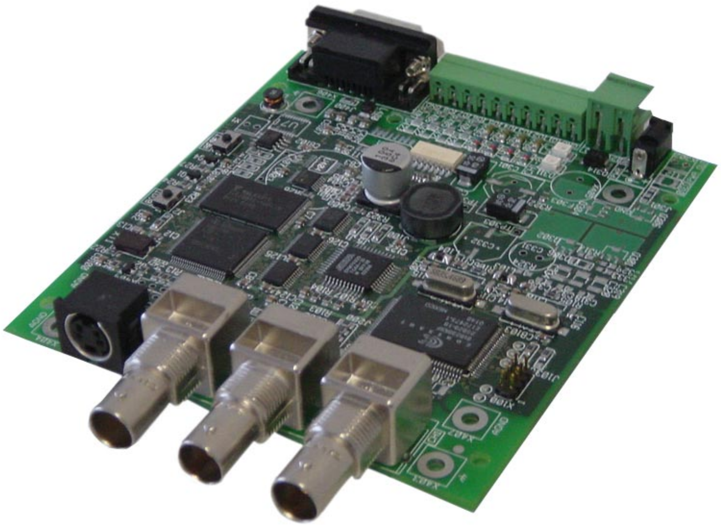
grabbMODUL-4



Hardware Manual

Edition February 2005

A product of a PHYTEC Technology Holding company

	EUROPE	NORTH AMERICA
Address:	PHYTEC Technologie Holding AG Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (800) 0749832 order@phytec.de	1 (800) 278-9913 sales@phytec.com
Technical Support:	+49 (6131) 9221-31 support@phytec.de	1 (800) 278-9913 support@phytec.com
Fax:	+49 (6131) 9221-33	1 (206) 780-9135
Web Site:	http://www.phytec.de	http://www.phytec.com

1st Edition May 2004 (German), February 2005 (English)

Index of Figures

Index of Tables

1 Introduction

1.1 About this Manual

This manual is divided into three parts:

- **Part 1 – Start Up**
  This part describes the start up of the grabbMODUL-4 with the Rapid Development Kit (Order code VPK-047). You will learn step by step how to load application firmware onto the grabbMODUL-4, how to establish the necessary electrical connections and how to transmit an image over the serial interfaces using software on your PC. (Start-Up is also possible using a video camera, whereby the Rapid Development Kit is not required. However some steps may vary from those described in the manual).

- **Part 2 – Technical Data**
  Part 2 contains information about technical data and specifications as well as connectivity options for the grabbMODUL-4.

- **Part 3 – Programmer's Manual**
  Part 3 contains a description of the driver library and an introduction to programming the grabbMODUL-4. This section is a reference for the programmer who wants to develop his or her own programs for the grabbMODUL-4 (firmware) or a program for a host PC. Requirements for this are a knowledge of the corresponding programming environment (e.g. Keil Compiler for the C165 microcontroller) and C programming language.

Hint:
Additional information can be found on our homepage www.phytec.de under SUPPORT > FAQs Bildverarbeitung (Image Processing).

1.2 Overview

The grabbMODUL-4 is a stand-alone image processor based on the Infineon C165 microcontroller.

It integrates the following components on a 100 x 110 mm printed circuit board:

- 16-bit microcontroller kernel
- Color Frame Grabber with digital video processor
- Power supply
- I/O interfaces

The grabbMODUL-4 can be used in a variety of ways depending on the application:

- **Stand-Alone System**

  <u>Application Examples:</u>
  - Quality Control: Liquid level measurement, automated inspection
  - Automation: solar panel tracking

  For a given application a special software has to be developed. This software is downloaded into and stored permanently in the grabbMODUL-4 (‘Firmware’). Based on this program, the grabbMODUL’s processor kernel is able to process image data autonomously (without additional processors).

  Image processing results can affect a process via the I/O ports or be transmitted over one of the interfaces.

  The application software is developed for the grabbMODUL’s C165 controller environment. A corresponding compiler (the Keil compiler is recommended) is required for use of the application software.
  The grabbMODUL’s specialized components, such as the Framegrabber or I/O ports, can be accessed for communication quite easily by using the included driver libraries.

PHYTEC can create the corresponding user software for your project upon request.

- **In conjunction with a Host PC**

  Application Examples:
  - Remote monitoring/maintenance of distributed systems
  - Access control, security systems

  The grabbMODUL-4 is connected to a host PC (for example over the serial interface). It captures the image data locally and performs preprocessing functions if necessary and then transmits the data to the host PC.
  This is also possible via a modem connection or radio data transmission to a remote PC. Use of existing data networks is also possible if the serial interfaces are used.

  This likewise requires firmware on the grabbMODUL-4 that communicates with the host PC in accordance with the specific task. PHYTEC supplies ready-made firmware with the grabbMODUL-4, which enables simple, uncompressed image transmission via the serial interface. *Specifications and commands for this firmware can be found in Part 3 of this manual.*

1.3 Delivery Contents

1.3.1 Delivery Contents of the grabbMODUL-4 (VM-004)

- grabbMODUL-4 PCB (VM-004)
- Driver and Demo CD (SO-670)
- PHYTEC Spectrum-CD (SO-376)
- This manual (L-612)

1.3.2 Delivery Contents of the Rapid Development Kit (VPK-047)

- grabbMODUL-4 PCB (VM-004)
- Driver and Demo CD (SO-670)
- PHYTEC Spectrum CD (SO-376)
- Color Camera VCAM-110-1 (AK039-1)
- S-Video Camera Connector Cable (WK051)
- Nullmodem Cable (WK041)
- AC adapter for grabbMODUL-4 (SV001)
- AC adapter for Camera (SV009)
- Video Lens 16mm focal length (AO012)
- Camera table mount tripod (AZ004)
- This manual (L-612)

1.4 Accessories

The following accessories can be purchased separately from PHYTEC:

- BNC camera connector cable, length 1 m, order code WK057
- BNC camera connector cable, length 2 m, order code WK058
- BNC camera connector cable, length 10 m, order code WK039
- S-Video camera connector cable, length 2 m, order code WK051
- Nullmodem cable for connection to a PC (serial interface, DB-9 socket), order code WK041
- Power adapter with coaxial supply voltage plug, 12 V, 500 mA, order code SV001
- Power adapter with open cable ends, 12 V, 500 mA, order code SV009
- Phoenix plug for I/O-Port, with straight cable outlet, order code GP130
- Phoenix plug for I/O-Port, with 90° angle cable outlet, order code GP102
- 2 post Phoenix plug for supply voltage input, with straight cable outlet, order number GP088

Part 1

Start-Up

2. Rapid Development Kit Start-Up

This section lists the clear and simple steps for the initial Start-Up of the grabbMODUL-4. Start-Up is demonstrated using the components included in a grabbMODUL-4 Rapid Development Kit. It is also possible to use the demonstration for Start-Up with a grabbMODUL-4 and custom user components.

The following steps are described in detail below:

- System Requirements
  - What are the hardware requirements?
- Interfaces and their use with the grabbMODUL-4.
  - What has to be connected to the grabbMODUL-4 and how are these connections established?
- Installation and application of PHYTEC FlashTools
  - How is software loaded onto the grabbMODUL-4?
- Download of programs in Hex-File-Format from a PC to the module’s external Flash memory
  - How is software loaded to the grabbMODUL-4?
- Testing a program in the grabbMODUL-4
  - How can I test a program in the module?

The example shown in the demo software only demonstrates a few of the possible applications using a grabbMODUL-4. In order to adapt this module to a range of applications, software has to be created that has been adapted to the application’s specific tasks. Instructions for this are described in *section 6, „Driver Software“*.

2.1 System Requirements

The following hardware components are required to use the grabbMODUL-4:

1) When using the RDK „grabbMODUL-4“:

- The PHYTEC-RDK „grabbMODUL-4“
- An IBM compatible PC (486 or higher with a Windows operating system: Windows9x/NT/2K/ME/XP) with a free serial interface

2) When using the grabbMODUL-4:

- The PHYTEC grabbMODUL-4
- A DB-9 null modem cable for program download
- Power supply 8…28 V (2W) direct current (unregulated)
- Analog video camera with a power supply and lens
- Video connector cable BNC or S-Video on camera connector
- An IBM compatible PC (486 or higher with a Windows operating system: Windows9x/NT/2K/ME/XP) with a free serial interface

In addition the included PHYTEC FlashTools and the grabbMODUL-4 demosoftware are required for Start-Up.

Hint:
If possible use a manufactured null modem cable in order to eliminate errors. Information on constructing a null modem cable can be found on the FAQ pages on our website.

2.2 Interfaces of the grabbMODUL-4

The connectors for the grabbMODUL-4 are located on the right and left sides of the board and can be accessed from the outside when surrounded by a protective housing.

- The figure shows the connectivity possibilities of the grabbMODUL-4:

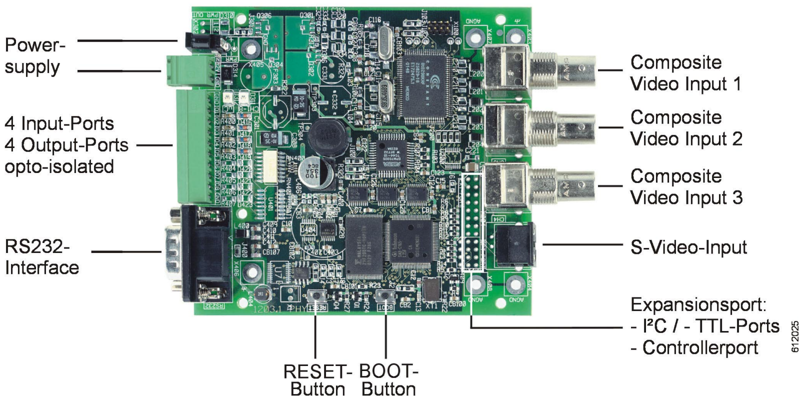


Figure 1: grabbMODUL-4 – Overview of Connectivity Possibilities

- The following connections and user elements are required for downloading a Hex file (e.g. Firmware) to the FLASH memory of the grabbMODUL-4:
- Power supply
- PC (host with a free serial interface, COM) connected via ist RS-232 interface socket over a nullmodem cable to the RS-232 interface on the grabbMODUL-4.
- BOOT / RESET button

- For operation the following connections are also required:
- Standard Video Signal connected to a composite video input or S-Video input

In its delivery state the current PHYTEC firmware „Lokal_COM“ is already present in the module’s Flash memory. For Start-Up you can go directly to *section 2.5, „Testing the Module Program LOCAL_COM“.*

2.3 Connecting the grabbMODUL-4 for „Download“

Connect the RS-232 interface (COM) on your PC with the DB-9 (RS-232) socket on the grabbMODUL-4 using a serial „null modem“ cable (WK041).

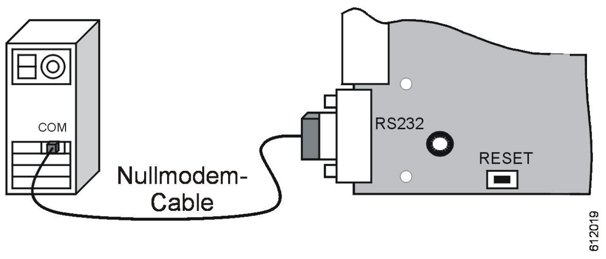


Figure 2: Connecting the grabbMODUL-4 to a PC

Connect the „PWR IN“ socket either using the NG plug (power adapter SV001) or the 2-pin Phoenix plug with an 8…28 V (2W) DC power supply. Please note the polarity of the connector plug described in *section 3.1.1*. The green LED illuminates indicating ready status.

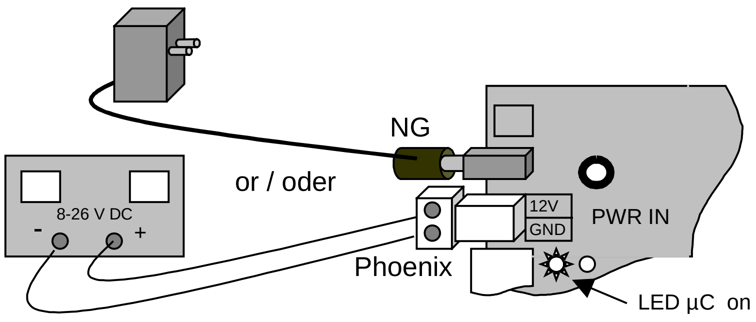


Figure 3: grabbMODUL-4 Power Connection

Press the RESET and BOOT buttons on the grabbMODUL-4 simultaneously *(see Figure 4)*. Release the RESET button first, followed 2 to 3 seconds later by the BOOT button.

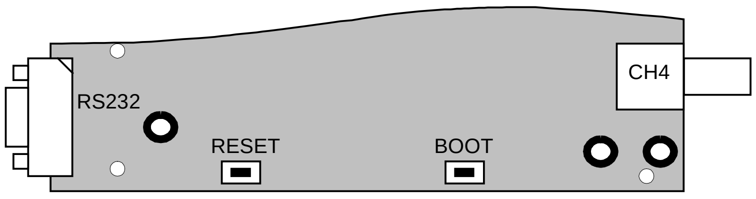


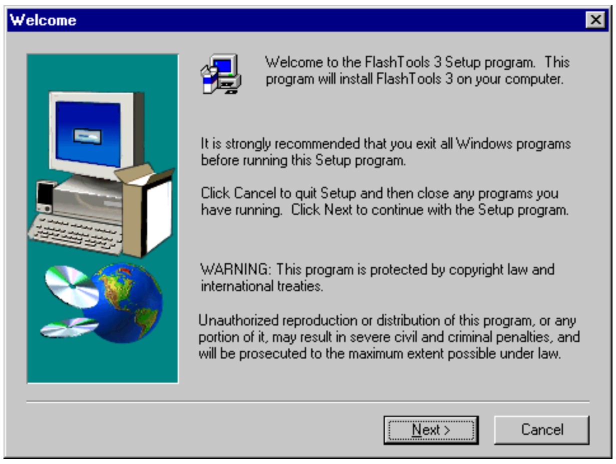
Figure 4: RESET and BOOT Buttons

This sequence of closing and opening the RESET and BOOT buttons sets the grabbMODUL-4 in „Bootstrap Mode“. This mode is required for the FlashTools to function correctly.

The grabbMODUL-4 should now be connected with a PC and a power supply via a „null modem“ cable. The module should also be in „Bootstrap Mode“ following the BOOT/RESET sequence. The grabbMODUL-4 will be described from this point on as „target hardware“.

2.4 Installation of the PHYTEC FlashTools and Downloading a Program Code

The current version of FlashTools is located on the PHYTEC Spectrum CD and is also available for download on the PHYTEC homepage. Start the setup program „***setup.exe***“ from within the folder ***.../Software/FlashTools3***.



Welcome

Welcome to the FlashTools 3 Setup program. This program will install FlashTools 3 on your computer.

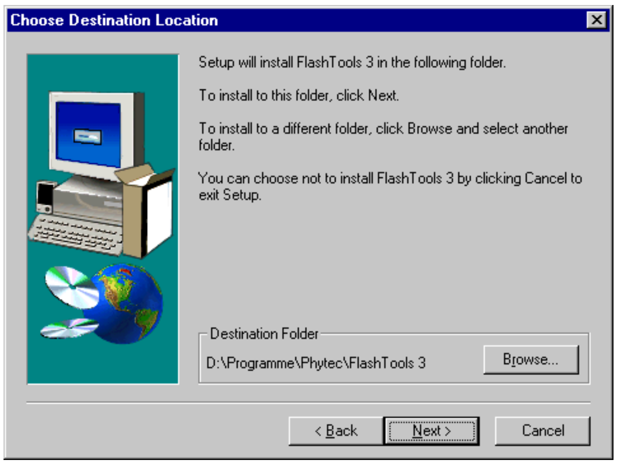
It is strongly recommended that you exit all Windows programs before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Follow the instructions for assigning the path and application names until installation has been carried out successfully.



Choose Destination Location

Setup will install FlashTools 3 in the following folder.

To install to this folder, click Next.

To install to a different folder, click Browse and select another folder.

You can choose not to install FlashTools 3 by clicking Cancel to exit Setup.

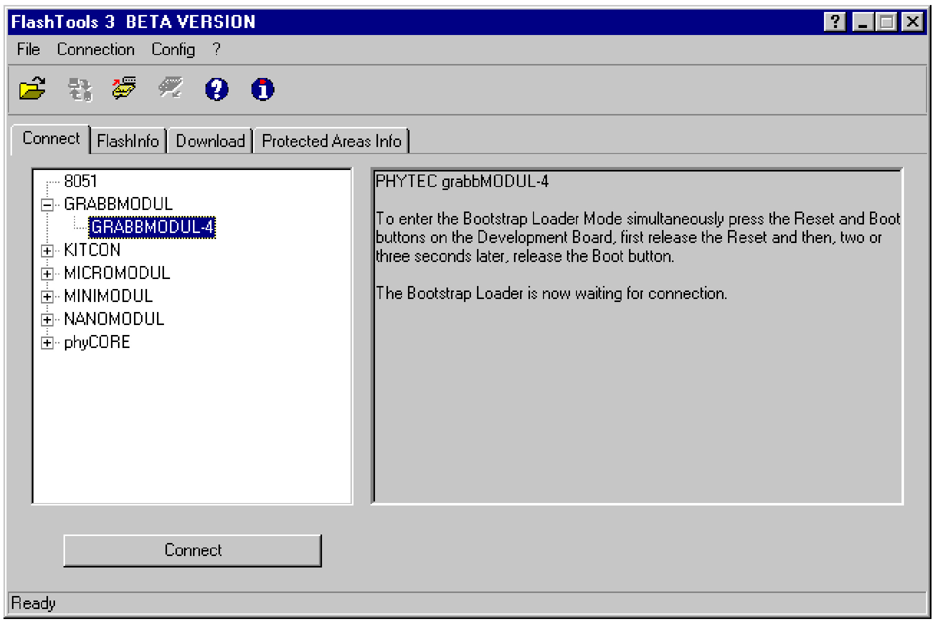
Destination Folder

D:\Programme\Phytec\FlashTools 3

After installation is complete, FlashTools can be started directly. It is not necessary for you to restart your host PC.

Upon initial installation it is likely that a request for a PHYTEC registration key will appear. If this is the case you can obtain this license key free of charge from PHYTEC Messtechnik GmbH by telephone or email.

- Start FlashTools for Windows by double clicking on the FlashTools symbol, or select FlashTools from the Programs/PHYTEC FlashTools program group.
- ***„Connect“*** will appear as an active tab sheet.



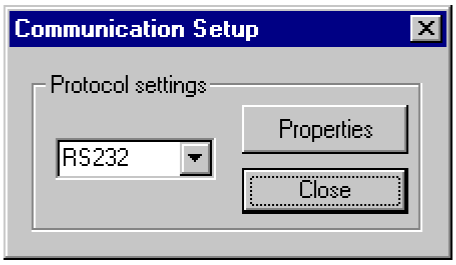
PHYTEC grabbMODUL-4

To enter the Bootstrap Loader Mode simultaneously press the Reset and Boot buttons on the Development Board, first release the Reset and then, two or three seconds later, release the Boot button.

The Bootstrap Loader is now waiting for connection.

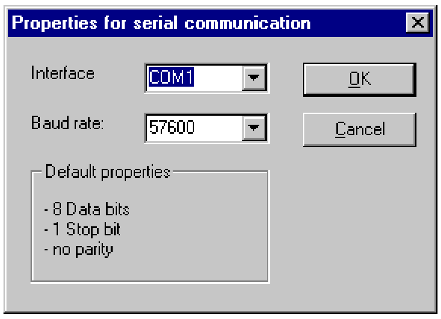
- Select the *„GRABBMODUL“* directory followed by the entry *„GRABBMODUL-4“* from the target hardware list.

- To choose the correct settings for your serial interface select the entry *„Protocol“* in the ***„Config“*** menu to start the *„Communication Setup“*.


Communication Setup

Protocol settings: RS232

- Select the entry *„RS232“* and then click on *„Properties“*.

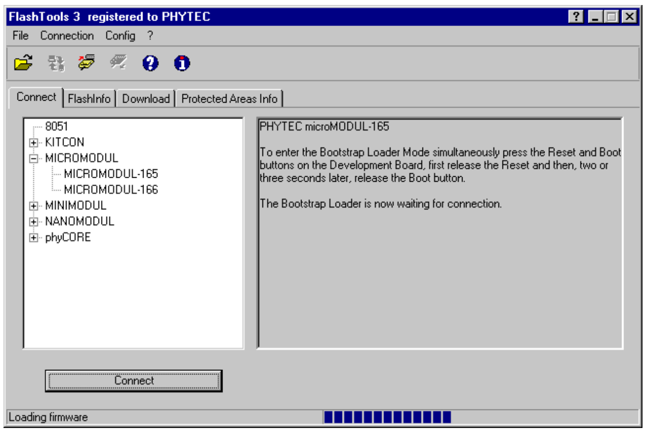

Properties for serial communication

Interface: COM1

Baud rate: 57600

Default properties
- 8 Data bits
- 1 Stop bit
- no parity

- Select the COM interface number (e.g. COM1) that you use on your PC and then select *57600* as your *„Baud rate“*. Confirm your selection by clicking on *„OK*“. The ***„Communication Setup“*** window by clicking on *„Close“*.


