_WIDTH;/* #pixels/line */
in2out.roi.height = ORL_PAL_FRAME_HEIGHT;/* #lines/frame */
```

```
/* full frames at video output */
in2out.outp_region.xoffs = 0;
in2out.outp_region.yoffs = 0;
in2out.outp_region.width = in2out.roi.width;
in2out.outp_region.height = in2out.roi.height;

/* initialize analog video output */
in2out.output.module = ORL_MODULE_VIDEO_OUT1;/* use output module 1 */
in2out.output.port = ORL_PORT_CVBS0; /* composite output 0 */
in2out.output.vtime = ORL_VTIME_PAL;/* video timing PAL (CCIR601) */
in2out.output.flags = 0UL;

OrlFunc( hbrd, ORLF_INIT_VIDEO_IN_TO_OUT_STREAM, &in2out );
```

11.2.4 Enable/disable video stream

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_ENABLE_STREAM`

parameter

struct SOrlStreamEnable

item	type	direction
hstream	OrlStrHandle	in
enable	ArvDWORD	in

see also

Get video stream info 153
Allocate on board buffers 158

description

item	description
hstream	Handle to a video stream
enable	0: disables stream 1: enables stream

Call this function to enable or disable a video stream. Before enabling, the stream state should be ORL_STRSTATE_READY_TO_RUN (this state will be reached by allocating on board buffers).

example

```
SOrlStreamEnable enable;

/* here: creating stream and allocating video stream buffers... */

ORLINITSTRUCT( enable );
enable.hstream = hstream;
enable.enable = 1; /* = enable stream*/

OrlFunc( H, ORLF_ENABLE_STREAM, &enable );
```

11.2.5 Get video stream info

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_GET_STREAM_INFO`

parameter

struct SOrlStreamInfo

item	type	direction
streamix	ArvDWORD	in/out
hstream	OrlStrHandle	in/out
request_is_hstream	ArvDWORD	in
inp_module	OrlModule	out
outp_module	OrlModule	out
stream_state	OrlStreamState	out

see also

Initialize video input to host stream 137
Initialize host to video output stream 143
Initialize video input to output stream 147

description

item	description
streamix	Index of a video stream. First index is '0'.
hstream	Handle of a video stream
request_is_hstream	Decides if streamix or hstream is input argument: • 0: streamix is input, hstream is output • 1: hstream is input, streamix is output
inp_module	Returns the input module of the requested video stream
outp_module	Returns the output module of the requested video stream
stream_state	Returns the state of video stream: • ORL_STRSTATE_UNKNOWN unknown state • ORL_STRSTATE_NOT_INIT stream not initialized • ORL_STRSTATE_INIT stream is initialized, without video frame buffers • ORL_STRSTATE_READY_TO_RUN stream is initialized with buffers • ORL_STRSTATE_RUN video stream is enabled

This function retrieves information of a video stream. The requested stream can be retrieved by an index or stream handle.

example

```
SOrlStreamInfo    strinfo;
char  strbuff[256];
ORLINITSTRUCT( strinfo );

strinfo.streamix = 0;/* stream index 0 */
strinfo. request_is_hstream = 0;/* request is stream index */

OrlFunc( H, ORLF_GET_STREAM_INFO, &strinfo );

OrlEnumToStr( ORLSTREAMSTATE, strinfo.stream_state, strbuff, 256 );
printf( "Steam ix: %lu: stream handle: %lu, state: %s",
      strinfo.streamix,
      strinfo.hstream,
      strbuff );
```

11.2.6 Initialize stream interrupts

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_INTERR_STREAM`

parameter

struct SOrlInitInterrStream

item	type	direction
hstream	OrlStrHandle	in
interr_mask	OrlStrInterrType	in

see also

Initialize board interrupts 132
Wait on video stream interrupt 157

description

item	description
hstream	Handle of a video stream. The video stream shouldn't be enabled
interr_mask	Interrupt mask. If this value is '0' the stream handling will be disabled. Next items can be OR-ed: • ORL_STRINT_FRM_IN interrupt after a frame is acquired at the video input (from input module or host) • ORL_STRINT_FRM_RDY_FOR_OUT interrupt if a frame is ready for output • ORL_STRINT_FRM_OUT interrupt after a frame is set to the video output (to output module or host) • ORL_STRINT_FRM_SKIPPED interrupt if a frame is skipped in the video stream (eg because the output buffer is full) • ORL_STRINT_OVL_TRANSF_RDY interrupt after the host has transferred an overlay buffer to the onboard overlay buffer

This function initializes the video stream interrupts. It can be used to e.g. count the number of frames acquired. If an interrupt occur, function ORLF_WAIT_INTERR_STREAM will unblock.

example