FlashTools 3 registered to PHYTEC

8051
KITCON
MICROMODUL
MICROMODUL-165
MICROMODUL-166
MINIMODUL
NANOMODUL
phyCORE

PHYTEC microMODUL-165

To enter the Bootstrap Loader Mode simultaneously press the Reset and Boot buttons on the Development Board, first release the Reset and then, two or three seconds later, release the Boot button.

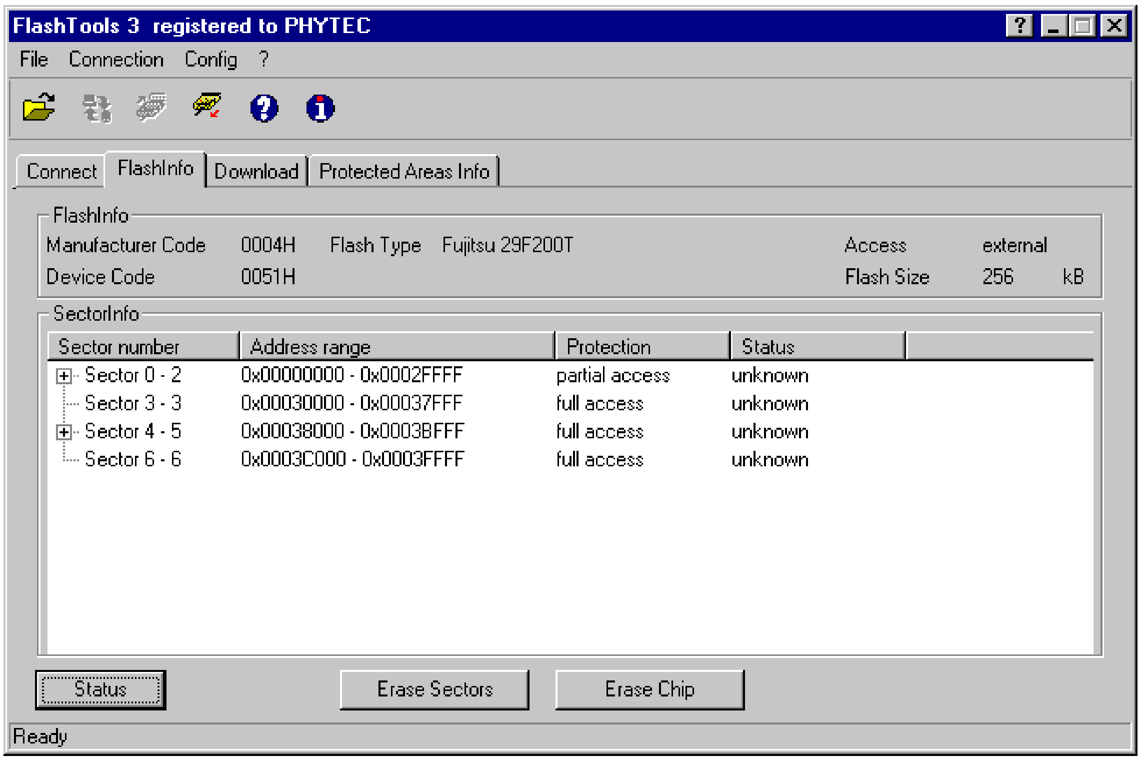
The Bootstrap Loader is now waiting for connection.

Loading firmware

- In the tab sheet ***„Connect“*** click on *Connect* in order to transfer the microcontroller-based portion of FlashTools to the target hardware.

The microcontroller is capable of communicating with the Baud rate set by the program due to an automatic Baud rate recognition. Should a problem with the set Baud rate occur, select a different Baud rate. Put the module in „Bootstrap Mode“ once again to test the new Baud rate.

- After the data has been transferred successfully, the tab sheet ***„FlashInfo“*** will become active in the FlashTools program. This displays the sectors and their address range in the Flash memory.



FlashInfo

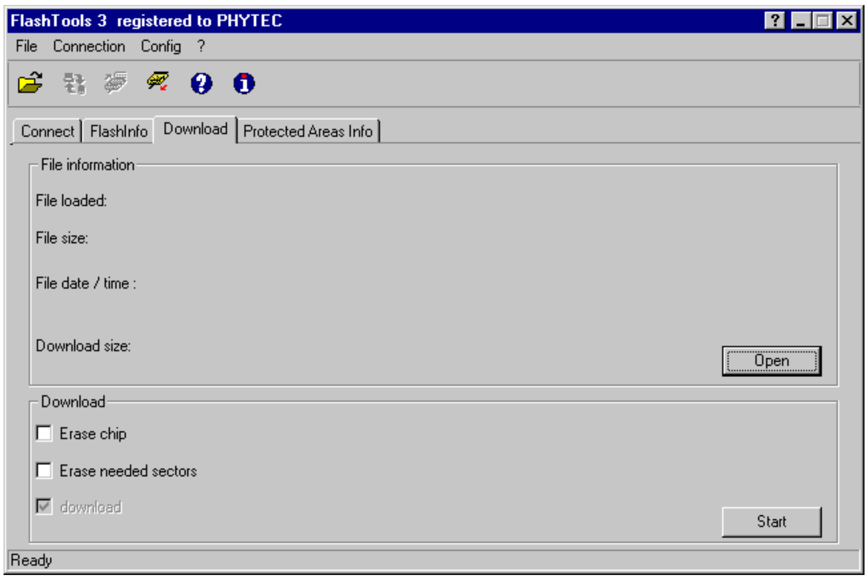
Manufacturer Code	0004H	Flash Type	Fujitsu 29F200T	Access	external
Device Code	0051H			Flash Size	256 kB

SectorInfo

Sector number	Address range	Protection	Status
Sector 0 - 2	0x00000000 - 0x0002FFFF	partial access	unknown
Sector 3 - 3	0x00030000 - 0x00037FFF	full access	unknown
Sector 4 - 5	0x00038000 - 0x0003BFFF	full access	unknown
Sector 6 - 6	0x0003C000 - 0x0003FFFF	full access	unknown

- Click select *„Erase Chip“* to erase all sectors of the Flash memory. Afterwards the status of all sectors should be displayed as „blank“.

- Select the tab sheet ***„Download“*** in order to select and download a Hex file to the Flash memory.



File information

File loaded:

File size:

File date / time :

Download size:

Download

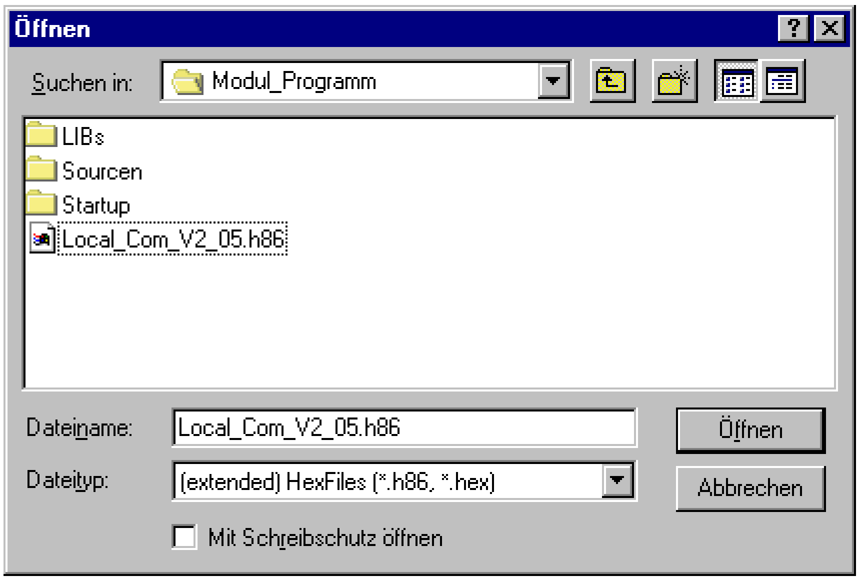
☐ Erase chip

☐ Erase needed sectors

☑ download

Ready

- Click on *„Open“* to select a Hex file for downloading.



Suchen in: Modul_Programm

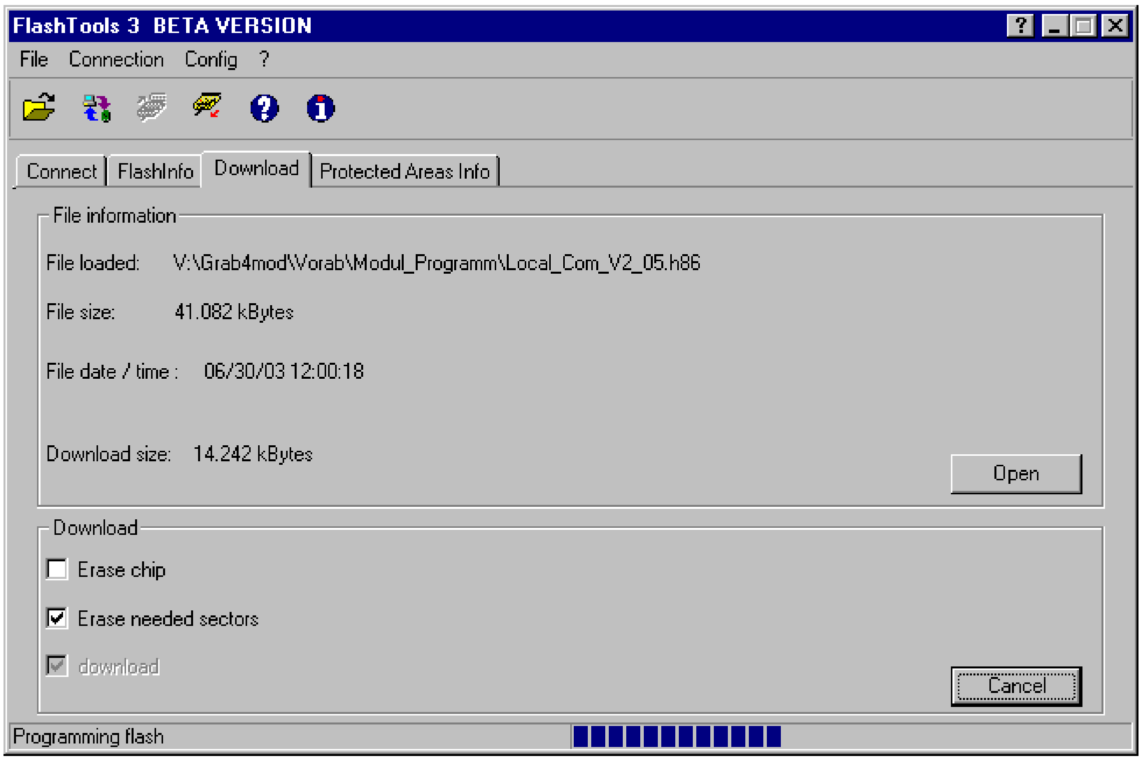
LIBs
Sourcen
Startup
Local_Com_V2_05.h86

Dateiname: Local_Com_V2_05.h86

Dateityp: (extended) HexFiles (*.h86, *.hex)

☐ Mit Schreibschutz öffnen

- Select the Hex file that was created for the grabbMODUL-4. For the first test select the PHYTEC firmeware „*Local_Com_Vx_y.h86* (e.g for version number 2.5 = *V2_05.h86*) from the CD for download.
- Click on *„Open“*.



File information

File loaded: V:\Grab4mod\Vorab\Modul_Programm\Local_Com_V2_05.h86

File size: 41.082 kBytes

File date / time : 06/30/03 12:00:18

Download size: 14.242 kBytes

Download

☐ Erase chip

☑ Erase needed sectors

☑ download

Programming flash

- Now click on the „*Start*“ button to begin downloading the Hex file. During the download the „*Start*“ button will change to „*Cancel*“ and can be used at this time to interrupt the procedure. The progress of the download is shown at the bottom right of the window as a progress bar.

- Now the program has been transferred to the Flash memory of the grabbMODUL-4. To break off the connection simply close the FlashTools program.

Hint:
The FlashTools program has to be closed to free the COM port on the PC allowing the COM interface can be used by other programs.

2.5 Testing the Module Program LOCAL_COM

The firmware LOCAL_COM („Local_Com_Vx.h86“) is a ready-made module software that allows you to control the grabbMODUL-4 from your PC and transfer (‘upload’) image data to the PC via the serial interface.

As part of the start up, the following sections describe the software counterpart on the PC. *A detailed description of the firmware, its function and applications are described in section 6.4, „Using PHYTEC Firmware“.*
The firmware LOCAL_COM can be tested with a Windows demo application (supplied with the grabbMODULe) or with a terminal program.

2.5.1 Power and Communication Connections

Connect the module to a PC and a power supply as described in *section 2.3, „Connecting the grabbMODUL-4 for „Download““.*

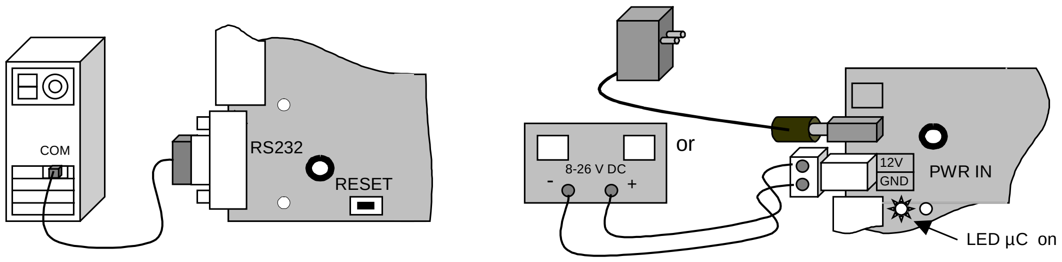


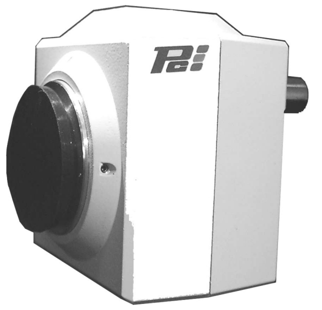
Figure 5: grabbMODUL-4 Power and PC Connections

Once the module is connected to the power supply it is ready for use. The green LED illuminates indicating its ready status. If you have performed a download previously, press the RESET button to restart the module.

2.5.2 Connecting the Camera

To digitize image data you will need an analog standard video camera with an appropriate lens. The following section describes the camera included with the RDK and its connection to the grabbMODUL-4. If you are using your own camera, be sure to adhere to the proper video signal norms.

The Camera



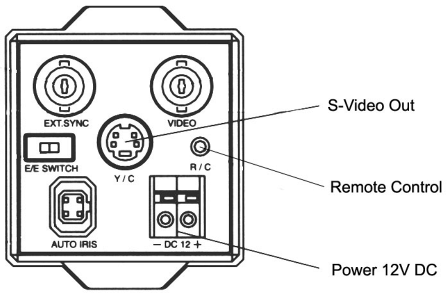


Figure 6: The Camera Included with the Kit and its Connection Field

Hint:
The camera is delivered without a lens. The black cap serves to protect the sensor chip and must be removed before a lens can be attached.

The connection field is located at the back of the camera (*Figure 6*). This is where the power supply and the video cable are connected. The round, black socket in the middle with the inscription Y/C is the S-Video socket (so-called Mini-DIN socket), with which the camera is connected to the grabbMODUL-4 via the S-Video cable.

Connect the camera and the grabbMODUL-4 using the S-Video cable (WKK051). Connect the camera's S-Video-Out (*see Figure 6*) with the S-Video Input (see Figure 1) on the grabbMODUL-4.
With cameras that have a composite signal output, use Composite Video Input 1 (BNC socket) on the grabbMODUL-4.

Now the power supply has to be connected to the camera via the power adapter SV009 (*Figure 6*). The camera's connection field has a four sided clamp pair with a red and black clamp and the inscription '– DC 12 +'. Hold down the small gray buttons in the middle of the clamps and feed the **black** cable end into the **black** clamp and the **red** cable end into the **red** clamp.
Release the buttons; the cable ends should be clamped securely. Double check to make sure the polarity is correct and pull lightly on the cables to make sure that they are connected securely.

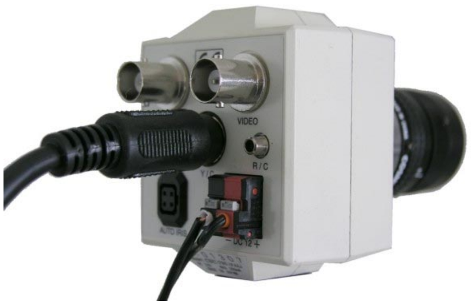


Figure 7: Camera Connector S-Video and Power Supply

The camera should now be connected as shown in *Figure 7*.

Hint:
Please note the polarity of the power supply.

The Lens and Focussing

The lens included with the Rapid Development Kit (*Figure 8*) has a 16 mm focal length and is made according to the C-mount standard.

Figure 8: The C-Mount Lens

The camera has a thread enabling connection of all lenses that have a C-mount or CS-mount screw connection. With C-mount lenses the 5 mm adapter ring (*see Figure 9*) included with the camera accessories must be used. With CS-mount lenses, the 5 mm adapter ring depicted in *Figure 9* can not be used.

Assembly

First remove the protective cap (*Figure 9*). Be sure not to touch the window in front of the sensor chip.

Hint:
Be sure to keep the cap. If the camera is transported without a lens, the cap should be re-attached to protect the sensor chip from finger prints, dust or any mechanical damage.

Take the 5 mm adapter ring and screw it onto the camera as shown in *Figure 9*.

Hint:
The threaded ring in the camera should not be adjusted. The fixing screws on the sides should not be removed.

Now attach the lens. The lens can be focused later when the camera and the grabbMODUL-4 are in use.

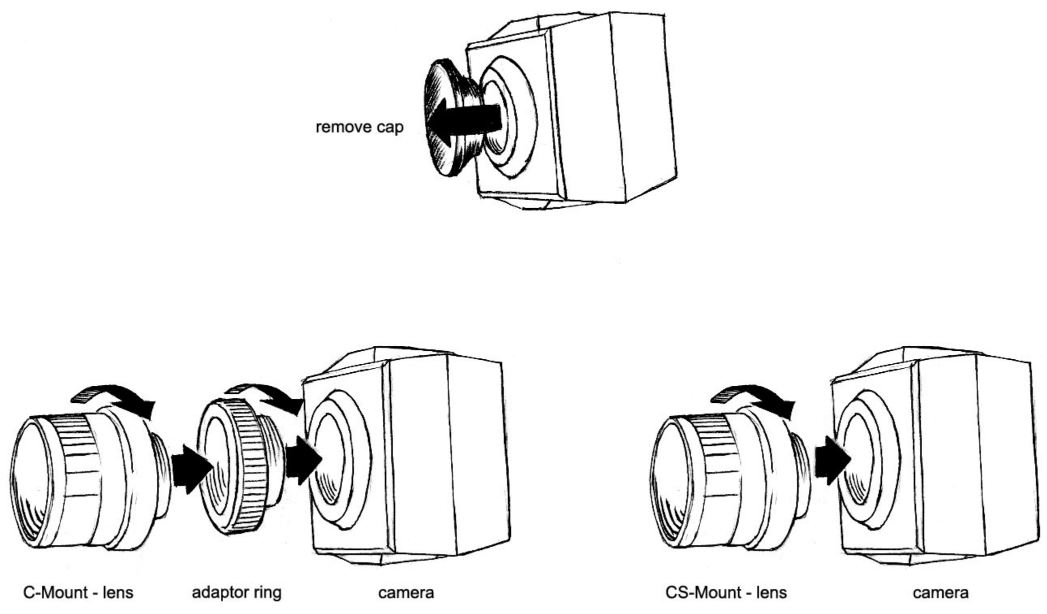


Figure 9: How to Mount the Lens on the Camera

Attaching the Camera to a Tripod

The included tripod (only with the RDK) serves to fix the camera stably in place. The tripod has a ball mount that can rotate in all directions and also be locked in place, allowing the camera to be put in any position (horizontal or vertical).

The three telescoping legs extend to various heights. If a large or heavy lens is used, the tripod's legs should be pulled out to prevent it from tipping over if the lens extends, shifting the center of gravity.

Loosen the locking screw on the top of the tripod and attach the camera as shown in *Figure 10*.

Turn the head of the tripod to screw the camera in place. The inscription on the camera's connection field must be upright (the BNC sockets must face up), in order to obtain an upright image later.

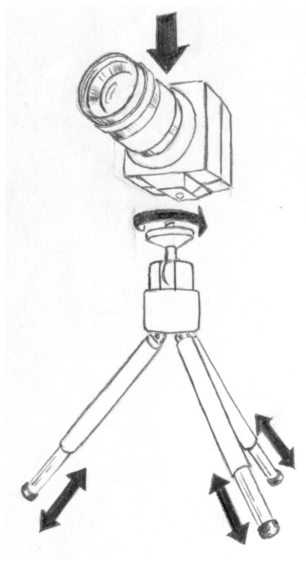


Figure 10: Attaching the Camera to a Tripod

Now you should have met all the hardware requirements for Start-Up (*see Figure 11*).

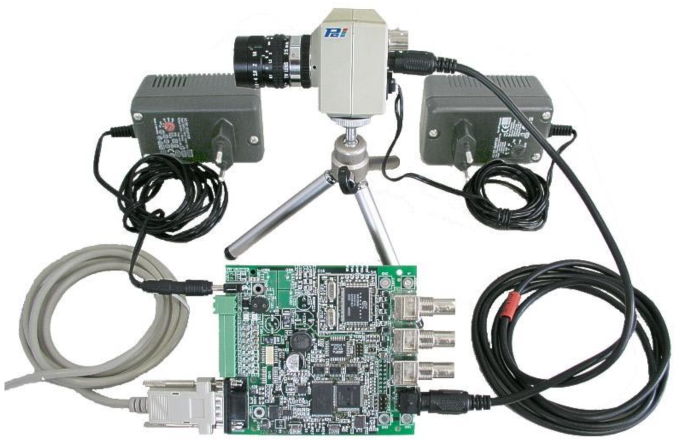


Figure 11: grabbMODUL-4 RDK Ready for Use

2.5.3 Testing the Module Program with the Windows Demo Program

The user can communicate with the firmware „LOCAL_COM“ on the grabbMODUL-4 over the RS-232 interface using the included PC demo program ***„PC_Vxx.exe“.*** It is thereby possible to configure and query settings as well as request or display image data. The software functions with the protocol described in s*ection 6.4.1*.
The PC software is compatible with Windows operating systems WIN98, ME, NT, 2000 and XP.

Hint:
The demo program can only demonstrate the preset functions of the „LOCAL_COM“-firmware. The considerably larger features and above all the „stand-alone“ capabilities of the grabbMODUL-4 can not be demonstrated with this program.

Starting and Configuring the Windows Demo Program

Start the PC Software *„**PC_Vxx.exe**“* (xx stands for the current version number). The program window „Com Image“ will open.

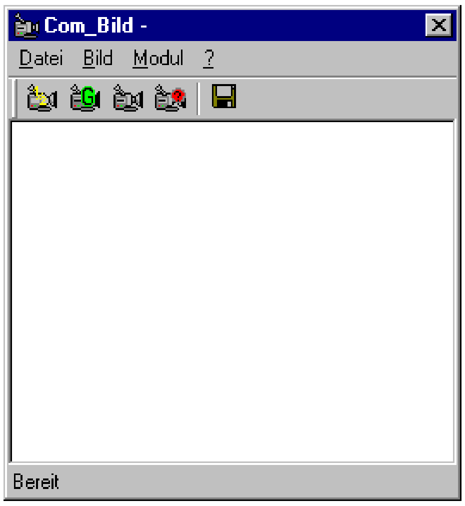
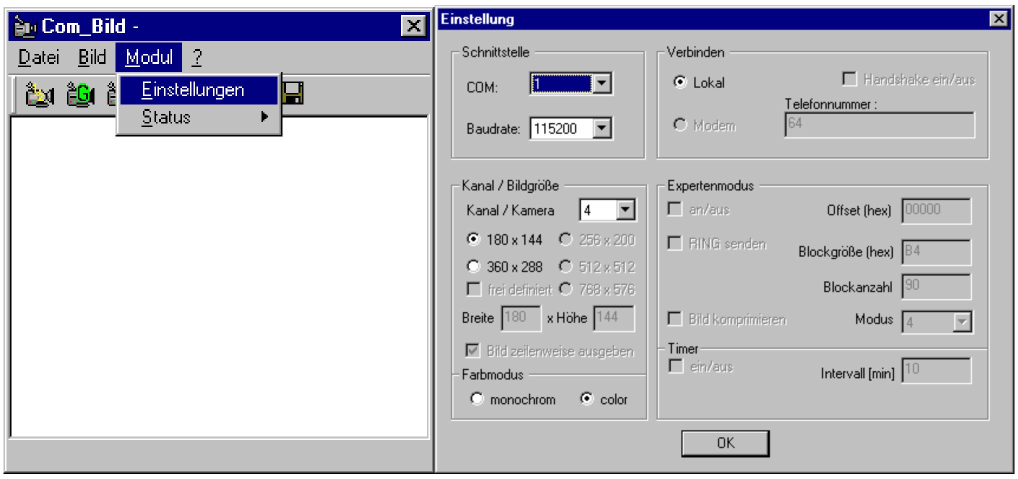


Figure 12: Windows Demo Program for the grabbMODUL-4

The individual icons of the PC program have the following functions:

Transmit image in 4-bit Nibble Mode (black&white)	Grab and store image on the module	Transmit image in 8-bit mode	Open the module status menu	Save image currently in the window

Select Module – Settings from the program:



Schnittstelle
COM: 1
Baudrate: 115200

Verbinden
(●) Lokal
( ) Modem
☐ Handshake ein/aus
Telefonnummer : 64

Kanal / Bildgröße
Kanal / Kamera 4
(●) 180 x 144 ( ) 256 x 200
( ) 360 x 288 ( ) 512 x 512
☐ frei definiert ( ) 768 x 576
Breite 180 x Höhe 144
☑ Bild zeilenweise ausgeben

Farbmodus
( ) monochrom (●) color

Expertenmodus
☐ an/aus
☐ RING senden
Offset (hex) 00000
Blockgröße (hex) 84
Blockanzahl 90
☐ Bild komprimieren
Modus 4

Timer
☐ ein/aus
Intervall [min] 10

Figure 13: Windows Demo Program: Menu Setting

In the Setting tab sheet (*see Figure 13*) please select:

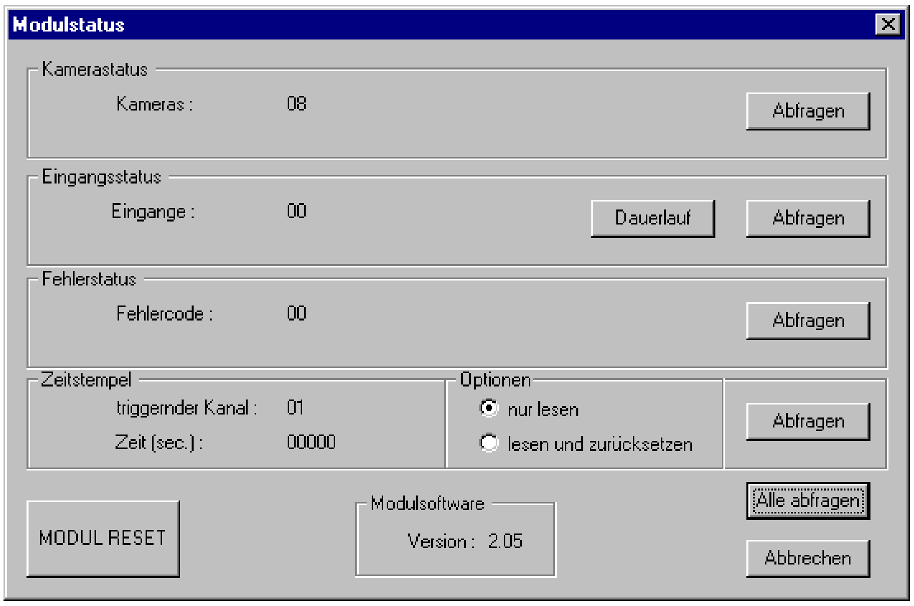
- the interface *(COM)* used on your PC and the *Baud rate 115200*
- Under *"Verbinden" (Connect)* the mode *"Lokal"* (*Local*) and *„Handshake:* ***aus****" (off).*
- Under *"Kanal / Figuregröße"* (*Channel/Image Size*) the *"Kanal / Kamera"* (*Channel/Camera)* „4" (which represents the S-Video input) and set the image size initially to 180 x 144 pixels.
- The desired color mode *„monochromatic"* or *„color"*.

Click on *<OK>* to close the setting window.

Hint:
If you have connected your camera to a channel other than S-Video, then you have to select this channel.
If a Baud rate other than 115200 Baud is to be used, then first the module software has to be modified. The module software does not have an automatic Baud rate recognition!

Testing the Communication and the Camera Signal

Click on the icon „Module Status“ and the „Module Status“ sheet will appear.


Modulstatus

Kamerastatus
Kameras : 08

Eingangsstatus
Eingange : 00

Fehlerstatus
Fehlercode : 00

Zeitstempel
triggernder Kanal : 01
Zeit (sec.) : 00000

Optionen
[✓ nur lesen]
lesen und zurücksetzen

Modulsoftware
Version : 2.05

Figure 14: Windows Demo Program: Menu Module Status

For Camera Status select *“Abfragen” (Query).* If the S-Video Camera is connected correctly, an „08“ will appear (binary coded, represents Channel 4, for a description *see 6.4.6, „Request Camera Status“*).

Possible Outputs:

- Value = 08: A camera is connected to channel 4. An image can be requested.
- The message *„Fehler V“* (*Error V*) appears: a connection with the grabbMODUL-4 could not be established. Check the settings, the connector cable and the power adapter.
- Value = 00: No camera signal was recognized. Check the video connection and the camera’s power supply.
- Value is <08: You have connected the camera to a channel other than S-Video. An image can be requested if the correct channel is selected from the the settings menu.
- Value is >08: You have connected more than one camera. An image can be requested if you select the correct channel in the settings menu.

Transferring Images

Now click on the icon „Grab a New Image“, in order to grab a current image from the camera.

The following will occur: The PC will send a control command to the module. The module software („Firmware“) will make the corresponding settings on the video grabber and start the image capture with the predefined image size and color format. The image will be stored in the module's memory.

Click on the icon „Transmit Image“, to retrieve and display an image.

The image will be read from the module's memory and transferred to the PC over the serial interface. The PC software calculates the color format for display and then shows the image line by line in the output window (*see Figure 15*).

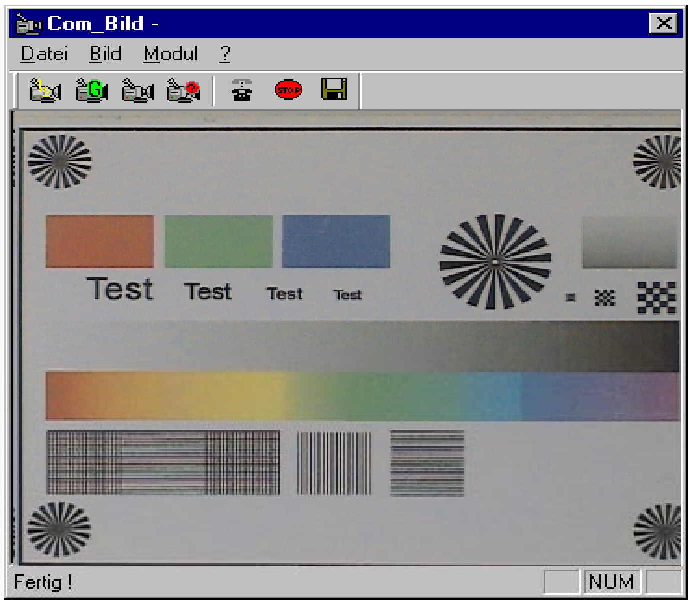


Figure 15: Windows Demo Program: Displaying the Image

The speed with which the image is displayed is limited by the transfer rate of the serial interface. Please note that the image is captured and available in the module considerably faster.

Possible Problems:

1) **An blurrd image appears:**

- Correct the focus at the lens. (Use the smallest black and white image in Loop mode (F7 key). Stop the Loop mode with the ESC key. **Please note**: **the image will be read completely**)

2) **No image is requested and the error message „Protocol Error“ (“*Protokollfehler*”) appears:**

- There is either no connection to the module or the wrong interface was selected.
- A Baud rate other than 115200 is being used.
- No null modem cable is being used.

3) **A slightly distorted image appears and the end of the error message „Protocol Error“ or „Time Out“:**

- The interface’s FIFO buffer has been set to high by the operating system (modification in ⇨ Control Panel ⇨Connections (COM_PORT) ⇨ Extended Settings ⇨ Use FIFO and set the receive buffer to „LOW“).
- There is interference in the serial connection.

4) **A black or blue image appears:**

- There is no camera connected on the selected channel.

5) **A distorted image appears with no color**

- An NTSC camera was connected instead of a PAL camera.

Additional Functions

Additional firmware functions can be tested with the software. These are described in *section 6.4.1* and are only mentioned here in brief.

By clicking on the icon „Transmit Image in Nibble ½ Mode" a black/white image will be retrieved and displayed in Nibble mode (4-bit).

By clicking on the icon „Save Image" the current image in the output window will be saved.

In the Module Status menu, information can be read about the intput/output status, error status, time stamp and version code. A module reset can also be carried out here.

2.5.4 Testing the Module Program with a Terminal Program

A terminal program can also be used to test the functions and the protocol. It is also possible to use the „HyperTerminal" application included with Windows. The control characters can be sent to the grabbMODUL-4 and the answers or raw image data can be displayed.

The following steps may differ slightly between operating systems.

- Start the *HyperTerminal* by clicking on **Start\Program\Accessories** (the file structure may vary from one operating system to another).
- The following *HyperTerminal* window will open:

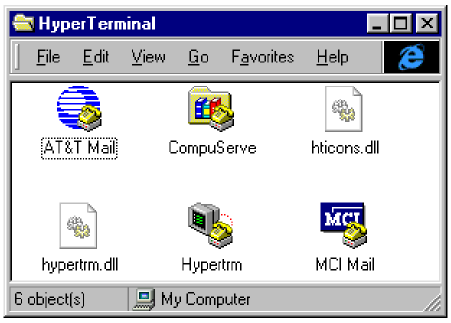
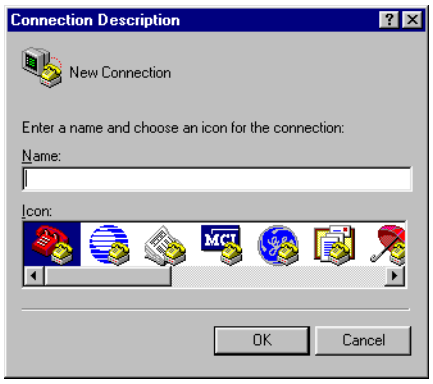


Figure 16: HyperTerminal: Creating a Program

- Double-clicking on the „HyperTerminal“ symbol will start a new HyperTerminal session.


Connection Description

New Connection

Enter a name and choose an icon for the connection:

Name:

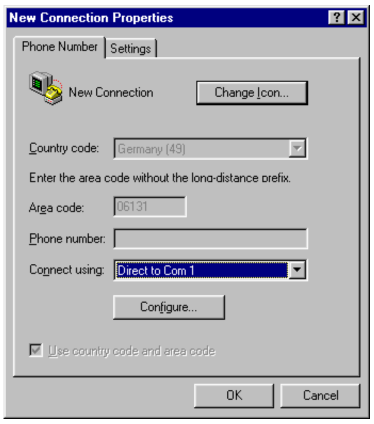
Icon:

Figure 17: HyperTerminal: Naming a Program

- The *Connection Description Window* will appear. Enter „*COM1 Connection*“ in the Name field (be sure to use the correct COM port on your PC).

- After you have confirmed these entries by clicking on „*OK*“ a new HyperTerminal connection with the name „COM1 Connection“ will be created.

- Specify the COM connection properties in the „COM1 Connection Properties“ window by clicking on the „*Configure*“ button.



New Connection Properties

Phone Number | Settings

New Connection

Country code: Germany (49)

Enter the area code without the long-distance prefix.

Area code: 06131

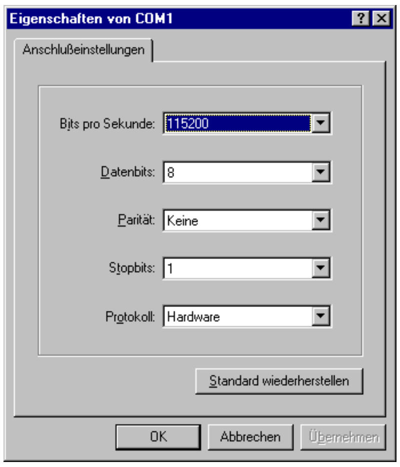
Phone number:

Connect using: Direct to Com 1

Use country code and area code

Figure 18: HyperTerminal: Program Properties

- Select the following COM parameters in the „COM1 Properties“ window:
  Bits per second = 115200
  Data bits = 8
  Parity = None
  Stop Bits = 1
  Protocol = None



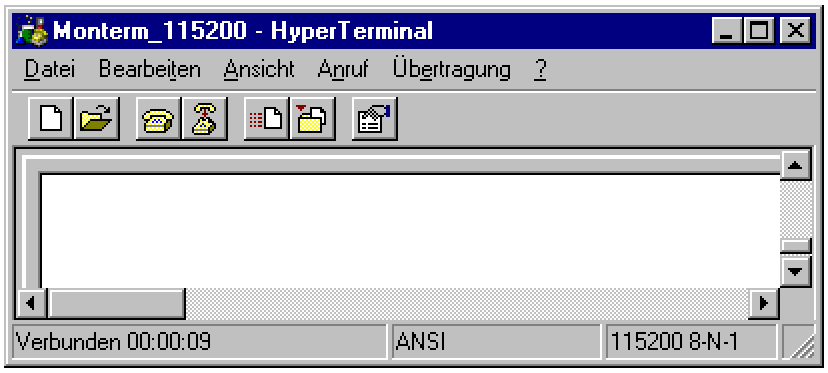
Bits pro Sekunde: 115200
Datenbits: 8
Parität: Keine
Stopbits: 1
Protokoll: Hardware

Figure 19: HyperTerminal: Configure Program

- Click on *„OK“* and switch to the monitor window *„COM1 Connection – HyperTerminal“.*

Close and restart HyperTerminal so that the settings entered are adopted.

- After starting the *„COM1 Connection – HyperTerminal“* you will see the monitor window. In addition status information about the connection will be shown by the bar at the bottom.

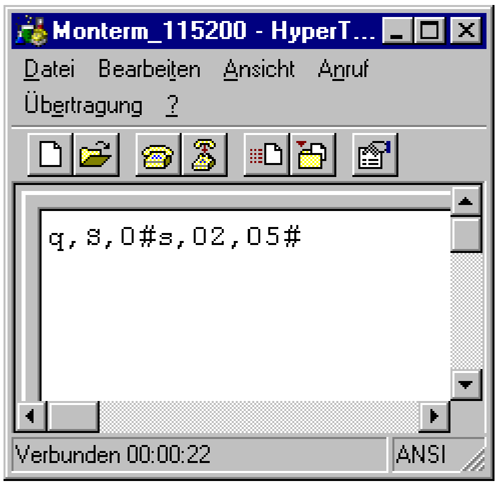


Verbunden 00:00:09 | ANSI | 115200 8-N-1

Figure 20: HyperTerminal: Start Program

- Perform a reset on the grabbMODUL-4, by pressing the *RESET* button on the module. This will start the ***Vx_y.h86*** program from the onboard FLASH memory.

- As an example we will test the function „Query Software ID“:
  Type in the following characters: `S#`
  Module answer: `q,S,0#s,02,05`



```
q,S,0#s,02,05#
```

Figure 21: HyperTerminal: Program Test

- If you desire an echo for the value entered, then activate the HyperTerminal’s echo function.
- It is possible to test all protocol functions in this manner.

Hint:
When testing the „I“ command (Image Data request) you have to send the confirm receipt (handshake) of each data block by entering „`q,i,0#`“ manually in the HyperTerminal.
To prevent the input from being interrupted by preset timeout times, it is recommended that you use the z-command („`z,30#`“) described in *section 6.4.16*, „*Setting the Interface’s Receive Timeout*“ to set the allowed delay time for the handshake to maximum.

- To end the connection, click on the „*End Connection*“ button in the HyperTerminal tool bar, or simply close the program.
  “Verbindung beeenden”(End Connection)

If you do not see any outputs in the HyperTerminal window, check the power supply, the COM port parameters and the RS-232 connection.