```
void Init()
{
      /* here: creating video stream input to host
      (ORLF_INIT_VIDEO_IN_TO_OUT_STREAM), and allocated buffers for it
      */

      SOrlInitInterrStream   interr;
      ORLINITSTRUCT( interr );

      interr.hstream = hstream; /* stream handle */
      interr.interr_mask = ORL_STRINT_FRM_IN |
      ORL_STRINT_FRM_RDY_FOR_OUT | ORL_STRINT_FRM_OUT;

      OrlFunc( H, ORLF_INIT_INTERR_STREAM, &interr );

      /* here: create an thread which handles the stream interrupts.
            CreateThread( ..., StrInterrHandling, ... );
      */
}

/* thread function which handles interrupts of an orlando video stream
*/
void StrInterrHandling()
{
      SOrlWaitInterrStream   waitstr;
      ORLINITSTRUCT( waitstr );
      waitstr.hstream = hstream;/* stream handle which generates
                                              interrupts */

      do {
            /* wait until an interrupt occur */
            OrlFunc( H, ORLF_WAIT_INTERR_STREAM, &waitstr );

            /* check if interrupt handling is stopped */
            if ( !waitstr.stopped )
            {
                  /* check which interrupts are occured
                     between two ORLF_WAIT_INTERR_STREAM calls
                  */

                  /* now only print it */
                  printf( "#frms in: %lu\n", waitstr.num_frms_in );
                  printf( "#frms ready for out: %lu\n",
                                    waitstr.num_frms_rdy_for_out );
                  printf( "#frms out: %lu\n", waitstr.num_frms_out );
            }
      } while ( !waitstr.stopped );
}
```

11.2.7 Wait on video stream interrupt

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_WAIT_INTERR_STREAM`

parameter

struct SOrlWaitInterrStream

item	type	direction
hstream	OrlStrHandle	in
stopped	ArvDWORD	out
num_frms_in	ArvDWORD	out
num_frms_rdy_for_out	ArvDWORD	out
num_frms_out	ArvDWORD	out
num_frms_skipped	ArvDWORD	out
num_ovl_in	ArvDWORD	out

see also

Initialize stream interrupts 155

description

item	description
hstream	waits on interrupts of this stream handle
stopped	indicates if interrupt handling is stopped: 0: interrupt handling is running 1: interrupt handling is stopped or not initialized, don't call ORLF_WAIT_INTERR_STREAM again
num_frms_in	number of frames between previous call and this call which are acquired at the input of the stream
num_frms_rdy_for_out	number of frames between previous call and this call which are ready for the stream ouput
num_frms_out	number of frames between previous call and this call which are put to the output of the stream
num_frms_skipped	number of frames between previous call and this call which are skipped (eg not enough output buffers)
num_ovl_in	number of overlay buffers transferred from host to stream between previous call and this call

This function should be called after initializing stream interrupts. ORLF_WAIT_INTERR_STREAM blocks until one or more stream interrupts are occurred. The various counters (num_xxxx) gives the number of interrupts occurred between two ORLF_WAIT_INTERR_STREAM calls.

example

See page 137.

11.2.8 Allocate on board buffers

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_ALLOC_BUFFS`

parameter

struct SOrlAllocBuff

item	type	direction
hstream	OrlStrHandle	in
inp_onboard_buffs	ArvDWORD	in
outp_onboard_buffs	ArvDWORD	in

see also

Initialize video input to host stream 137
Initialize host to video output stream 143
Initialize video input to output stream 147
Enable/disable video stream 152
Get video stream info 153
Allocate host buffer 123

description

item	description
hstream	video stream handle which need buffers
inp_onboard_buffs	number of buffers at the video stream input
outp_onboard_buffs	number of buffers at the video stream output

After a video stream is initialized, you should create on board buffers for it. These buffers are allocated in the on board memory of the orlando and organized as a FIFO (First In First Out). Take a minimum of three buffers at the input and output of a stream.
If this function returns successful (ORL_RET_SUCCESS), the stream state will be ORL_STRSTATE_READY. This means that the video stream can be enabled.

example