Part 2

Technical Data
Hardware Description

3 Technical Data

Dimensions:	100 x 110 mm (Printed Circuit Board) 100 x 140 mm with sockets (integration in Euro-housing possible)
Weight:	130 g (standard configuration VM-004)

Temperature Range:

Parameter	Condition	Min.	Typ.	Max.
Temperature	operating	0°C	-	70°C
	Storage	-40°C	-	90°C
Humidity (r.F.)	not condensed	-	-	95 %

Power supply: 8..28 V DC unregulated

Parameter	Condition	Min.	Typ.	Max.
Pwr Supply Voltage	-	8 V	-	28 V
	Abs. Maximum	-	-	30 V

Power consumption:

Parameter	Condition	Min.	Typ.	Max.
Power Consumption	operating	-	2 W	-
	Framegrabber Sleep-Mode	-	1.2 W	
	Grabber Sleep + CPU Idle	0.9 W		

Video Inputs:	(Model VM-004) 3 composite video inputs, 75Ω, $1V_{ss}$ 1 S-Video Input 75Ω ($0{,}7\ V_{ss}$ / $0{,}3V_{ss}$)
Video Format:	PAL (B,G,H,I), NTSC (M) or corresponding CCIR format monochromatic

Synchronization:	Composite-sync. or synchronized to Y-Signal Exernal synchronization not possible
Image Data Format:	16 Mio. Colors: YCrCb 4:2:2 256 Gray levels: Y8
Image resolution:	max. 768 x 576 Pixel (PAL) or 640 x 480 Pixel (NTSC) resolution is freely scalable in X- and Y direction to 14:1
Image Aquisition:	field: 20 ms frame: 40 ms Image transfer to the controller memory in real time without using controller resources
Image Correction:	Gamma correction Brightness (+/- 50 %) Contrast (0%...235 %) Color saturation (U: 0...201 %, V: 0...283 %) Hue (+/- 90°, NTSC only)
Bit map memory:	1 MB SRAM (524.288 pixels) Sequential storage of 1st and 2nd field
Processing Kernel:	16-bit Infineon C165
Clock Frequency:	25 MHz
Memory Configuration:	RAM: 256 kByte, optional to 1 MB, additional Bit map memory FLASH: 256 kByte, optional to 1 MB EEPROM: 1024 Byte, serial

Realtime Clock: optional

Ports : 4 optically isolated signal inputs

Parameter	Condition	Min.	Typ.	Max.
Input Low Voltage	-	0 V	-	5.5 V
Input High Voltage	-	6.5 V	-	24 V
Input Voltage	Absolute Max.	-	-	26 V
Input High Current	V_{IH}=26 V	-	-	7 mA
Reverse Voltage	Absolute Max.	-	-	-21.5 V
Reverse Current	V_{IR}=-21.5 V	-	-	7 mA
Switch Freq. (symm. Rectangle)	V_{IH}=6.5 V	40 Hz	-	-
	V_{IH}>7.0 V	-	90 Hz	-

4 optically isolated signal outputs
open collector, shared emitter

Parameter	Condition	Min.	Typ.	Max.
CE-Breakdown Volt.	Ic=0,5mA	80 V	-	-
EC-Breakdown Volt.	I_E=0,1 mA	7 V	-	-
Collector Dark Current	V_{CE}=48V	-	0.01uA	0.1uA
	V_{ce}= 48V, T_a=85°C	-	2 uA	50 uA
Capacitance (C-E)	V=0, f=1MHz	-	10 pF	-
C-E Saturation Volt.	switched ON	-	0.3 V	-
Off-State Collector Current	V_{CE}= 48V	-	-	15 uA
max. Power P_c max	1 x ON	-	-	100mW
Collector Current	Rc=1k, Uc=10V	-	5 mA	-
ΔP_T/°C	Maximum Rating	-	-	-1.7 mW/°C

1 I²C interface (Master, software emulated)

4 TTL-I/O lines (controller ports)

Parameter	Condition	Min.	Typ.	Max.
Input Low Voltage		-0.5 V	-	0.9 V
Input High Voltage		1.9 V	-	5.5 V
Output Low Voltage	I_{OL}=1.6 mA	-	-	0.45 V
Output High Voltage	I_{OH}=-250 uA	4.5 V	-	-
	I_{OH}=-1.6 mA	2.4 V	-	-

3.1 Connectivity

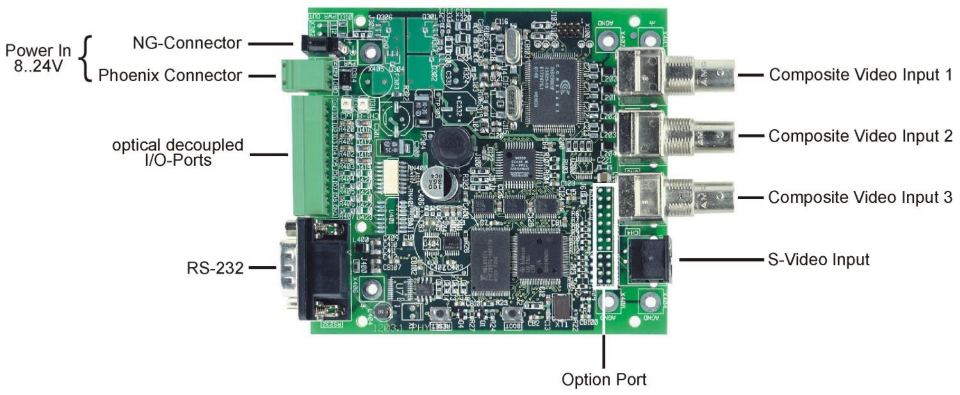


Figure 22: Overview of Connectivity

3.1.1 Power supply

The grabbMODUL-4 can be supplied with power either via the coaxial power adapter plug („NG-plug“) or the two-pronged Phoenix socket (*see Figure 22*).

The allowable input voltage range is 8...28V DC. An unregulated DC power adapter will suffice as a power source. The grabbMODUL-4’s power consumption is approximately 2 Watts. The power supply input of the grabbMODUL-4 is protected against reverse polarity.

Appropriate NG-plug diameter: 1.1 mm

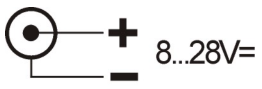


Figure 23: Polarity NG-Socket (X300)

The Phoenix plug offers the possibility of connecting the current supply via a screw clamp connector. The corresponding plug can be purchased from PHYTEC under order number GP088.

The connector layout corresponds to the print on the PCB:

Power supply X301	
Pin	**Function**
12V	+8...28 V Power in
GND	Power Ground

Table 1: Power supply Plug

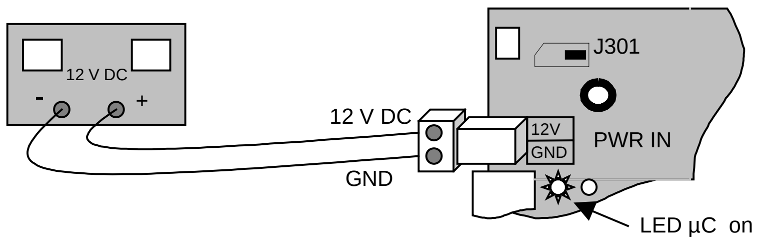


Figure 24: Connecting the Power Supply

If the power supply is connected properly the green LED uC will illuminate.

Hint:
The power supply is secured to the underside of the module via fuse F302 with 1A. Should LED uC not illuminate despite a proper supply voltage connection, then the securing device may be defective. In this case the module will have to be inspected by PHYTEC.

The plug fuse F303 is not required for normal operation of the grabbMODUL-4 and is therefore not populated in the standard delivery state.

Caution!
The power supply is protected against voltage spikes by transil diode. If the power supply exceeds the value 28 V the board or the power supply can be damaged. If necessary an additional external fuse should be connected.

Upon connection of the operating voltage the module will begin execution of the firmware programmed in the FLASH. An additional manual reset is not required. However, it is important to make sure that the power supply increases steadily when switched on to ensure a correct internal module reset.

If necessary the grabbMODUL-4 can be switched on by a control signal. The *shutdown* connector TP304 is used for this purpose. By connecting TP304 to Ground shuts off the module. TP304 is left *open* to switch on the module.

Hint:
TP304 can not be connected with Vcc or any other voltage. The pin can only be connected to GND. If necessary a Schottky diode (e.g. BAT42) should be connected in order to prevent any current flow into the pin.

3.1.2 Serial Interface

The module's serial interface extends outward to the 9-pin Sub-D plug RS232 (P400). The socket's wiring corresponds to that of a PC (DTE-device):

Serial RS-232 Interface P400	
Pin	**Function**
1	N.C.
2	RxD (Received Data)
3	TxD (Transmitted Data)
4	DTR (Data Terminal Ready)
5	GND (Signal Ground)
6	DSR (Data Set Ready)
7	RTS (Request To Send)
8	CTS (Clear To Send)
9	N.C.

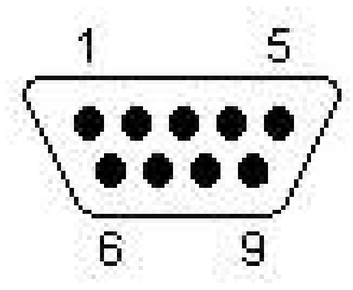


Table 2: Pinout of the Serial Interface

Hint:
The interface wiring corresponds to a PC's wiring. This means that data devices, such as a modem, can be connected directly via a normal serial cable.
Devices such as a PC that are wired like DTE-devices, have to be connected via a null modem cable (order code WK041). A null modem cable crosses the RxD and TxD signals.
The DSR and DTR signals are connected to one another on the module. If this is not desired then Jumper J400 has to be left open.

Descriptions of the Remaining Controller Signals

Primarily only the signals TxD, RxD and GND are required for a serial data transfer. This „three-wire connection“ is sufficient for most applications if the connection is synchronized with control characters.
In applications where no control characters can be sent (e.g. with a binary modem connection), the synchronization occurs over the serial interface's handshake signals.

There are two signal pairs here:

DSR/DTR: These signals ensure the basic ready status for sending and receiving between send and receive devices. With the grabbMODUL-4, DSR and DTR signals are connected with a jumper internally. Ready status for sending or receiving is always signalled this way, which suffices in most cases. If the DSR/DTR connection is not desired, J400 can be removed.

RTS/CTS: These signals indicate ready status during transmission. The start of a transmission is signaled to an attached device (e.g. a modem) by the module by activation of the RTS-signal. The device responds by indicating its own ready status by activating the CTS-signal.
During transmission, the signals can be deactivated by either device, which will result in a temporary interruption of the transmission (e.g. if the receive buffer is full). *More details about this protocol are available in descriptions of the RS-232 Interface.*

Hint:
As opposed to UART devices in PCs, there is no automatic control of the C165 controller's handshake signals, since the controller does not have the corresponding FIFO buffers. If necessary, the user will have to implement these buffers and corresponding interface control via software (e.g. as an interrupt routine).
Such a function can be found on the driver CD.

The signal lines RTS/CTS are available on the following controller port pins:

Signal	Controller-Port	Direction	Description
RTS	P3.13	OUT	RS-232: Ready To Send
CTS	P3.15	IN	RS-232: Clear To Send

Table 3: Signal Allocation of the RS-232 Handshake Signals

3.1.3 Video Inputs

Two types of video sources can be connected to the grabbMODUL-4:

(a) Composite Video Sources

With composite signals, all image signals are carried on a single line. Typical connector types are BNC- or cinch plugs. Up to three composite video sources can be connected to the grabbMODUL-4 via the BNC sockets CH1 through CH3.
Pinout:

BNC Video Inputs	
Pin	**Function**
Shield	GND (Signal Ground)
Pin	Video Input (Composite)

Table 4: BNC Video Input Wiring

Color as well as black and white cameras can be connected to these inputs.

(b) S-Video Sources

The advantage of the S-Video connector is the separation of brightness and color signals. This prohibits Moiré interference of fine image structures and improves the actual resolution of color images.

An S-Video source can be connected to the 4-pin Mini-DIN socket CH4.
Pinout:

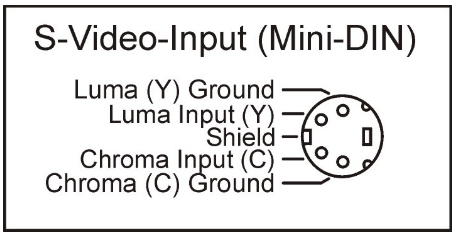
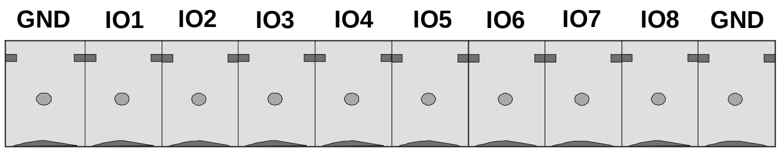


Figure 25: Pinout of the S-Video Socket

Hint:
Software is used to switch between the input channels. Please note that the correct mode (Composite/S-Video) has to be selected.

3.1.4 Optically Isolated I/O Ports

Control signals can be exchanged with peripheral devices over the optically isolated I/O ports. There are four inputs and four outputs available:



Optically Isolated I/O (X400)	
Pin	**Function**
GND	I/O-Ground (– / Emitter)
IO1	INPUT 0 (+)
IO2	INPUT 1 (+)
IO3	INPUT 2 (+)
IO4	INPUT 3 (+)
IO5	OUTPUT 0 (Collector)
IO6	OUTPUT 1 (Collector)
IO7	OUTPUT 2 (Collector)
IO8	OUTPUT 3 (Collector)
GND	I/O-Ground (– / Emitter)

Figure 26: I/O-Port Pinout

Hint:
The Ground signals of all the I/O channels are connected. The I/O Ground signals are galvanically separated from the Ground of the grabbMODUL-4.

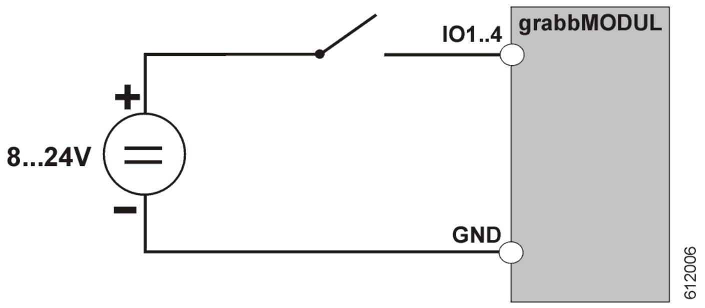


Figure 27: Typical Input Wiring of the I/O-Port

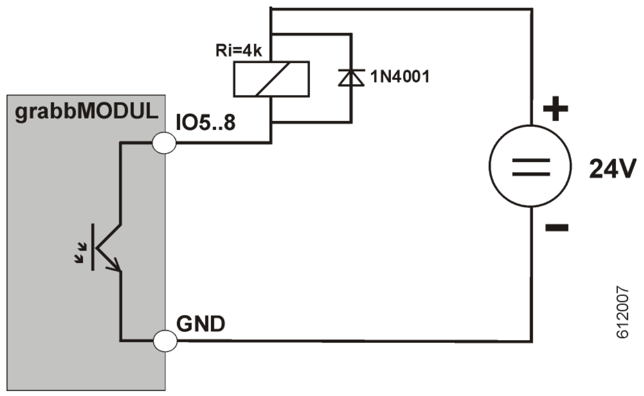


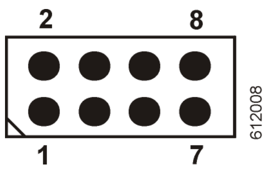
Figure 28: Typical Output Wiring of the I/O-Port

Figure 27 and Figure 28 show typical wiring of the I/O-port.

Caution!
When wiring the ports, be sure that the polarity is correct and that the maximum allowed current and voltage values *(section 3)* are not exceeded. Otherwise the module may be damaged.

3.1.5 Option Port

The Option port OPT-PORT, X401 is an 8-pin header connector, on which some of the controller's additional control signals are available. This port is well suited for connecting user peripherals to the grabbMODUL-4.



Option Port (X401)	
Pin	**Function**
1	Vcc (+5 V)
2	GND (Signal Ground)
3	SCL (I²C-Bus Clock Line)
4	SDA (I²C-Bus Data Line)
5	P6.4 / CS4 (Controller Port 6.4)
6	P6.5 (Controller Port 6.5)
7	P6.6 (Controller Port 6.6)
8	P6.7 (Controller Port 6.7)

Table 5: Signals of the Option Port Connector X401

The SCL and SDA signals are the integrated I²C interface's clock and data signals.
External devices that have an I²C interface in Slave mode can be queried or controlled over the I²C interface. Multiple I²C devices can be connected to the Bus, but they must all have differing device addresses.

The grabbMODUL-4's I²C interface is an I²C Bus Master implemented by software. It is therefore possible to operate Slave devices on this Bus. The Bus is managed by the Master.
To control the devices, use the I²C routines in the included driver library.

Hint:
The I²C interface operates with a TTL level. Therefore the maximum signal length is limited. No signal lines longer than 30 cm should be used.

Caution!
I²C devices are already connected to the module's internal I²C interfaces (EEPROM and optional RTC). Make sure when connecting external I²C devices that there is no address conflict with the internal I²C devices. Otherwise the devices will not function correctly.
In some cases it is possible that the module's firmware will not start correctly.

The following I²C address ranges cannot be occupied by an external device:

Internally Occupied I²C Address Ranges		
Device	**Option**	**Range**
EEPROM (40C08, **default**)	-	A8$_{HEX}$...AF $_{HEX}$
EEPROM (40C04)	J4 = 1+2	AC$_{HEX}$...AF$_{HEX}$
	J4 = 2+3	A8$_{HEX}$...AB $_{HEX}$
RTC	Optional	A2$_{HEX}$...A3 $_{HEX}$
Video Processor	-	80 $_{HEX}$...8C $_{HEX}$

Hint:
Additional information about the I²C interface is available on the internet, for example at the Philips Semiconductor homepage.

The controller ports P6.4...P6.7 are freely available to the user. You can connect devices here that are TTL compatible (for precise specifications, *please refer to section 3, „Technical Data*"). The controller ports can be addressed directly in your program. This requires no particular software driver.
Pleas note that the controller pins are defined as inputs by default. To use them as outputs, the relevant pin has to be configured accordingly.

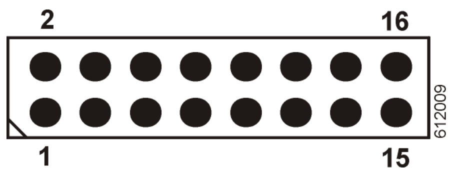
Hint:
Controller port pins can not carry strong currents. Especially those currents that flow from the pin (if the pin is switched to Vcc), cannot be strong, otherwise the voltage connected to the pin will fail. *Please refer to the portpin specifications in this regard*.

The signals P6.4..P6.7 are also available on the uC interface X402. Make sure that the signal lines have not been wired multiple times by mistake. As an alternative the signal P6.4 can also be used as Chip-Select signal /CS4. Make sure when connecting P6.4 that there will be no conflicts should the signal be used as a Chip-Select signal.

3.1.6 Extended Video Port (Optional)

Hint:
This connector is only available for special configurations of the grabbMODUL-4. The connector is not available in the standard variant.

With some variants of the grabbMODUL-4 it is possible to connect four additional video inputs to pin header EXT-PORT X204.



External Video Port (X204)	
Pin	**Function**
1	Vcc (+5 V)
2	PE GND (Shield Ground)
3	V_{OUT} (camera supply, optional)
4	GND (Signal Ground)
5	GND (Signal Ground)
6	Composite Video Input 5
7	GND (Signal Ground)
8	Composite Video Input 6
9	GND (Signal Ground)
10	Composite Video Input 7
11	GND (Signal Ground)
12	Composite Video Input 8
13	V_{OUT} (camera supply, optional)
14	GND (Signal Ground)
15	Vcc (+5 V)
16	PE GND (Shield Ground)

Table 6: Pinout of the External Video Port (X204)

The channels can be switched using software *(see Driver Description).*

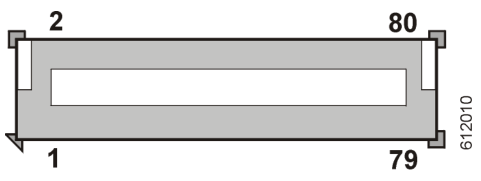
Caution!
An expansion board (e.g. VM-004-EXT-BNC) is required to connect the additional inputs. Video sources can **not** be fed directly to the pin header.

The shield (PE-GND) has a capacitive coupling with the power Ground and has an additional galvanic connection to the through-holes X403 and X406. If necessary a leading connection to the housing can be created here.

3.1.7 Microcontroller-Port (Optional)

Hint:
This connector is only available for special configurations of the grabbMODUL-4. The connector is not available in the standard variant.

The microcontroller port is an expansion interface on which all important microcontroller signals are available. It can be used to connect complex hardware.
The interface is in the form of a Molex connector (2 x 40-pin, 0.653 mm pitch). The corresponding mating connector can be purchased through PHYTEC under the part number VB091 (approx. 5 mm profile) or VB084 (approx 10 mm profile).