```
SOrlAllocBuff     buff;
ORLINITSTRUCT( buff );

buff.hstream = hstream;
buff.inp_onboard_buffs = 3;
buff.inp_onboard_buffs = 3;

OrlFunc( H, ORLF_ALLOC_BUFFS, &buff );
```

11.2.9 Free stream resources

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_FREE_STREAM_RESOURCES`

parameter

struct SOrlFreeStrRes

item	type	direction
hstream	OrlStrHandle	in

see also

Initialize video input to host stream 137
Initialize host to video output stream 143
Initialize video input to output stream 147

description

item	description
hstream	video stream handle wich should be free'd

This function disables a video stream and frees all used resources. The interrupt stream handling will be stopped too.
After calling this function the stream state will be: ORL_STRSTATE_NOT_INIT.

11.2.10 Initialize overlay

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_OVERLAY`

parameter

struct SOrlInitOverlay

item	type	direction
hstream	OrlStrHandle	in
enable	ArvDWORD	in
vw_annot	struct SOrlAnnotView	
○ enable	ArvDWORD	in
○ posrx	ArvDWORD	in
○ posty	ArvDWORD	in
○ font	OrlFontType	in
vw_user	struct SOrlUsrOvlView	
○ width	ArvDWORD	out
○ height	ArvDWORD	out
○ plbuff	struct SOrlPlaneSize	
○ • buff_size	ArvDWORD	out

see also

Initialize frame annotation 165
Transfer overlay host buffer to stream 168

description

item	description
hstream	handle to video stream
enable	enables (1) or disables (0) overlay
vw_annot	
○ enable	enables (1) or disables (0) frame annotation value in overlay
○ posrx	overlay buffer x-coordinate of the right side of the annotation value range: 0..360 (ORL_OVL_WIDTH)
○ posty	overlay buffer y-coordinate of the top of the annotation value PAL timing range: 0..288 (ORL_OVL_PAL_HEIGHT) NTSC timing range: 0..242 (ORL_OVL_NTSC_HEIGHT)
○ font	font type of annotion value • ORL_STDFONT_8X8 • ORL_STDFONT_16X16 • ORL_STDFONT_24X24
vw_user	

○ width	returns the needed width of the on board overlay buffer, in bytes
○ height	returns the needed height of the on board overlay buffer
○ plbuff	
○ • buff_size	returns the needed size of the overlay host buffer

This function enables (or disables) video overlay. The video overlay is added on board and consists of two parts:
- frame annotator
- user overlay

Frame annotation is described in Initialize frame annotation 165. Function ORLF_INIT_OVERLAY offers the possibility to write the annotator value in overlay, see next figures. This is processed on board.

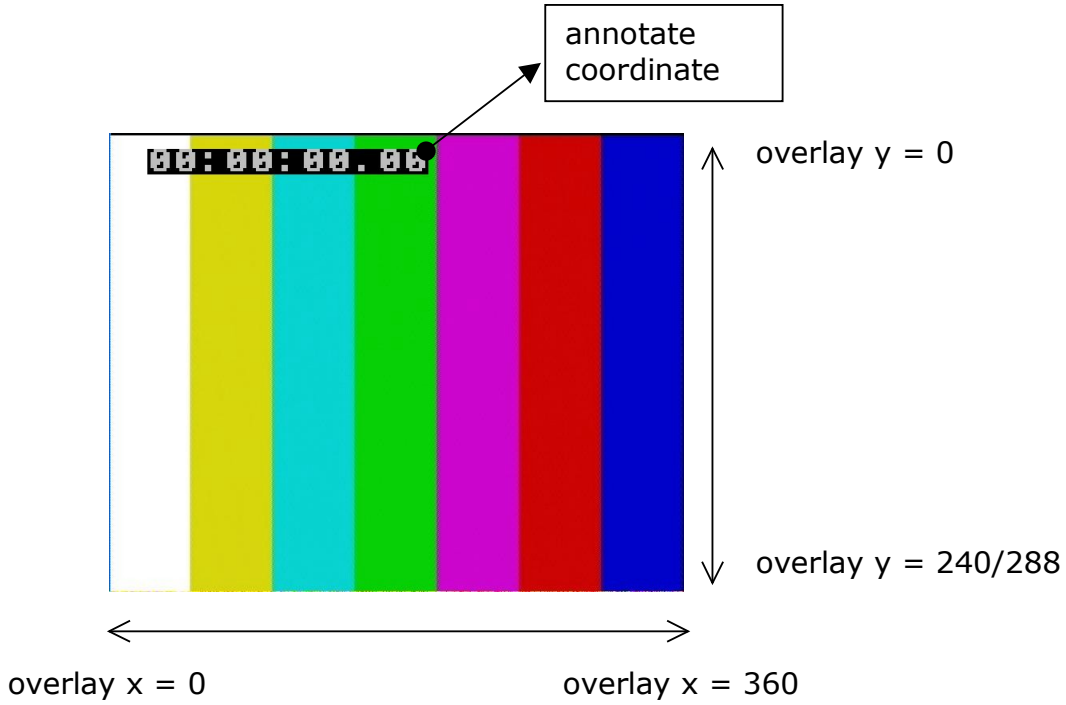
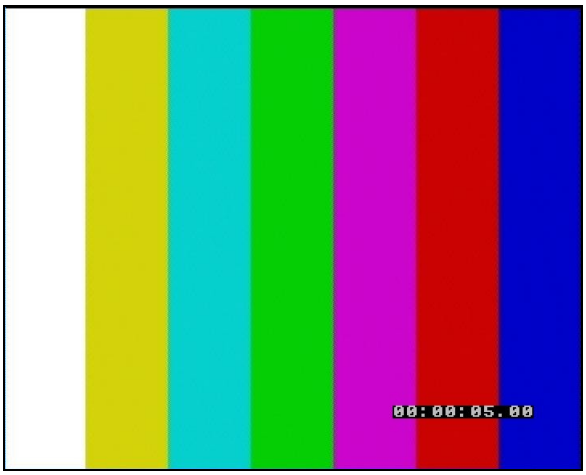


Figure 53 annotate in overlay at (200,10), STDFONT_16X16, FRM_COUNTER


00:00:05.00

User overlay can be created in an overlay buffer at the host. After creating overlay the host buffer can be transferred to the on board overlay buffer and will be added to the video frames.
The host overlay buffer should be filled with color indexes. Next color indexes can be used:

```
typedef enum OrlOvlColorIxEnum
{
   ORL_OVL_COLOR_TRANS = 0,
/* transparent (video) */
   ORL_OVL_COLOR_WHITE = 1,
/* various colors */
```

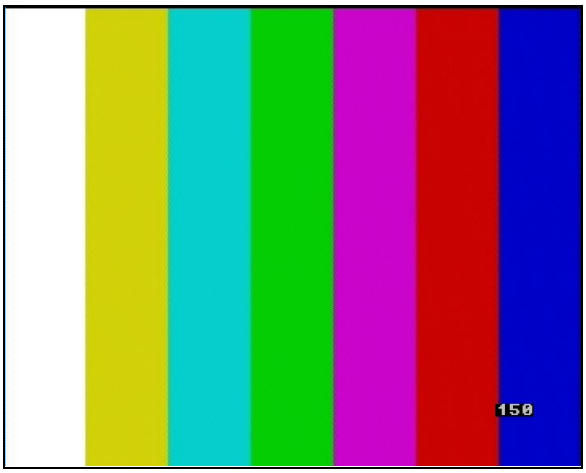


Figure 55 annotate in overlay at (330,248), STDFONT_8X8, COUNTER

Figure 54 annotate in overlay at (330,248), STDFONT_8X8, FRM_COUNTER