Microcontroller Port (X402)			
Pin	**Dir.**	**Name**	**Function**
1	PWR	Vcc	+5 V Supply Output
2		Vcc	
3	-	GND	Signal Ground
4	I/O	/RESET	/Reset (Initiate)
5	I/O	IN7/P2.9	Microcontroller Port P2.9
6		IN6/P2.8	Microcontroller Port P2.8
7	IN	/NMI	Not Maskable Interrupt Input
8	-	GND	
9	I/O	P6.4/CS4	uC-Port P6.4 = /CS4
10	OUT	/CS3	Chip Select 3 (peripheral CS)
11	OUT	ALE	Address Latch Enable *)
12	OUT	/RESOUT	/Reset (Output)
13	-	GND	
14	OUT	/RD	Memory Read
15	OUT	/WRL	Memory Write / Write Low Byte
16	OUT	A0	Microcontroller Address A0 *)
17		A1	Microcontroller Address Bus
18	-	GND	
19	OUT	A2	Microcontroller Address Bus
20		A3	
21		A4	
22		A5	
23	-	GND	
24	OUT	A6	Microcontroller Address Bus
25		A7	
26		A8	
27		A9	
28	-	GND	
29	OUT	A10	Microcontroller Address Bus
30		A11	
31		A12	
32		A13	
33	-	GND	
34	OUT	A14	Microcontroller Address Bus
35		A15	
36	I/O	D0	Microcontroller Data Bus
37		D1	
38	-	GND	

(continued on next page)

39	I/O	D2	Microcontroller Data Bus
40		D3	
41		D4	
42		D5	
43	-	GND	
44	I/O	D6	Microcontroller Data Bus
45		D7	
46	OUT	A16	Microcontroller Address Bus
47		A17	
48	-	GND	
49	OUT	A18	Microcontroller Address Bus
50		A19	
51		A20	
52		A21	
53	-	GND	
54	OUT	A22	Microcontroller Address Bus
55		A23	
56		D8	Microcontroller Data Bus
57		D9	
58	-	GND	
59	I/O	D10	Microcontroller Data Bus
60		D11	
61		D12	
62		D13	
63	-	GND	
64	I/O	D14	Microcontroller Data Bus
65		D15	
66	OUT	/WRH	Memory Write (High Byte)
67	IN	/RDY	Microcontroller /Memory Ready
68	-	GND	
69	IN	BOOT	Activate Bootstrap Loader ²)
70	I/O	P6.5	Microcontroller Port 6.5
71		P6.6	Microcontroller Port 6.6
72		P6.7	Microcontroller Port 6.7
73	-	GND	
74	OUT	/ACTIVE	Framegrabber Active Status
75	I/O	SDA	Serial I²C Bus: Data Line
76	OUT	SCL	Serial I²C Bus: Clock Line
77	I/O	CTRL1	reserved (P2.14)
78	-	GND	
79	PWR	Vcc	+5V Supply Output
80		Vcc	
*) used in multiplexed bus mode only ²) must not be connected to GND, might only be tied to VCC			

Table 7: Pinout of the Microcontroller Expansion Bus X402

3.1.8 RAM Memory Backup Device

Hint:
This connector is only available for special configurations of the grabbMODUL-4. The connector is not available in the standard variant.

On the grabbMODUL-4 it is possible to buffer the internal SRAM (main memory and video Bit map memory) with a battery in the event of a power failure. To do this a module is required that is already equipped with the battery controller U9.

Hint:
This memory buffering option is dependent upon the type and size of RAM populated on the board. Large and fast RAMs require so much current even in standby mode that a battery would quickly be depleted.
Should you require a battery buffered RAM for your application, be sure to request a module variant with a memory configuration that will allow for this.
Up to 512 Byte (optional: 1kByte) of user data can also be stored as non-volatile memory in the module’s EEPROM.
The backup battery is also required if the module is equipped with a Real Time Clock (RTC).

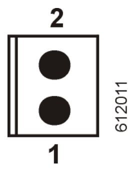
There are two possibilities for buffering the RAM:

(a) Connection of a lithium battery

> A lithium battery of type CR2032 (PHYTEC part number BL003) can be connected at BAT1 (circular symbol on the top side of the module).
> This battery is also required if the Real-Time Clock (RTC) is connected.
> The life time of the battery depends on the capacity and memory configuration on the module.
> The battery’s charge can be checked with the help of the battery voltage monitor.

(b) Connection of an external buffer condenser

A power failure bypass is also possible with a buffer condenser for a brief time. In this case it is best to use a high-capacitive condenser (Gold Cap).
This buffer condenser can be connected to X1:



Buffer Condenser (X1)	
Pin	**Function**
1	VPD (Condenser +)
2	GND

Table 8: Buffer Condenser Pinout

Caution!
Since the condenser is charged and discharged over this connector, it is important that a corresponding charging circuit will be attached.
If there is a direct connection, a high current will flow to the condenser as soon as the supply voltage is turned on, which could result in damage to the module. Therefore be sure to connect a corresponding charging circuit (*Figure 29* shows a simple wiring example).

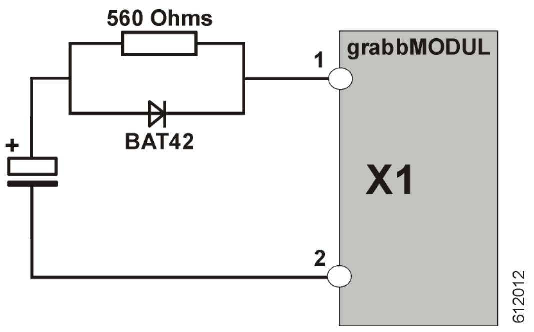


Figure 29: Simple External Wiring for Buffer Condenser

Hint:
X1 can also be used to connect a rechargeable battery, whereby it is important that the correct charging circuit is used.

The RTC is always powered by a battery and/or a buffer condenser. For activation of the RAM memory buffer, the module has to have the following configuration:

Pos.	Description	Setting
J1	Battery Backup Enable 1+2 = Fast RAM, Battery Backup Disabled 2+3 = Normal RAM, Battery Backup Enabled	2+3
R20	Battery Backup Jumper 0 R = Fast RAM, Battery Backup Disabled open = Normal RAM, Battery Backup Enabled	open

Table 9: Battery Backup Configuration

(These settings are required for both battery and condenser buffering).

4 Block Diagram

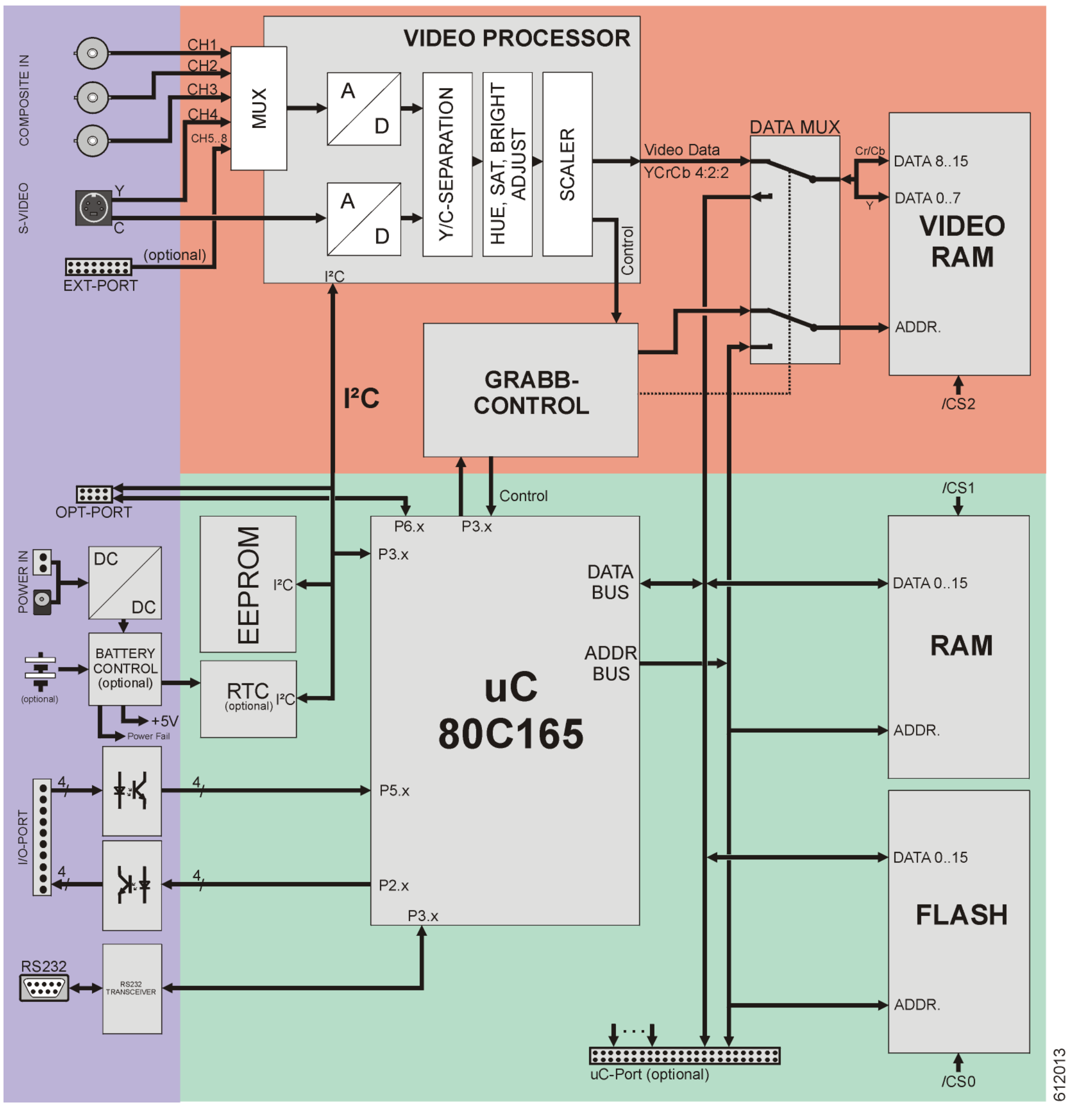


Figure 30: Block Diagram VM-004

Processing Kernel

The processing kernel of the grabbMODUL-4 consists of a C165 microcontroller with Flash program memory and SRAM data memory. The memory operates in 16-bit non-multiplexed mode. The grabbMODUL-4 is also popluated with an EEPROM for storage of non-volatile data.

The grabbMODUL-4 can be populated with an optional Real-Time Clock, which can be addressed over the I²C interface just like the EEPROM. The important controller signals extend outward to the optional pin header row „uC-Port“ for expansion circuitry.

Framegrabber

The independently functioning Framegrabber portion can be accessed via the Video-RAM. This RAM is connected with either the video processor or the microcontroller via a data multiplexer. Switching between the two is performed automatically by the Framegrabber controls.

During a Grabb procedure (image capture), the Video-RAM is masked out of the controller memory storage area so that the video processor can write the image data into this storage area. Afterwards the Video-RAM is enabled again in the controller memory map automatically.

The image capture is carried out by the Grabb Control integrated in the Framegrabber portion, without consumption of controller resources. The controller starts the procedure with a control signal. Then it takes no longer than 80 ms to capture a frame. In this time the controller can continue to function normally. Only the Bit map memory (Video-RAM) is not available, as previously stated.
The Grabb Control registers the conclusion of the recording procedure with the controller by sending a status signal. After this time the controller can access the Video-RAM again and read image data (write accesses to the image data memory by the controller are not possible).

Up to three composite and S-Video sources can be connected to the grabbMODUL-4.
As an option the grabbMODUL-4 can also be populated with an 8-channel multiplexer. This makes eight (composite-) video inputs available.

The video inputs extend to a multiplexer. The desired input channel can be selected with software. It takes less than 300 ms to switch between two channels.

The video processor digitizes the video signal. With S-Video signals, the digitization of the color signal is carried out by a second A/D converter, which achieves best color reproduction and image quality.

The digital image data is subsequently filtered and corrected. Here the user has the possibility of influencing brightness, contrast or color saturation. The *Scaler* determines the image resolution and desired screen window. Therefore the exact amount of data required for an application can be taken from the image data without consuming controller resources.
The image data is always stored in the Video-RAM in YCrCb format. This format only requies 2 Bytes of memory per pixel for full color reproduction (16 mio. Colors). This format is good for transferring color images, since it reduces transfer bandwidth. Another advantage presents itself for image analysis, since the brightness (Y) and color (CrCb) portions are already seperated, which is not the case in RGB format.

Periphery

Frequently needed peripheral components are integrated on the grabbMODUL-4. The switching power supply enables the module to be powered from an unregulated direct current with a wide input voltage range of 8...28 V. This range covers typical voltages used in industrial and automotive applications. As an option there is also the possibility of buffering the RAM from a backup battery.
There are 4 input lines and 4 output lines (optically isolated) available for any switching signals.
A fully configured RS-232 interface (with Hardware-Handshake) can be used to communicate with a host PC, a modem or another peripheral device. Additional devices can be connected to the I²C interface and controller port lines via the expansion port.

5 Jumper and Resource Layout

5.1 Jumper Plan

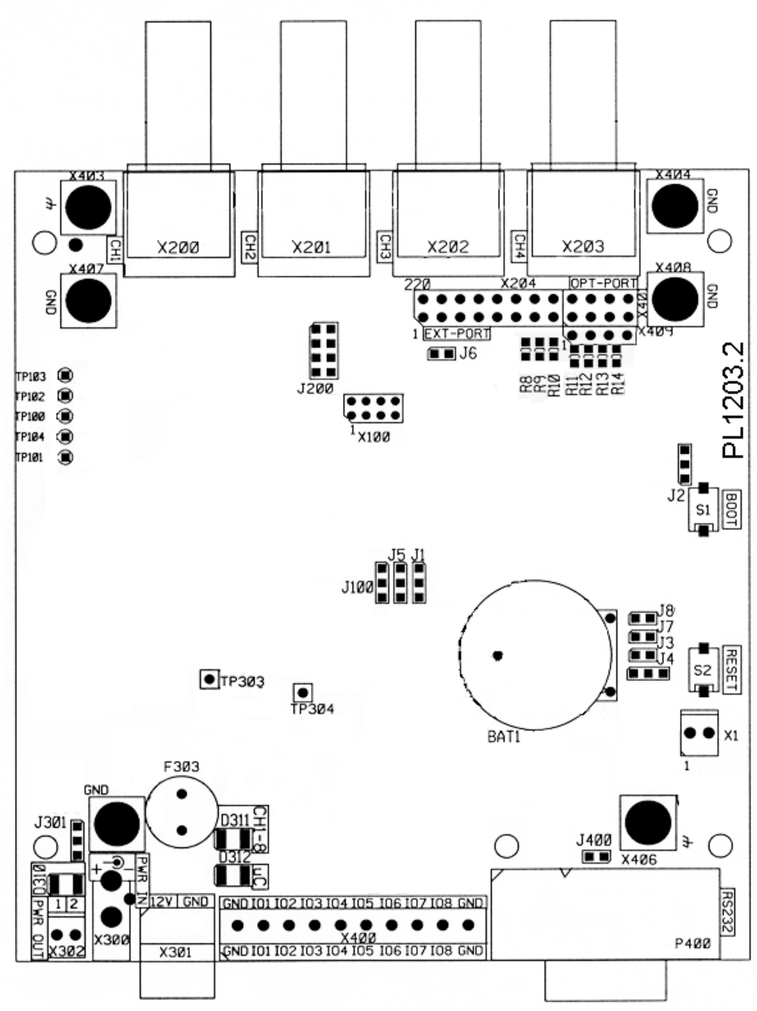


Figure 31: Jumper Positions (Top View)

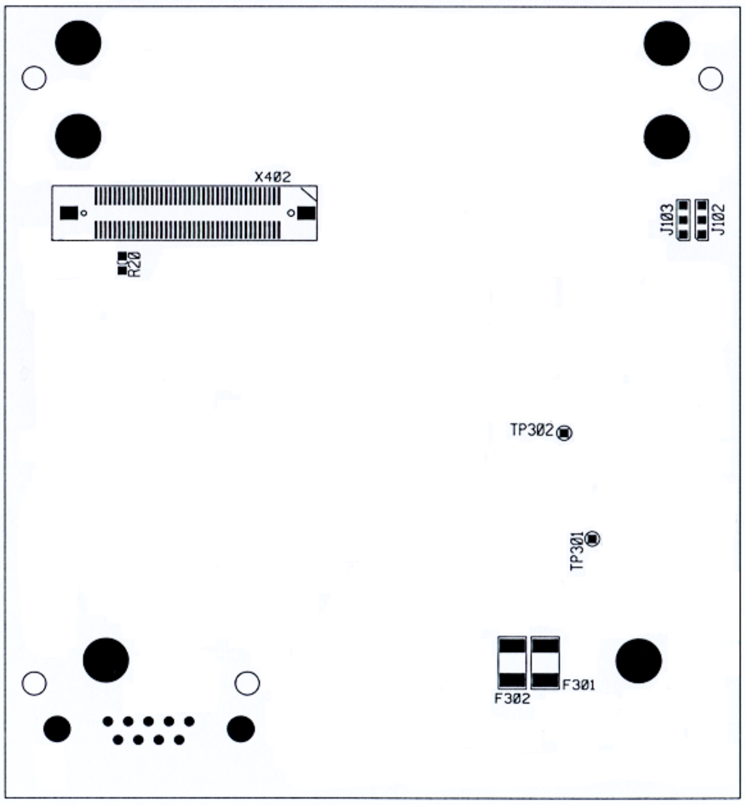


Figure 32: Jumper Positions (Bottom View)

5.2 Jumpers and Options

Fuses

Pos.	Description	Value
F301	not populated	-
F302	Module supply	1A T
F303	not populated	-

Power Supply Shutdown

Pos.	Description	Standard
TP304	Open: Module in operation GND: Shutdown (power supply switched off)	open

The module can be switched off over connector TP304. It is important that the pin is not wired to Vcc or any other operating voltage (*see section 3.1.1*).

RAM Size

Pos.	Description	Default
J5	1+2 = 1 MByte main memory (2 x 512 kB SRAM) 2+3 = 256 kByte main memory (2 x 128 kB SRAM)	2+3

The grabbMODUL-4 can have various factory-installed memory sizes. J5 has corresponding factory settings. Do not alter the position of J5.

RAM Battery Backup

Pos.	Description	Default
J1	Battery Backup Enable 1+2 = Fast RAM, Battery Backup Disabled 2+3 = Normal RAM, Battery Backup Enabled	1+2
R20	Battery Backup Jumper 0 R = Fast RAM, Battery Backup Disabled open = Normal RAM, Battery Backup Enabled	0R

If required the grabbMODUL-4 can be equipped with a battery buffered RAM. This requires a special module version that supports battery buffering (*refer to section 3.1.8*).

EEPROM Bus Address

Pos.	Description	Default
J4	40C08: No meaning (default configuration) 40C04: 1+2 = I²C-Bus Address EEPROM = AC_{HEX} 2+3 = I²C-Bus Address EEPROM = $A8_{HEX}$	2+3

The jumper determines the base address of the serial EEPROM on the I²C-Bus.

EEPROM Write-Protect

Pos.	Description	Default
J3	**24C01 / 24C02** open (no options)	

	24C04/24C08
	Open = no write protection
	Closed = write protection
		open

By closing J3, the contents of the EEPROM can be write protected. If the jumper is closed, no more write accesses to the EEPROM are possible.
This option does not exist for all installable EEPROMs. Write protection can be activated for the standard EEPROM.

RTC Interrupt

Pos.	Description	Default
J7	open = no interrupt possibilty for the RTC closed = RTC interrupts possible via hardware	open

If the RTC is installed on the grabbMODUL-4, it is possible to connect the interrupt output of the RTC to the controller portpin P3.2/CAPIN.

Power-Fail – Interrupt

Pos.	Description	Default

J8	open = no Power-Fail-Interrupt
	closed = Power-Fail-Interrupt possible
		open

Pos.	Description	Default

R30
R31
R32
	Voltage monitoring of backup battery
	R30 = R31 = 10 MΩ
	R32 = open
		R32=open
		R30=10M
		R31=10M

	Operating voltage monitoring:
	R30 = 100k
	R31 = open
	R32 = $\left(\frac{U\min}{1{,}25V}-1\right)\cdot 100k\Omega$	

With an installed battery controller (option), if the operating voltage or the backup battery voltage falls below a certain value, a non-maskable interrupt (NMI) may be generated.
If the backup battery's voltage is being monitored, an NMI will be generated if the voltage threshold falls below 2.5 V.
If the operating voltage is to be monitored, the user can set the switching threshold themselves by selecting the resistor value of R32 (*see formula*).

If the switching threshold U_{min} is set approximately 15 % below the rating voltage, then the following values for R32 result for typical operating voltages:

U_B	U_B-15%	R32	U_{MIN}
10V	8,5 V	560k	8,25 V
12V	10,2 V	750k	10,6 V
15V	12,75 V	910k	12,63 V
20V	17 V	1,2M	16,25 V
24V	20,4 V	1,5M	20,0 V

(The deviating voltage values for U_{min} are the result of rounding to standard row values.)
The resistor value should not be smaller than 560 kΩ.

Controller Configuration

The microcontroller is configured over configuration resistors, which are attached to specific data lines. Most settings have a direct relation to the hardware architecture and should therefore not be modified by the user!

Pos.	Description				Default
R8 R9	CHIPSEL Configuration				4k7,4k7
	R8	R9	external Chip Select		
	open	open	/CS3,/CS4 active		
	4k7	4k7	/CS3, /CS4 disabled (default)		
R10 R11	Configuration Segment Address Lines				4k7,4k7
	R11	R10	Address Area		
	open	open	256 kByte		
	4k7	4k7	1 MB (default)		
	open	4k7	16MB		
R12 R13 R14	Clock Mode				4k7,4k7,4k7
	R14	R13	R12	CPU-Clock (C165)	
	open	don't care	don't care	f_{OSZ} : 2	
	4k7	don't care	don't care	f_{OSZ} (default)	

RS-232 Handshake

Pos.	Description	Default

J400	open = DTR / DSR not connected
	closed = DTR / DSR connected (modem operation)
		open

For binary data transmission over the serial interface, the Hardware Handshake signals are required. This is the case when data is transmitted over a modem.
The RS-232 interface has two pairs of Handshake signals. The pair DTR/DSR signals the basic ready state of the device for transmission. The grabbMODUL-4 cannot control or evaluate these signal pairs. Therefore these signals can be jumpered with J400, whereby the principle transmission ready state is always signaled. Communication control occurs over signal pair RTS/CTS, which is available at P3.13/P3.15 (*see section 3.1.2*).

5.3 Resources

This section gives an overview of the allocation and organization of controller resources.

Hint:
Signals that have an entry in the „Assignment“ column of the table below, have fixed assignments on the module. These signals may also be externally available. The function of these signals, however, cannot be changed (e.g. by reprogramming the port function), since this could affect the module’s function. Please note the data flow direction (IN/OUT) of these signals, should you choose to connect external circuitry.

Signals without entries in the „Assignment“ column can be used freely by the user. The „Available“ column shows which connector pin the signal extends to (header - pin).

Signals that are identified as „Reserved Control Signal“ can not be used by the user.

Modifications to the predefined port functions could destroy the module!

Chip Select Signals

Signal	Assignment	Available	Description
/CS0	FLASH	-	Chip Select Flash Memory
/CS1	RAM	-	Chip Select RAM Memory
/CS2	VIDEO-RAM	-	Chip Select Video RAM Memory
/CS3	-	X402-10	Chip-Select 3
/CS4	-	X402-9	Chip-Select 4 (shared with P6.4)

Controller Ports

Signal	Assignment	Available	Description
P0.x	DATA	-	Data Bus
P1.x	ADDRESS	-	Address Bus
P4.0 - P4.3	ADDRESS	-	Address Bus (Bank Select)
P4.4 P4.5 P4.6 P4.7	ADDRESS / PORT	X402-51 X402-52 X402-54 X402-55	Address Bus (Bank Select) or free port pin (independent of memory configuration)
P6.0 P6.1 P6.2	/CS0 /CS1 /CS2	-	Internal Chip Select Signals
P6.3 P6.4	-	X402-10 X402-09 ²	Free /CS3 or free port Free /CS4 or free port
P6.5 P6.6 P6.7	-	X402-70 ² X402-71 ² X402-72 ²	Free port
P5.x	OPTICAL IN	-	Optically isolated inputs
P2.0 P2.1	-	X402-05 X402-06	Free port
P2.2 - P2.13	OPTICAL OUT	-	Optically isolated outputs (X400)
P2.14	CTRL1	X402-77	Reserved control signals
P2.15	-	-	N.C.
P3.0	/ACTIVE	X402-74	Framegrabber active signal (Output)
P3.1	MUX_A0	-	Reserved control signal
P3.2	RTC-IRQ*	-	Interrupt signal Real Time Clock
P3.3	SDA	X402-75 ²	I²C interface data signal
P3.4	SCL	X402-76 ²	I²C interface clock signal
P3.5	/START	-	Reserved control signal
P3.6	INIT	-	Reserved control signal
P3.7	FRAMECAP	-	Reserved control signal
P3.8	INSTANT	-	Reserved control signal
P3.9	-	-	N.C.
P3.10	TXD0_TTL	X409 ³	RS-232 interface TxD, TTL level
P3.11	RXD0_TTL	X409 ³	RS-232 interface RxD, TTL level
P3.12	/WRH	X402-66	Controller Write-High-Signal
P3.13	RTS_TTL	X409 ³	RS-232 interface RTS, TTL level
P3.15	CTS_TTL	X409 ³	RS-232 interface CTS, TTL level

*) dependent on module configuration
²) also available at X401
³) these signals can oly be used if the RS-232 transceiver U5 is not populated

Additional Controller Signals

Signal	Assignment	Available	Description
/NMI	POWER FAIL*	X402-07	Not Maskable Interrupt
/RD	/READ	X402-14	Controller Read-Signal
/WRL	/WRL	X402-15	Controller Write-Low-Signal
/READY	-	X402-67	Memory timing delay request
ALE	-	X402-11	Address-Latch Enable
/RESET	/RESET	X402-04	Reset Initiate
/RESOUT	/RESOUT	X402-12	Reset Output

*) dependent on module configuration

Reserved I²C Address Space

Internally Occupied I²C Address Areas		
Device	**Option**	**Range**
EEPROM		A8$_{HEX}$...AF $_{HEX}$
RTC	Optional	A2$_{HEX}$...A3 $_{HEX}$
Video Processor	-	80 $_{HEX}$...8C $_{HEX}$

5.4 Fields of Application and Safety Instructions

Please adhere to the specified operating conditions when using the grabbMODUL-4. Read this section carefully prior to starting up the module.

- The grabbMODUL-4 is used to digitize video signals from standard TV cameras and to process this data. Signals from composite video cameras can be processed, as long as they correspond to the standards CCIR B,G,H,I and the subnorms CCIR B,G,H,I/PAL. Alternatively, signals corresponding to CCIR M/NTSC can be processed. The camera signals can also be sent separately in Luma and Chroma portions according to the S-Video standard.

- The grabbMODUL-4 is designed to be a system component. When designing a system, be sure take the technical data of the grabbMODUL-4 into consideration in addition to the appropriate standards and safety guidelines relevant to the application in question.

- The device is designed for implementation in a clean and dry operating environment. For most applications it will be necessary to give the grabbMODUL-4 a protective housing, to be sure that operating conditions can be maintained. It is also important to check whether additional measures need to be taken for adherence to radio interference norms as well as to ensure operating safety.

- For commercial applications, you must follow all rules defined by the government safety organisation or the corresponding authority for your country / the area where the system is to be used.

- Prior to start up it is important to make sure that the device is suitable for the application and environment in question. If there is any doubt, you must inquire with a technical expert or with the manufacturer.

- The product must be protected against strong vibration. If necessary, a suspension device or shock absorber of some kind should be used that does not hinder the device's ventilation.

Any repairs that may become necessary should only be undertaken by a technical expert using original parts. When connecting the device only permitted and tested connector cables should be used. Be sure that the cable is properly shielded and free of interference.

5.5 References for CE-Conformance and Interference Protection (Concerning Europe)

A note about EMC laws for the grabbMODUL-4

When building a device or system that contains a grabbMODUL-4, make sure that all CE-conformance and interference protection requirements are met.

Since microcontroller systems function with high clock frequencies, there is a risk that they will generate electromagnetic fields and possible interference. This can be counteracted by a carefully designed system.

The grabbMODUL-4 has been tested successfully in specific configurations for adherence to the CE norms DIN EN 55024 and DIN EN 55022.

Caution!
Please note that if you connect addtional components or systems to the grabbMODUL-4, certain module properties relevant to certification may be affected.
It is therefore the responsibility of the system integrator to make sure that the entire device or entire system conforms to the necessary norms.

The grabbMODUL-4 offers various grounding possibilities for adaptation to different operation situations. A portion of the mounting holes are connected to Ground (GND). The module’s GND can be connected to the housing or PE with metallic dowel pins or via cable connections. The Ground potentials can be isolated with plastic dowel pins. This is a good idea if you want to avoid Ground loops.

The Shield-GND connectors X403 and X406, located next to the video sockets and the RS-232 socket, establish a connection to the shielded connector of the corresponding sockets. If necessary they can be used to establish a shield connection to the housing.

Upon examination and measurement of the basic configuration VM-004 by PHYTEC, no additional shielding measures were found to be necessary.

Caution!
Please be aware that strong interference spikes on the video signals or on the cable shield can destroy the input levels of the grabbMODUL-4.
Therefore, additional measures to protect against interference have to be taken in industrial environments with high levels of interference, as well as when longer cables are used.
If long video cables are used or if the image processing components are integrated in machines or systems, potential equalization currents can occur, which must be kept away from the grabbMODUL-4 inputs by appropriate devices.
PHYTEC takes no responsibility for damages that result from improper connection of the signal sources.

Part 3

Programming Manual

6 Driver Software

6.1 Basics

In order for the grabbMODUL-4 to be implemented in your application, appropriate user software (*firmware*) must be present in the module.
You can create this firmware yourself or you can obtain an existing firmware, which is designed for a specific task or range of tasks.
Another possibility is to have PHYTEC create firmware specially designed for your application.
There are different steps and development tools required for an application, depending on how you have chosen to proceed:

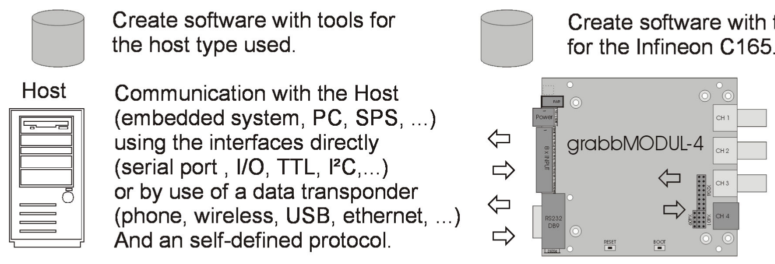
- **Creating your own software for the grabbMODUL-4**

  This allows you the greatest degree of flexibility when creating the application software. You can create your own program for the C165 processing kernel of the grabbMODUL-4.
  To do this you will need expertise in programming this microcontroller and a corresponding development environment (e.g. from Keil). Programming is usually carried out in C.
  The firmware program you've created can be programmed into the grabbMODUL-4's Flash memory with the help of PHYTEC FlashTools, which is included with the module.

  If your application requires communication with another device (e.g. remote image transferrence), then a program must also be created for this device. When transferring an image to a PC, the program would be a rendering and communication software on the PC. This software is created using standard PC programming tools.

Option 1) Creating your own software for the grabbMODUL-4

a) Using a Host for Communication

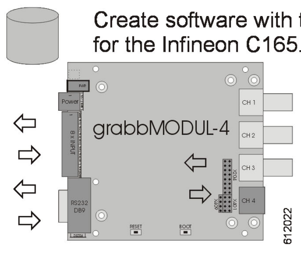


Create software with tools for the host type used.

Communication with the Host (embedded system, PC, SPS, ...) using the interfaces directly (serial port , I/O, TTL, I²C,...) or by use of a data transponder (phone, wireless, USB, ethernet, ...) And an self-defined protocol.

Create software with tools for the Infineon C165.

a) Using the grabbMODUL as a “stand alone”-device



Create software with tools for the Infineon C165.

Direct control of applications or units via the interfaces of the grabbMODUL-4 vorhandene Schnittstellen (serial, I/O, I²C, ...).

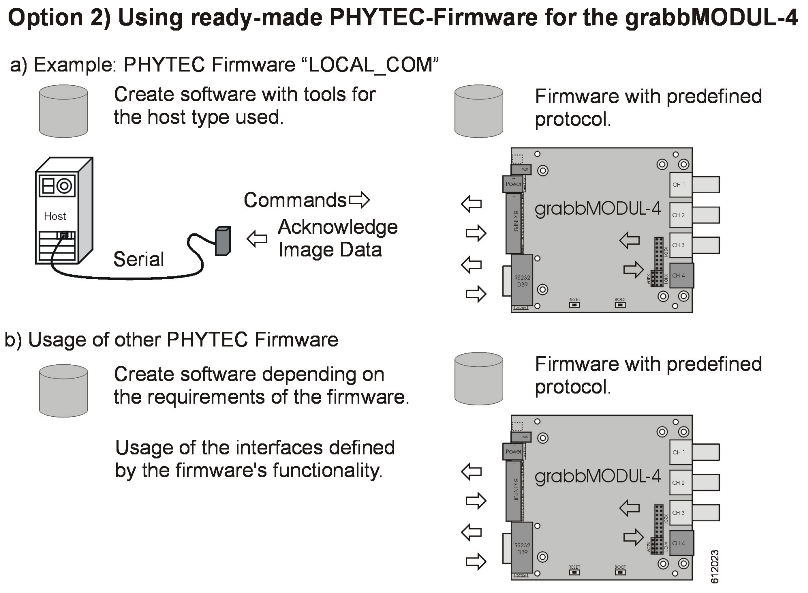
Hint:
To better understand the programming of the grabbMODUL-4 and the structure of the image data, we recommend that you read the following sections: *section 6.2, „Working with Video Data“, section 6.3.1, „Configuring the grabbMODUL-4“ and section 6.3, „Creating Your Own Software (Firmware)“.*

- **Using ready-made PHYTEC Firmware**

  The use of pre-fabricated firmware offers you the advantage of being able to create an application without having to know how to program the grabbMODUL-4.
  However, the pre-fabricated firmware has a fixed function range and is limited to the preprogrammed interfaces. Therefore it is less flexible than a custom-designed firmware.
  The pre-fabricated firmware can be loaded easily to the programming memory of the grabbMODUL-4 with the help of the 'FlashTools' utility, which is part of the module's delivery contents.
  Depending on the function of the firmware, an additional parameterization may be required. Parameterization can be performed over the serial interface with the help of a PC.

  The delivery contents of the grabbMODUL-4 include a firmware for transferring images to a PC. To use this firmware the grabbMODUL-4 has to be connected to the PC via the serial interface. An image capture can be started with control commands. The module transfers the image data to a PC on command, where the data can be evaluated or displayed.
  A corresponding PC program, which carries out the desired function, can be created using standard PC programming tools.

  A description of the control commands and the data protocol on the serial interface can be found in section 6.4.1, „LOCAL_COM: Firmware for Local Image Transferrence“.


Option 2) Using ready-made PHYTEC-Firmware for the grabbMODUL-4
a) Example: PHYTEC Firmware "LOCAL_COM"
Create software with tools for the host type used.
Firmware with predefined protocol.
Host
Commands
Acknowledge
Image Data
Serial
grabbMODUL-4
b) Usage of other PHYTEC Firmware
Create software depending on the requirements of the firmware.
Usage of the interfaces defined by the firmware's functionality.
Firmware with predefined protocol.
grabbMODUL-4
612023

Hint:
To better understand the structure of the image data, we recommend that you read *section 6.2, „Working with Video Data“*.

6.2 Working with Video Data

In this section we will offer a brief glimpse at video technology and the signals and data formats it uses. This knowledge will make it easier for you to understand the processes that lead to an image capture. You will also need this information to understand the data formats of the images and how to work with them.
(This description is based on the European PAL video norms. These statements also apply for the US-American NTSC-system, however the number of lines is different.).

6.2.1 Video Signals and Digitization prodedures

The analog norm video signal processed by the Grabber consists of 625 lines, which are divided into two fields (*field*s) and delivered by the video source in succession. The first field (odd) contains lines 1 through 313, the second (even) contains lines 314 through 625. The fields are interlaced in order to minimize the flicker effect in a TV image. Spatially, line 1 is followed immediately by line 314. If a complete image – called *frame* - is to be displayed, the two successive fields have to be interlaced accordingly (*Figure 33*).

In addition to a few leading and trailing blanking lines the video signal also contains test and data lines as well as lines containing video text information. Therefore the effective image area consists of two fields with 288 lines each.

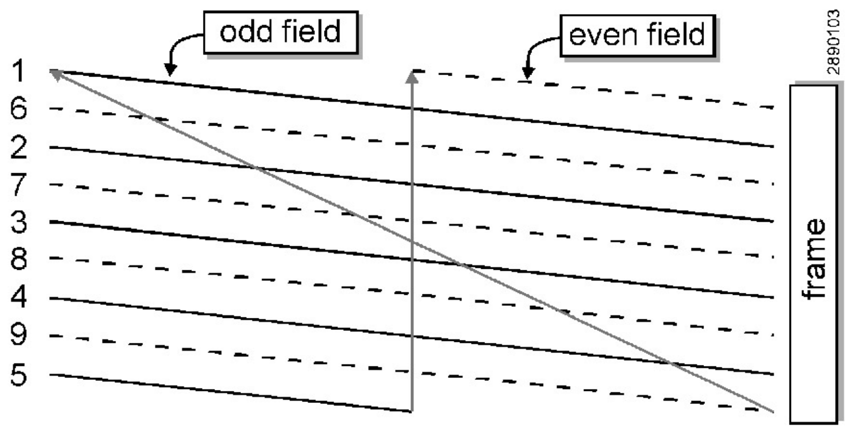


Figure 33: Interlaced Scanning (Example with 9 Lines)

Each field is constructed in 20 ms. In a field, the complete image plane is already recognizable, however the verticle resolution is reduced by half. In many applications this is already enough so that a complete digitization result is present after 20 ms.

In some cases the resolution can also be reduced by half in the X-direction, in order to achieve an undistorted image.

Minimizing the resolution in the X-direction will not speed up the digitization process, since the time allotted for the screen layout is fixed.

If the full TV image resolution is required, then time must be allowed for both fields to be generated (40 ms). The two fields are generated in immediate succession.

So that the two fields fit together properly, the last line of the odd (first) field will be cut in half. Accordingly, the first line of the second field contains the second half of the line.

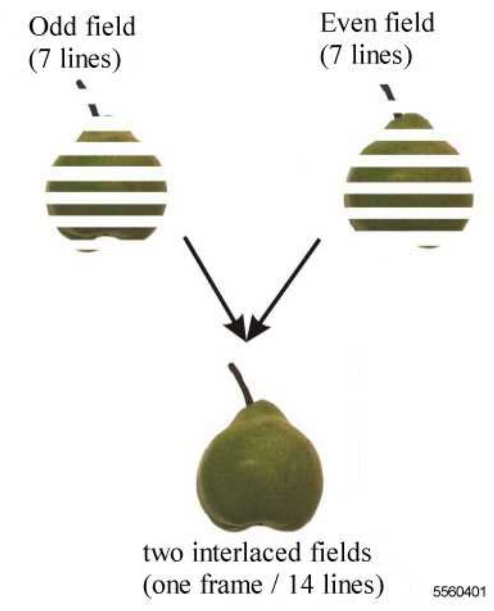


Figure 34: Fields and Frames

One problem with the digitization of TV images is that a fast moving object will have moved noticeably between the capture of the first and second field; therefore the two fields will no longer fit together (blur/distortion).

Frequently only one field is used for this reason – at the cost of the verticle resolution.

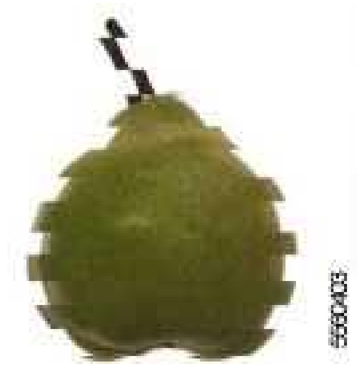


Figure 35: Comb Effect With Moving Objects in Frame Mode (Schematically)

What does the time line of a digitization prodedure look like?

This depends on the point in time that the digitization was started (relative to the image signal sent by the camera) and the selected image mode.

Figure 36 shows the various scenarios:

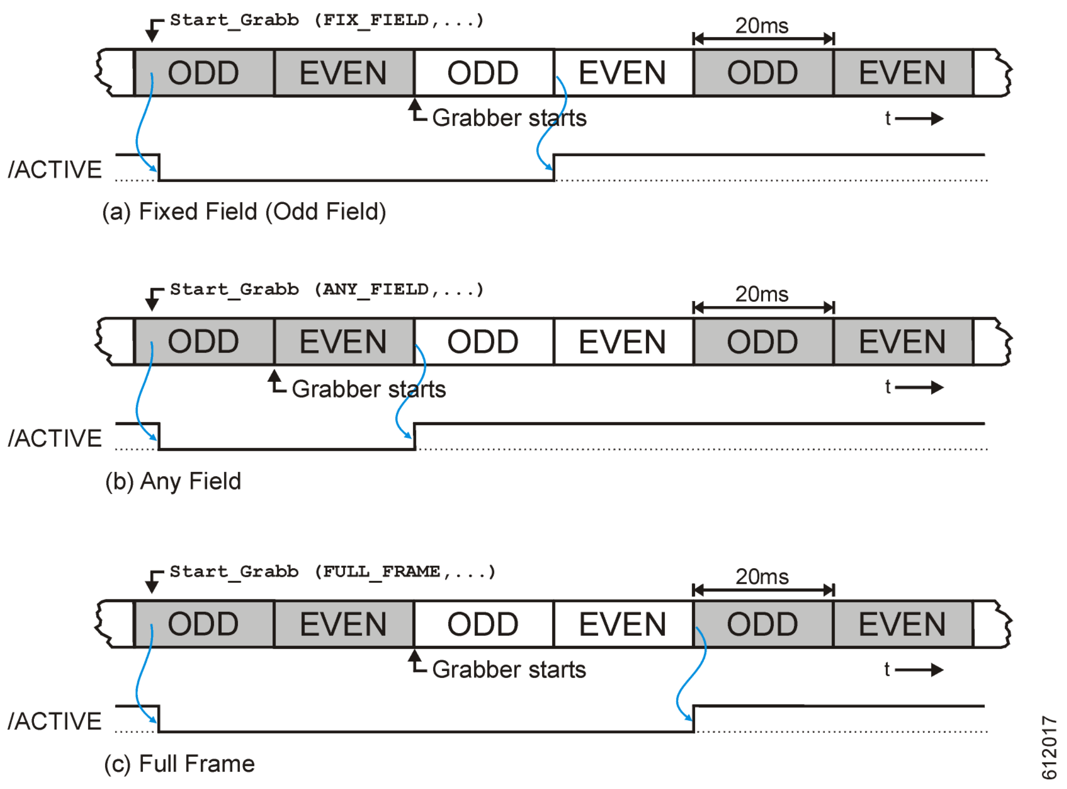


Figure 36: Timing of the Digitization Prodedure

The image signal sent by the camera contains alternatingly the first (ODD) and second (EVEN) field of image. The duration of a field is 20 ms. A complete image consists of two fields (colored accordingly in *Figure 36*).