```
ORL_OVL_COLOR_YELLOW = 2,
   ORL_OVL_COLOR_CYAN = 3,
   ORL_OVL_COLOR_GREEN = 4,
   ORL_OVL_COLOR_MAGENTA = 5,
   ORL_OVL_COLOR_RED = 6,
   ORL_OVL_COLOR_BLUE = 7,
   ORL_OVL_COLOR_BLACK = 8,
   ORL_OVL_COLOR_GREY1 = 9,            /* light grey */
   ORL_OVL_COLOR_GREY2 = 10,
   ORL_OVL_COLOR_GREY3 = 11,
   ORL_OVL_COLOR_GREY4 = 12,           /* very dark grey */
   ORL_OVL_COLOR_RES1 = 13,            /* reserved */
   ORL_OVL_COLOR_RES2 = 14,            /* reserved */
   ORL_OVL_COLOR_RES3 = 15,            /* reserved */
   NUM_ORL_OVL_COLORS
} OrlOvlColor;
```

The size of a color index is one byte.

Figure 56 shows a video frame with user overlay and the annotator value.

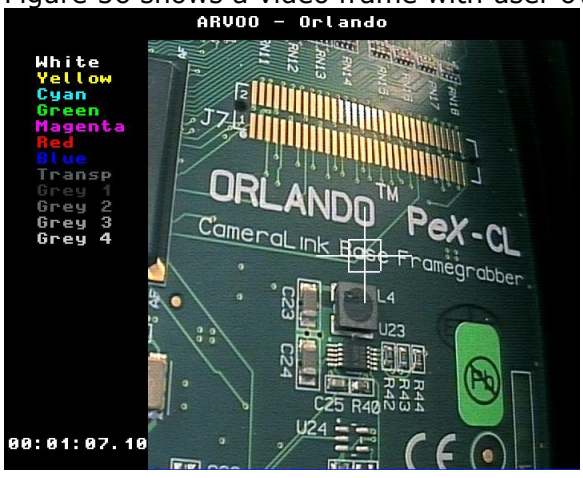


Figure 56 User overlay and frame annotator

example
see Transfer overlay host buffer to stream 168

11.2.11 Initialize frame annotation

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_INIT_ANNOTATION`

parameter

struct SOrlInitFrmAnnot

item	type	direction
hstream	OrlStrHandle	in
enable	ArvDWORD	in
type	OrlFrmAnnotType	in
value	struct SOrlFrmAnnotValue	
○ val0	ArvDWORD	in
○ val1	ArvDWORD	in
○ val2	ArvDWORD	in
○ val3	ArvDWORD	in

see also

Initialize overlay 160
Retrieve host plane buffer info 125

description

item	description
hstream	handle of video stream
enable	enables(1) or disables(0) frame annotation
type	annotation type • ORL_ANNOT_COUNTER counter value, start with '0', next: 1, 2,... • ORL_ANNOT_FRM_COUNTER frame counter, format: [hh]:[mm]:[ss].[ff] hh: hours, range 00.. >1000 mm: minutes, range: 00..59 ss: seconds, range: 00..59 ff: frames, range: 00..24 (PAL), 00..29 (NTSC)
val0	if ORL_ANNOT_COUNTER: start value of the counter if ORL_ANNOT_FRM_COUNTER: start value of hours
val1	only if ORL_ANNOT_FRM_COUNTER start value of minutes
val2	only if ORL_ANNOT_FRM_COUNTER start value of seconds
val3	only if ORL_ANNOT_FRM_COUNTER start value of frames

This function initializes the on board frame annotator of a video stream. At the input of a video stream, each new video frame will be annotated (gets a number). There are two annotation types: ORL_ANNOT_COUNTER and ORL_ANNOT_FRM_COUNTER. ORL_ANNOT_COUNTER counts 0, 1, 2, 3... ORL_ANNOT_FRM_COUNTER counts from 00:00:00.00, 00:00:00.01,..., 00:00:00.24, 00:00:01.00 etc.
If a annotated video frame is transferred to the host, the frame annotator value can be retrieved by function ORLF_GET_PL_HBUFFINFO; parameter SOrlPlHostBuffInfo. annotstr, see page 125.
The annotator value can be written in the video overlay, see page 160.

example