If an image size of maximum 768 x 288 pixels is sufficient, then only a field will be requested for digitization *(Figure 36a).* In this case, depending on the setting, the grabbMODUL-4 can digitalize the first or second field of a frame. The default setting is „first field“ (ODD); this is also the basis for the depiction in Figure (a).
Generally speaking it doesn’t matter if the first or second field is taken. But the same field should be used each time.

Due to the spatial offset of the fields there could be the impression that the image jumps a line up or down if multiple images are shown in succession.

After starting the Grabber with the command *Start_Grabb()*, the signal */ACTIVE* is set to a logical „0“. This indicates that a digitization process is active. Initially the Framegrabber will not digitize any data, but is in a temporary wait state instead. The image capture only begins once the next complete field that has the required parity (in this example: ODD) begins. Next the image data is written to the video RAM in real time. The image capture is complete at the end of the field and the signal */ACTIVE* returns to the inactive state „l“. Now the video RAM can be accessed by the microcontroller and the image data can be processed.

Caution!
The signal */ACTIVE* shows that the Framegrabber is active. During the time that /ACTIVE = „0“, the video RAM cannot be accessed.
The user program therefore has to check the */ACTIVE* signal after an image capture and may not access the image memory before the next positive edge (transition from 0 → 1) occurs.

Read accesses to the video RAM while */ACTIVE* = 0 result in a random value.

Since there is a slight delay between the Start command and /ACTIVE → 0, to avoid false interpretations /ACTIVE should not be queried immediately following the Start command. A delay of 15 to 20 ms is recommended.

Hint:
Details on the image request and the parameters for the command *Start_Grabb()* can be found in *section 6.3.6.*

Figure (a) shows an example of an unfavorable timing. The Start command comes shortly after the start of an ODD field.

The Framegrabber must allow almost two fields to elapse before the beginning of an ODD-image can start the image capture. Almost 60 ms elapse between the request and the end of the digitization process.

If a short delay time between the start comand the beginning of the image capture is crucial, then the parameter *ANY_FIELD* can be used when starting the Grabber *(Figure 36b).* Here the next field is digitized, regardless of whether it is an ODD or EVEN field. The disadvantage there is that the captured image can jump up a line *(vertical jitter, see above).*

The delay between the start command and the beginning of the digitization in this mode is less than 20 ms. The entire procedure completes itself in maximum 40 ms.

If more than 288 lines are required, a *frame* has to be requested (*Figure 36c*). The parameter *FULL_FRAME* is used here. The timing here is principly the same as a request for a field. However, since in this case two successive fields are captured, the capture process lasts 20 ms longer. The fields are stored spatially in succession in the Video RAM as well and have to be linked together accordingly for display or evaluation (*Figure 37*).

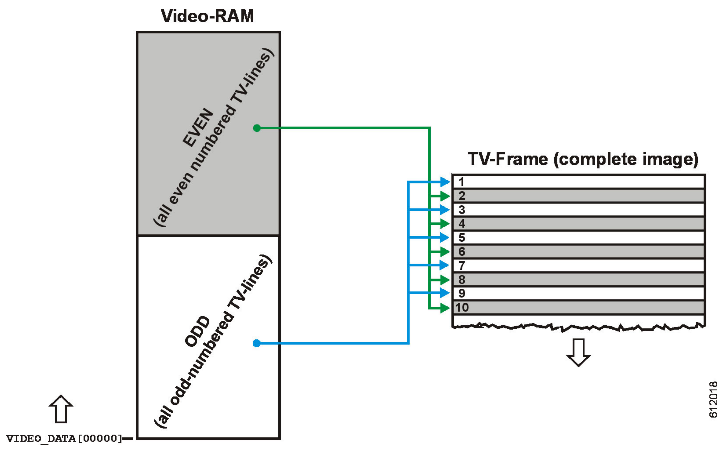

Video-RAM
EVEN
(all even numbered TV-lines)
ODD
(all odd-numbered TV-lines)
TV-Frame (complete image)
1
2
3
4
5
6
7
8
9
10
VIDEO_DATA[00000]
612018

Figure 37: Organization of Image Data when Storing a Frame

6.2.2 Color Transmission and Color Storage

Color information and image brightness are separated in principle when transmitting images for TV systems. The signal for image brightness (Luma signal, Y-signal) and the color difference signal (Chroma signal) are transmitted. The latter characterizes the color of a pixel according to color tone (hue) and color saturation.

TV systems reduce the bandwidth of the color signal compared to the brightness signal. The color information of a pixel is therefore more diffuse than the brightness information. This is analogous to an artist who uses a sharp pen to draw the contures of an object and subsequently uses a large brush to color in the image.

The Y-bandwith for the PAL (B,G,H,I) system is 5 MHz, the Chroma bandwidth is 1.5 MHz.

The Chroma signal is also identified as a U/V-signal in PAL and as a Q/I-signal in NTSC. A V-signal or I-signal characterizes red tones while the U- and Q-signals are assigned to blue-violet color tones. The term Cr/Cb-signal is commonly used for (Chroma Red / Chroma Blue).

The triple value (Y,Cr,Cb) is used to completely define the brightness and color of a pixel. These values can be transmitted to an image processing system for color recognition or control without any additional pre-processing.
To display an image (on a PC for example), as a rule the RGB color format is used. Here information about color and brightness is contained in the primary colors red, green and blue (R, G, B).

The conversion is defined for PAL according to CCIR-recommendation in the following matrix:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1{,}371 & -191{,}45 \\ 1 & -0{,}338 & -0{,}698 & 116{,}56 \\ 1 & 1{,}732 & 0 & -237{,}75 \end{pmatrix} \cdot \begin{pmatrix} Y \\ U \\ V \\ 1 \end{pmatrix}$$

The YCrCb values delivered by the video component in the grabbMODUL-4 correspond to the value range (Y[16...253] and CrCb[2...253] in the default state. The manufacturer of the video processor has given the following formulas for calculating RGB values.

R = 1,64 (Y-16) + 1,596 (Cr-128)
G = 1,64 (Y-16) - 0,813 (Cr-128) - 0,391 (Cb-128)
B = 1,64 (Y-16) + 2,018 (Cb-128)

Y[16...253], Cr/Cb[16...240], RGB[0...255]

Image data is always stored in the grabbMODUL-4 in YCrCb format. This is more efficient, since it requires a smaller volume of data. Instead of three bytes per pixel (RGB format) only two bytes (one word) are required. The lower eight bits specify brightness, the upper eight bits specify color (Cr/Cb).

Color information is included with every Y-value as Cb and Cr, alternatingly. Each pixel contains only „half“ the color information, either the red or the blue portion. The missing information can be obtained by assuming the value of the adjacent pixel. The color information is transmitted and stored with half the resolution of the Brightness information. Since the color information can already be transmitted by TV sytems with a reduced range, this process does not result in any lost information. This data format is identified as YCrCb 4:2:2.
Y_1,$Cr_{1/2}$ is delivered with the first pixel of each line, the second pixel contains Y_2,$Cb_{1/2}$ and so on. *Figure 38* shows how data gets stored in the memory.

How do you access image data that you require?

Figure 38 demonstrates how data is accessed for a color image. Pairs of pixels are always required (= two 16-bit words), in order to get all of the required information. For the first pixel, the missing Cr information is adobted from the color value of the second pixel, while the missing Cb information in the second pixel is assumed from the Cb value of the first pixel. Since the range of the color signal is limited anyway no information is lost, even though this would appear to be the case initially.
Obtaining gray scale data without color information is simple: here only the lower byte of a 16-bit word needs to be read. It contains the Brightness value for every pixel.

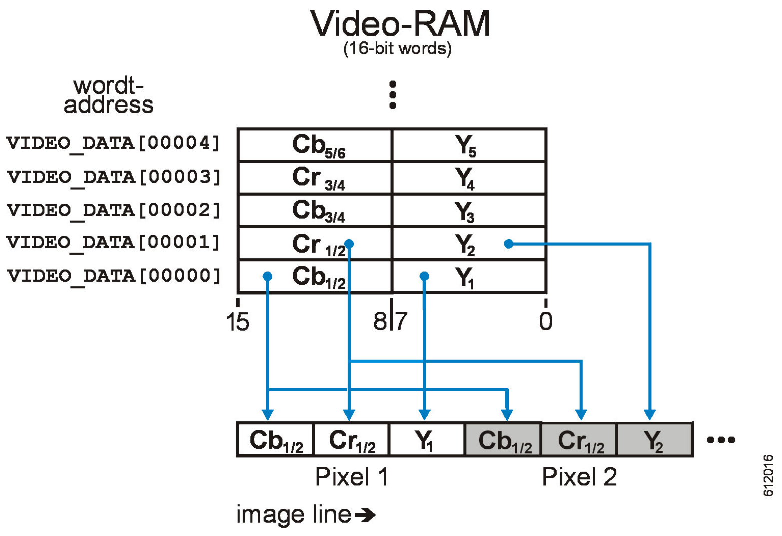


Figure 38: Organization of the Video-RAM

Hint:
When processing color data on the module, it is recommended to use the YCrCb color format, since this mode requires very little memory and offers the advantage of keeping Brightness and Color information separate.
When transmitting color images to a PC, it is also important that YCrCb format be used as long as possible. Firstly, this reduces the amount of data to be transmitted and secondly, the RGB conversion required for display the image on the PC can performed much more quickly.**ion!**
Since pixel pairs are always required for color image processing, the number of pixels per line always has to be even.
Be sure to check this when setting the image size!

By definition, the first 16-bit value in the memory begins with the Cb value, followed by the Cr value.

6.3 Creating Your Own Software (Firmware)

This section describes the basic steps required to create your own software for the grabbMODUL-4.

The following software components are required:

- A PC software tool to create programs for the Infineon C165 microcontroller. We recommend that you use the development tools from Keil Software (Compiler, Assembler, Linker and Hex-converter), contained in µVision. *For information on installation of the software developent tools, please refer to the Manufacturer's information sheets*.
- *Note:* The follwing examples can also be processed with the 'Light'-Version of µVision.
- A PC software utility that establishes serial communication with the grabbMODUL-4 and stores the generated Hex-program in the module's Flash memory. PHYTEC's *FlashTools* can be used here. Instructions on use of FlashTools can be found in *section 2.4, „Installation of the PHYTEC FlashTools and Downloading a Program Code"*.

Procedure:4

- Start the µVision Tool:
  - Create a new project or load an existing grabbMODUL-4 project
  - Connect the Phytec hardware driver (does not apply for existing grabbMODUL-4 project)
  - Write the program
  - Compile, assemble, link and create a Hexfile
- Start FlashTools:
  - Download the Hexfile to the Flash memory
- Finished! – The program will run independently on the grabbMODUL-4 after a Power On or a Reset.

6.3.1 Configuring the grabbMODUL-4

The grabbMODUL-4 in standard configuration is populated with 256k RAM, 256k Flash and 1 MB Video RAM. These memory devices are each assigned to a Chip Select signal via hardware and can therefore be assigned to certain memory ranges with software (*see Figure 39*)

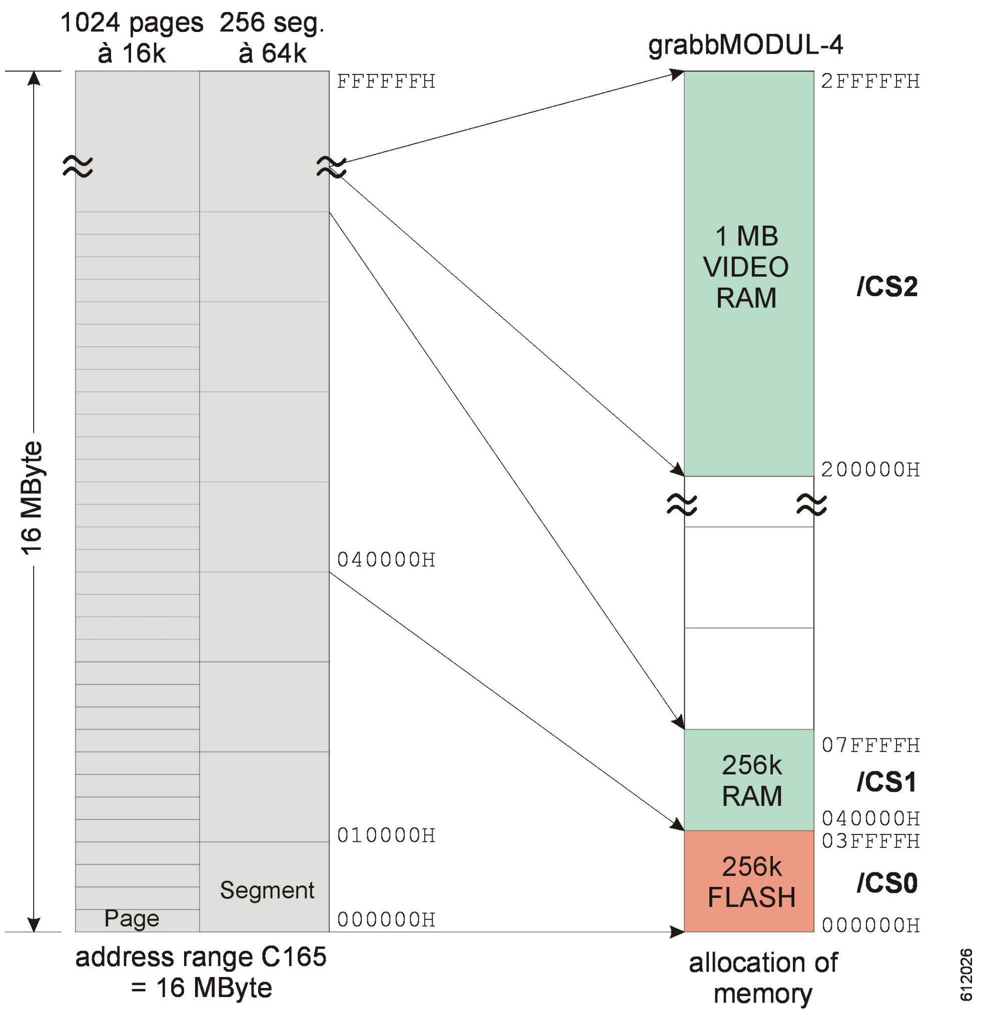


Figure 39: Memory Allocation on the grabbMODUL-4

The allocation of the memory areas in the basic configuration is divided as shown in *Figure 39*:

Memory Type	Chip Select Signal	Memory Range
256k FLASH	/CS0	000000H – 03FFFFH
256k RAM	/CS1	040000H – 07FFFFH
1MB VIDEO-RAM	/CS2	200000H – 2FFFFFH

The allocation occurs in the file „***startfla.a655***“ and must be taken into consideration in the project settings.
Additional allocations are then only necessary if the memory areas are used differently by the user or if a grabbMODUL-4 variant with a different memory configuration is to be used. The following memory configurations are possible

	64k	256k	512k	1MB	2MB
FLASH		x	x	x	x*
RAM	x	x		x	
VIDEO-RAM	x	x		x	

*) Only limited use possible.

6.3.1.1 Addressing the Memory

In general the RAM/Flash memory addressing is monitored in the software tool’s project settings. For direct addressing the configuration shown in *Figure 39* must be used.
The VIDEO-RAM memory area can be addressed directly in C using the array „unsigned int VIDEO_DATA“ which is defined in „startfla.a65“. A direct read access of a 16-bit pixel value is possible:

```
    Pixel = VIDEO_DATA[Pixelposition];
```

or by component:

```
Pixel Brightness = (unsigned char)
VIDEO_DATA[Pixelposition];
PixelColor = (unsigned char) (VIDEO_DATA[Pixelposition]>>8);
```

The orginization of pixel information in this 16-bit value is explained in *section 6.2.2* and illustrated in *Figure 39*.

Hint:
Controller access to the VIDEO_DATA[] has read-only restrictions!

6.3.2 Editing an Existing Project

<u>Task:</u>

Expanding a current project as follows:

- Query Video Channel 4 to see whether a video signal is present
- Output information over the serial interface

Realization:

In the following example, the µVision development studio from Keil is used. The project settings are for a memory configuration of 256k RAM, 256k FLASH and 1MB of Video memory on the module. The Hex file to be generated must be capable of running from the module's Flash memory.

1) Create a new folder on your hard drive (e.g. ***...\gm4_Test***).

2) Copy the entire folder ***„grabbMODUL4\Modul_Programm\Sourcen\gm4_Test\“*** from the grabbMODUL-CD to the folder you just created on your hard drive.

3) Start the µVision IDE on your PC and load the project „`gm4_Test.Uv2`“ from the index. All project settings relevant for the grabbMODUL-4 are predefined for this project. Furthermore, all software drivers are linked to the project and the necessary software initializations have been called.

   Note: a free 'Light' version of the µVision development studio is provided on the CD SO-670.