```
SOrlInitFrmAnnot  annot;
ORLINITSTRUCT( annot );

annot.hstream = hstream;
annot.enable = 1;
annot.type = ORL_ANNOT_FRM_COUNTER;
/* start with "05:43:15.00" */
annot.value.val0 = 05UL;/* hour */
annot.value.val1 = 43UL;/* min */
annot.value.val2 = 15UL;/* secs */
annot.value.val3 = 00UL;/* frms */

OrlFunc( H, ORLF_INIT_ANNOTATION, &annot );
```

11.2.12 Change annotation value

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_CHG_ANNOTATION_VALUE`

parameter

struct SOrlFrmAnnotValue

item	type	direction
hstream	OrlStrHandle	in
val0	ArvDWORD	in
val1	ArvDWORD	in
val2	ArvDWORD	in
val3	ArvDWORD	in

see also

Initialize frame annotation 165

description

item	description
hstream	video stream handle which contains frame annotation
val0	counter value or hours
val1	minutes
val2	seconds
val3	frames

This function can be used to change the current frame annotator counter. It depends on the frame annotation type (see Initialize frame annotation 165) what the items val0..val3 means.

example

```
SOrlFrmAnnotValue annotval;
ORLINITSTRUCT( annotval );

/* here init of frame annotator, type is ORL_ANNOT_COUNTER */

/* set annotator counter to 12345 */
annotval.hstream = hstream;
annotval.val0 = 12345UL;
OrlFunc( H, ORLF_CHG_ANNOTATION_VALUE, &annotval );
```

11.2.13 Transfer overlay host buffer to stream

OBSOLETE

supported boards

CL	AN
N	Y

OrlFuncType

`ORLF_USEROVL_TRANSF`

parameter

struct SOrlFrmTransf

item	type	direction
hstream	OrlStrHandle	in
buffernr	ArvDWORD	in

see also

Initialize overlay 160
Initialize stream interrupts 155
Allocate host buffer 123

description

item	description
hstream	video stream handle which is destination of host overlay buffer
buffernr	host buffer index this should be the host overlay buffer

Transfers a host overlay buffer to the video stream. After transfer, an interrupt is generated and the new overlay data will be used in the video stream.

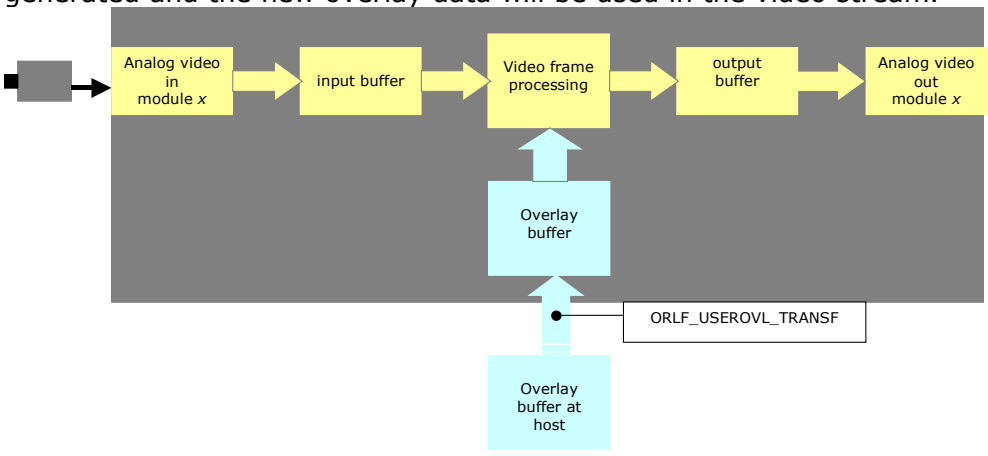


Figure 57 host overlay buffer transfer

example

```
/*
      adds annotation and user overlay to an existing
      video stream
```

```
*/
/* video stream should exist, stream handle: hstream */
SOrlInitFrmAnnot        annotation;
SOrlInitOverlay         overlay;
SOrlAllocPlHostBuff     ovlbuff;
SOrlFrmTransf           ovl_transf;

/* create frame annotation */
ORLINITSTRUCT( annotation );
annotation.hstream = hstream;
annotation.enable = 1;  /* enable */
annotation.type = ORL_ANNOT_COUNTER;
annotation.value.val0 = 0UL;
OrlFunc( H, ORLF_INIT_ANNOTATION , &annotation );

/* initialize on board overlay */
ORLINITSTRUCT( overlay );
overlay.hstream = hstream;
overlay.enable = 1;

/* display annotate value in overlay */
overlay.vw_annot.enable = 1; /* enable frame annotation in overlay */
overlay.vw_annot.posrx = ORL_OVL_WIDTH - 30; /* x right position */
overlay.vw_annot.posty = ORL_OVL_NTSC_HEIGHT - 40; /* y top position */
overlay.vw_annot.font = ORL_STDFONT_8X8;

OrlFunc( H, ORLF_INIT_OVERLAY, &overlay );

/* create overlay host buffer */
ORLINITSTRUCT( ovlbuff );
ovlbuff.numplanes = 1;
ovlbuff.plane0.sz = overlay.vw_user.plbuff;/* copy dimensions from
overlay init */
OrlFunc( H, ORLF_ALLOC_HBUFF, &ovlbuff );

/* make overlay buffer transparent */
memset( ovlbuff.plane0.pbuff, ORL_OVL_COLOR_TRANS,
ovlbuff.plane0.sz.buff_size );

/* todo: write overlay host buffer
 buffer width: overlay.vw_user.width
 buffer height: overlay.vw_user.height
 Use color indexes of of enum OrlOvlColor as data. Each byte (color
index) in the buffer is one pixel */

/* transfer overlay buffer to (on board) video stream */
ORLINITSTRUCT( ovl_transf );
ovl_transf.hstream = hstream;
ovl_transf.buffernr = ovlbuff.buffernr;/* index of overlay host buffer
*/
OrlFunc( H, ORLF_USEROVL_TRANSF, &ovl_transf );
```

12 Appendix

12.1 Pixel formats in memory

12.1.1 ORL_PIXFRMT_YCBCR422

byte 3	byte 2	byte 1	byte 0
Y1	Cr0	Y0	Cb0
Y3	Cr1	Y2	Cb1

12.1.2 ORL_PIXFRMT_YCBCR422P

plane0

byte 3	byte 2	byte 1	byte 0
Y3	Y2	Y1	Y0

plane1

byte 3	byte 2	byte 1	byte 0
Cb3	Cb2	Cb1	Cb0

plane2

byte 3	byte 2	byte 1	byte 0
Cr3	Cr2	Cr1	Cr0

12.1.3 ORL_PIXFRMT_Y

byte 3	byte 2	byte 1	byte 0
Y3	Y2	Y1	Y0

12.1.4 ORL_PIXFRMT_Y10

bit [31:26]	bit [25:16]	bit [15:10]	bit [9:0]
dummy	Y1	dummy	Y0

12.1.5 ORL_PIXFRMT_Y12

bit [31:28]	bit [27:16]	bit [15:12]	bit [11:0]
dummy	Y1	dummy	Y0

12.1.6 ORL_PIXFRMT_Y14

bit [31:30]	bit [29:16]	bit [15:14]	bit [13:0]
dummy	Y1	dummy	Y0

12.1.7 ORL_PIXFRMT_Y16

bit [31:16]	bit [15..0]
Y1	Y0

12.1.8 ORL_PIXFRMT_RGB565

bit[15:11]	bit [10:5]	bit [4:0]
G0	R0	B0
G1	R1	B1

12.1.9 ORL_PIXFRMT_RGB888

byte 3	byte 2	byte 1	byte 0
B1	R0	G0	B0
G2	B2	R1	G1

12.2 YCbCr values

The Orlando AN board uses YCbCr 4:2:2 values. Some API structs items needs a YCbCr value for eg. background color.

The YCbCr format is:

byte 3	byte 2	byte 1	byte 0
	Y	Cb	Cr

Ranges are:
Y: 16..235
Cb: 16..240
Cr: 16..240

Some colors are defined already:
ORL_YCBCR_WHITE
ORL_YCBCR_YELLOW
ORL_YCBCR_CYAN
ORL_YCBCR_GREEN
ORL_YCBCR_MAGENTA
ORL_YCBCR_RED
ORL_YCBCR_BLUE
ORL_YCBCR_BLACK

To convert a RGB value to YCbCr use next formulas:

R, G and B ranges: 0..255
Y = 0.257R + 0.504G + 0.098B + 16
Cb = -0.148R -0.291G + 0.439B + 128
Cr = 0.439R - 0.368G - 0.071B + 128

more information: http://en.wikipedia.org/wiki/YCbCr

12.3 Version changes

12.3.1 SDK 1.00

First release, Orlando CL
Supports Windows 32-bit, Linux 32-bit, QNX 6

12.3.2 SDK 1.01

add: Orlando AN supported by stream functions
fix: callback cbparm always NULL
fix: OrlVC ROI setting
name changed: orlc to orlc_cl
add: orlc_an sample

12.3.3 SDK 1.02

add: OrlInit.opt en SOrlCbParm.opt
add: more enumstrings in library
fix: Orlando AN, power down setting of ADC doesn't wok correctly
changed: Windows firmware dir stored in HKLM (was HKCU)

12.3.4 SDK 1.03

add: JNI interface to library
add: Java sample application
fix: Orlando CL: linescan blanking may 2 pixels or more
fix: orlc_an: 'ntsc' contains wrong value
fix: Orlando CL: Uart rx generates too much data
fix: ORLF_INIT_INTERR_BOARD: release of interrupts failed
add: Windows XPE component
Windows 64-bit supported

12.3.5 SDK 1.04

add: AN JPEG store YCbCr (1 plane)
add: ORLF_GET_PCI_STAT
add: orl_java: prefs stored (version 1.01)
add: CL SOrlVideoBuffSett.hbuff_elmnt_size_pages
add: CL ORLF_SET_BAYER_CONV, PSOrlBayerConv
add: CL SOrlHostBuffInfo.format
add: CL SOrlVideoAcqSett.buff_size_pages
add: OrlPixelFormat-s (Y10 etc): content info of image
fix: Orlando AN: Y/C and RGB inputs doesn't work
fix: orl_java: captureThread doesn't exit correctly when application closes
supports Orlando AN-104 rev B and PCIE rev C too

12.3.6 SDK 1.05

add: AN module functions, stream functions obsolete now
fix: AN-104: GPIO not stable
fix: AN fld1-2 sequence wrong if input 60Hz, output 50Hz
fix: AN analog output, 60Hz, only fld2
fix: load 16-bit BMP failed if BITMAPINFOHEADER.biSize!=40

fix: Y8 captures board to host doesn't work
fix: OrlVC capture handling improved
add: orl_an_mod sample application

12.3.7 SDK 1.06

fix: AN module states checked in module-h2b, module-b2h and module-in
fix: AN DAC output always CVBS0
fix: CL calculation of VE-size of sometimes wrong
fix: use cache for host capture buffers
Linux 64-bit supported