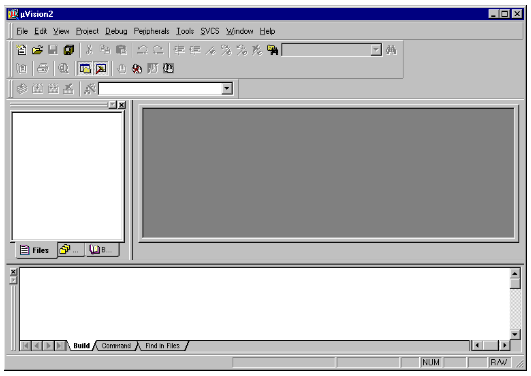
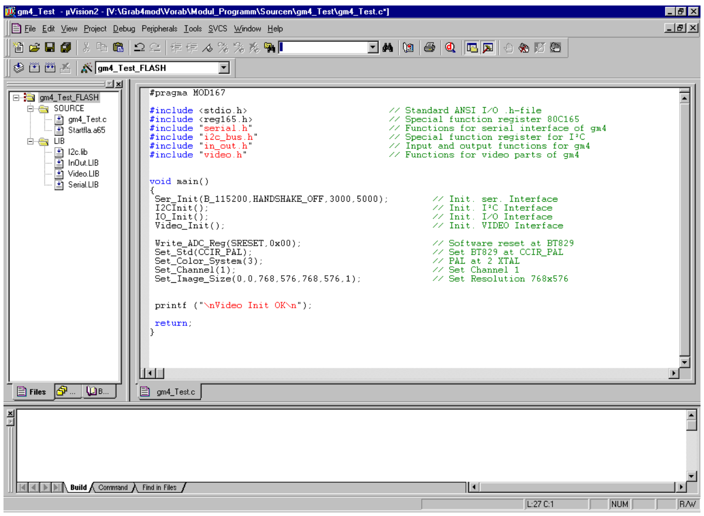

µVision2
File Edit View Project Debug Peripherals Tools SVCS Window Help
Files
Build
Command
Find in Files
NUM
R/W

Figure 40: uVision IDE

4) Activate the tab sheet „***gm4_Test.c***“ to edit the software source.


gm4_Test - µVision2 - [V:\Grab4mod\Vorab\Modul_Programm\Sourcen\gm4_Test\gm4_Test.c*]
File Edit View Project Debug Peripherals Tools SVCS Window Help
gm4_Test_FLASH
SOURCE
gm4_Test.c
Startfla.a65
LIB
I2c.lib
InOut.LIB
Video.LIB
Serial.LIB
Files
gm4_Test.c
Build
Command
Find in Files
L:27 C:1
NUM
R/W

```
#pragma MOD167

#include <stdio.h>                              // Standard ANSI I/O .h-file
#include <reg165.h>                             // Special function register 80C165
#include "serial.h"                             // Functions for serial interface of gm4
#include "i2c_bus.h"                            // Special function register for I²C
#include "in_out.h"                             // Input and output functions for gm4
#include "video.h"                              // Functions for video parts of gm4


void main()
{
 Ser_Init(B_115200,HANDSHAKE_OFF,3000,5000);        // Init. ser. Interface
 I2CInit();                                         // Init. I²C Interface
 IO_Init();                                         // Init. I/O Interface
 Video_Init();                                      // Init. VIDEO Interface

 Write_ADC_Reg(SRESET,0x00);                        // Software reset at BT829
 Set_Std(CCIR_PAL);                                 // Set BT829 at CCIR_PAL
 Set_Color_System(3);                               // PAL at 2 XTAL
 Set_Channel(1);                                    // Set Channel 1
 Set_Image_Size(0,0,768,576,768,576,1);             // Set Resolution 768x576


 printf ("\nVideo Init OK\n");

 return;
}
```

Figure 41: Project „gm4_Test.uv2“

5) In main() search for the free line between printf („\nVideo Init OK\n“); and return; :

```
printf ("\nVideo Init OK\n");
                                        ⇦  free line
return;
```

By this point all relevant initializations have been carried out for:

- the serial interface (115,200 Baud, 8-bit)
- Basic Grabber functions
- Video signals (PAL / CCIR)

6) After the line printf *("\nVideo Init OK\n");* enter the following code:

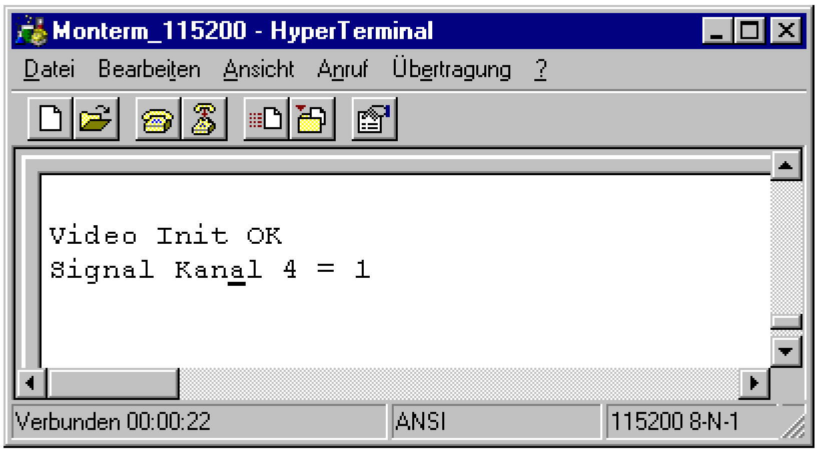
```
while(1) {
 printf ("\rSignal Kanal 4 = %i",((unsigned int)(Read_ADC_Reg(STATUS)&0x80)>>7));
 }
```

The „Signal-Present"-Flag will then be queried continuously in the STATUS register of the video device. *A more detailed description of the individual flags in the video device is available in the BT829A video processor manual.*

7) Next create a ***gm4_Test.h86*** – Hex-file from these sources using the function *Project* ⇨ *Rebuild all target files*

8) To load the ***gm4_Test.h86*** – file onto the grabbMODUL-4, please use PHYTEC FlashTools. Instructions for downloading a file are provided in *section 2.4, „Installation of the PHYTEC FlashTools and Downloading a Program Code".*

9) Since this program is constantly outputting the „Signal-Present-Flag" on the serial interface following a module reset, you will need a way of making this information visible as well. To do this, use a terminal program on your PC. The structure of the terminal program and instructions for starting it are described in *section 2.5.4, „Testing the Module Program with a Terminal Program".*


Monterm_115200 - HyperTerminal
Datei Bearbeiten Ansicht Anruf Übertragung ?
Verbunden 00:00:22
ANSI
115200 8-N-1

```
Video Init OK
Signal Kanal 4 = 1
```

Figure 42: Terminal Program with Test Output

If a video signal is connected to Channel 4, a „1" should be output by the terminal program. If a „0" is output, check the video line or the camera signal.

Hint:
When using other variants of the grabbMODUL-4 (VM-004-Xx) the special project subdirectories must be selected. The included firmware „LOCAL_COM“ can only be edited with the full version of the Keil compiler due to its size.

6.3.3 Creating Your Own Project

Task:

Creating a project for the grabbMODUL-4:

- Configure project settings
- Link libraries and sources to the project
- Create a source file with the following functions:
  - Initialize the serial interface
  - Perform basic initialization of the Grabber functions on the module
  - Query Video Channel 4 to see if a video signal is connected
  - Output information on the serial interface

Realization:

In the following example, the µVision IDE from Keil is used. The project settings are valid for the following memory configuration: 256 KB RAM and 256 KB Flash as well as 1 MB video memory on the module. The Hex file to be generated should be capable of running from the module’s Flash memory.

1) Create a new folder on your hard drive (e.g. ***...\gm4_Test\***)

2) Copy the following file from the CD to your local ***...\gm4_Test\...*** folder:

•	**...ModulProgramm\LIBs\Video_Lib\Hlarge\**	**VIDEO.LIB**
•	**...ModulProgramm\LIBs\Video_Lib\**	**VIDEO.H**
•	**...ModulProgramm\LIBs\Serial_Lib\Hlarge\**	**SERIAL.LIB**
•	**...ModulProgramm\LIBs\Serial_Lib\**	**SERIAL.H**
•	**...ModulProgramm\LIBs\InOut_Lib\Hlarge\**	**INOUT.LIB**
•	**...ModulProgramm\LIBs\InOut_Lib\**	**INOUT.H**
•	**...ModulProgramm\LIBs\I2C_Lib\Hlarge\**	**I2C.LIB**
•	**...ModulProgramm\LIBs\I2C_Lib\**	**I2C_Bus.H**
•	**...ModulProgramm\LIBs\I2C_Lib\**	**I2C_Hard.H**
•	**...ModulProgramm\LIBs\I2C_Lib\**	**Misc.H**
•	**...ModulProgramm\Start-Up\**	**StartFLA.A65**

Hint:
The file ***„Startup.a65“*** is required if you want to create a module program for processing code from the RAM.

3) Start the µVision IDE on your PC.

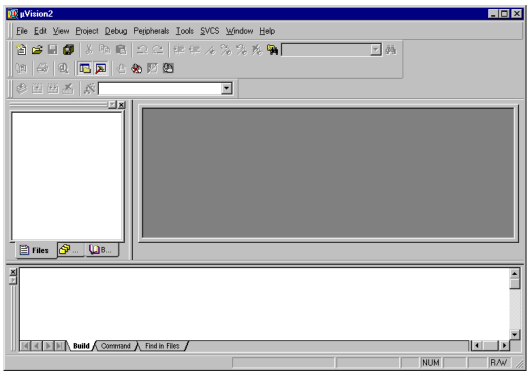

µVision2
File Edit View Project Debug Peripherals Tools SVCS Window Help
Files
Build
Command
Find in Files
NUM
R/W

Figure 43: µVision IDE

Note: a free ‘Light' version of the µVision development studio is provided on the CD SO-670.

4) Select *Project* ⇨ *New Project* and create a project with the name ***„gm4_Test“***.

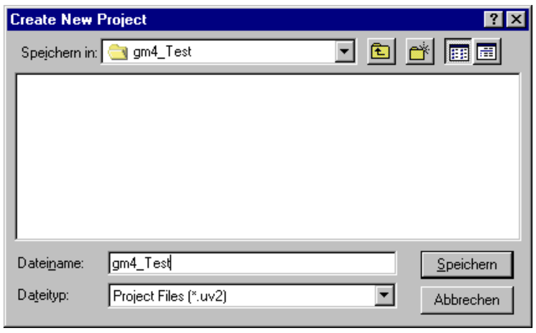
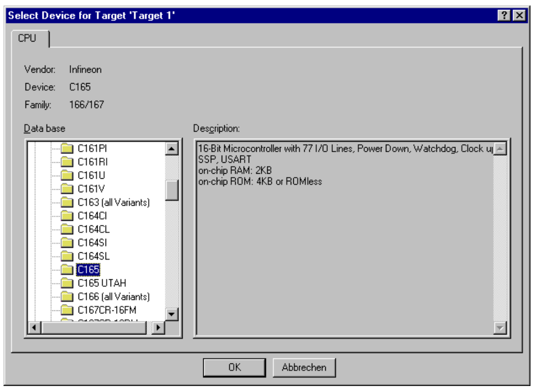


Figure 44: Creating a New Project

5) Now select *Infineon* ⇨ *C165* and confirm with OK.



Vendor: Infineon
Device: C165
Family: 166/167

16-Bit Microcontroller with 77 I/O Lines, Power Down, Watchdog, Clock u
SSP, USART
on-chip RAM: 2KB
on-chip ROM: 4KB or ROMless

Figure 45: Selecting the Controller

Enter a project name in the project window such as ***„gm4_Test_FLASH“***

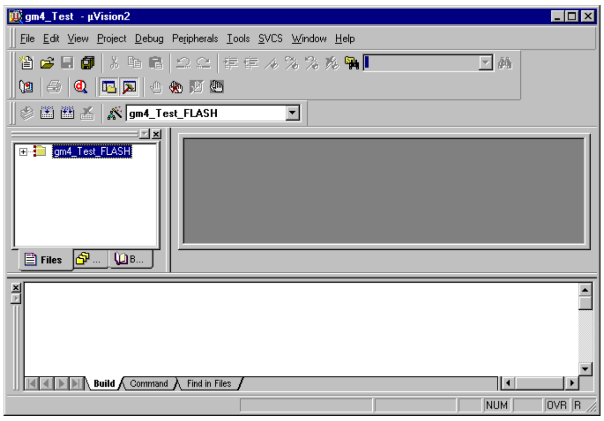


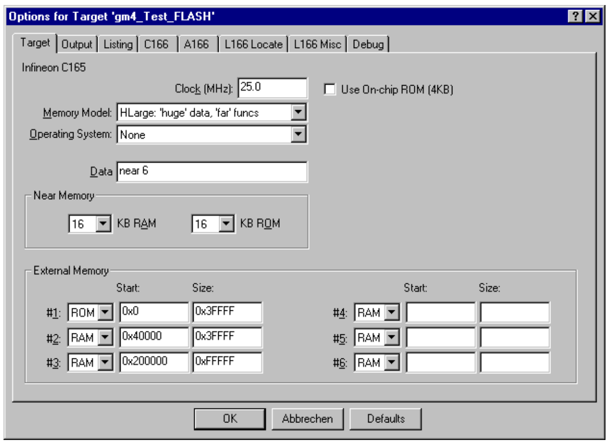
Figure 46: Assigning a Project Name

Properties are assigned in the next step under

Project ➪ Options for Target ‚gm4_Test_FLASH‘.

The following settings must be entered in all tab sheets.

Tab Sheet: Target


Options for Target 'gm4_Test_FLASH'

Target | Output | Listing | C166 | A166 | L166 Locate | L166 Misc | Debug

Infineon C165

Clock (MHz): 25.0 ☐ Use On-chip ROM (4KB)

Memory Model: HLarge: 'huge' data, 'far' funcs

Operating System: None

Data: near 6

Near Memory: 16 KB RAM, 16 KB ROM

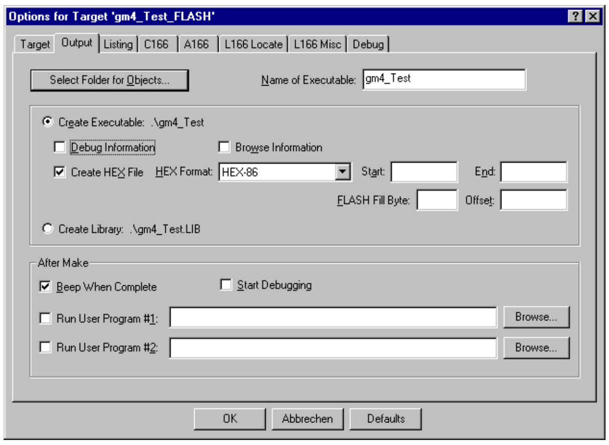
External Memory

		Start:	Size:			Start:	Size:
#1:	ROM	0x0	0x3FFFF	#4:	RAM		
#2:	RAM	0x40000	0x3FFFF	#5:	RAM		
#3:	RAM	0x200000	0xFFFFF	#6:	RAM		

Figure 47: Tab Sheet „Target“

Memory Model ➪ „Hlarge“
External Memory ➪ #1 ➪ „ROM“ ➪ „0x000000“ ➪ „0x3FFFF“
External Memory ➪ #2 ➪ „RAM“ ➪ „0x040000“ ➪ „0x3FFFF“
External Memory ➪ #3 ➪ „RAM“ ➪ „0x200000“ ➪ „0xFFFFF“

Tab Sheet: Output


Options for Target 'gm4_Test_FLASH'

Target | Output | Listing | C166 | A166 | L166 Locate | L166 Misc | Debug

Select Folder for Objects... Name of Executable: gm4_Test

● Create Executable: .\gm4_Test

☐ Debug Information ☐ Browse Information

☑ Create HEX File HEX Format: HEX-86 Start: End:

FLASH Fill Byte: Offset:

○ Create Library: .\gm4_Test.LIB

After Make

☑ Beep When Complete ☐ Start Debugging

☐ Run User Program #1: Browse...

☐ Run User Program #2: Browse...

Figure 48: Tab Sheet „Output“

Name of Executable: ➪ „gm4_test“
Create Executable ➪ Activate: ☑ Create HEX File

Next the individual source and library files have to be linked to the project. For this purpose two „Source Groups“ were created in the project window. Also add the two groups „SOURCE“ and „LIB“ to *Project ⇨ Targets, Groups, Files, ...*

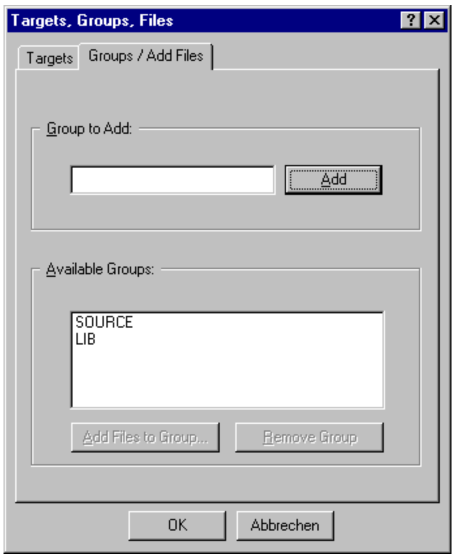


Figure 49: Tab Sheet „Targets, Groups, Files ...“

By right-clicking on „Source Group“ the following files will be added to both „Source Groups“:

- SOURCE ⇨ StartFLA.A65
- LIB ⇨ VIDEO.LIB, SERIAL.LIB, INOUT.LIB, I2C.LIB

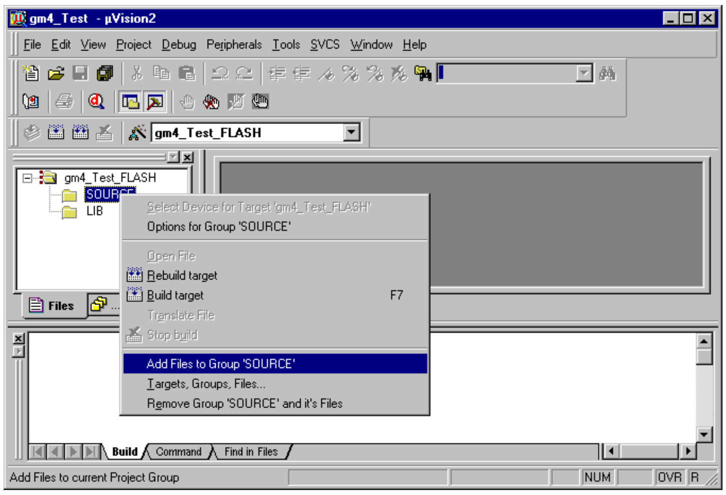

gm4_Test - µVision2
File Edit View Project Debug Peripherals Tools SVCS Window Help
gm4_Test_FLASH
gm4_Test_FLASH
SOURCE
LIB
Files
Select Device for Target 'gm4_Test_FLASH'
Options for Group 'SOURCE'
Open File
Rebuild target
Build target F7
Translate File
Stop build
Add Files to Group 'SOURCE'
Targets, Groups, Files...
Remove Group 'SOURCE' and it's Files
Build
Command
Find in Files
Add Files to current Project Group
NUM
OVR R

Figure 50: Tab Sheet “Add Files to Group…”

Finally a file, in which the C-code is to be created, has to be linked to the project. To do this create a new file, save it under the name ***„gm4_Test.c“*** and link it to the Source Group ***„SOURCE“***. Your project interface should now be properly structured:

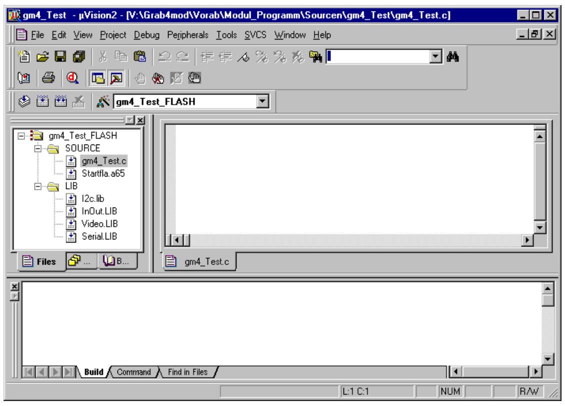

gm4_Test - µVision2 - [V:\Grab4mod\Vorab\Modul_Programm\Sourcen\gm4_Test\gm4_Test.c]
File Edit View Project Debug Peripherals Tools SVCS Window Help
gm4_Test_FLASH
SOURCE
gm4_Test.c
Startfla.a65
LIB
I2c.lib
InOut.LIB
Video.LIB
Serial.LIB
Files
gm4_Test.c
Build
Command
Find in Files
L:1 C:1
NUM
R/W

Figure 51: Project „gm4-Test“

Add the following code to the file *„gm4_Test.c“:*

```
#pragma MOD167
#include <stdio.h>                  // Standard ANSI I/O .h-file
#include <reg165.h>                 // Special function register C165
#include "serial.h"                 // Functions for serial interface of gm4
#include "i2c_bus.h"                // Special function register for I²C
#include "in_out.h"                 // Input and output functions for gm4
#include "video.h"                  // Functions for video parts of gm4

void main()
{
 Ser_Init(B_115200,HANDSHAKE_OFF,3000,5000);    // Init. ser. Interface
 I2CInit();                                     // Init. I²C Interface
 IO_Init();                                     // Init. I/O Interface
 Video_Init();                                  // Init. VIDEO Interface

 Write_ADC_Reg(SRESET,0x00);                    // Software reset at BT829
 Set_Std(CCIR_PAL);                             // Set BT829 at CCIR_PAL
 Set_Color_System(3);                           // PAL at 2 XTAL
 Set_Channel(4);                                // Set Channel 4
 Set_S_Video();                                 // Set S-Video Input
 Set_Image_Size(0,0,768,576,768,576,1);         // Set Resolution 768x576

 printf ("\nVideo Init OK\n");

 while(1) {
   printf ("\rSignal Kanal 4 = %i",((unsigned int)(Read_ADC_Reg(STATUS)&0x80)>>7));
   }

 return;
}
```

The following program sequence is carried out:

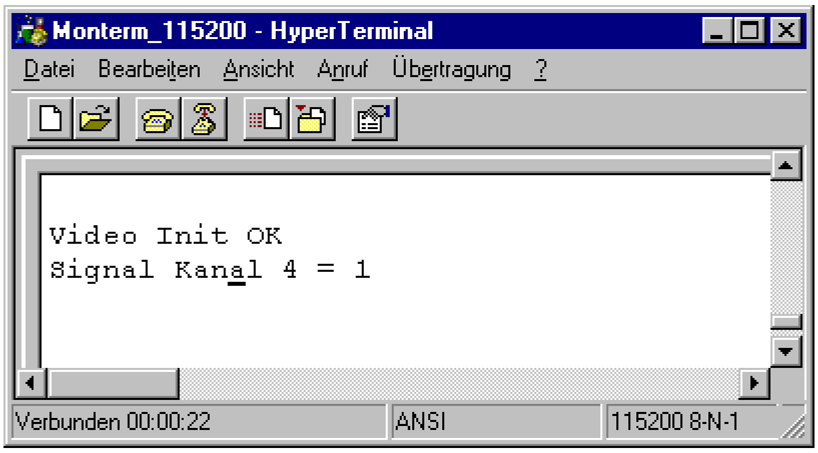
- Setting the MOD167
- Linking all Header files
- Initialization of the serial interface
- Initialization of the I²C routines (access to the video processor)
- Initialization of the I/O ports on the grabbMODUL-4
- Initialization of the video processor
- Software reset of the video processor
- Setting the CCIR/PAL standard
- Selecting the CCIR/PAL clock on the module
- Setting the video input to channel 4 (S-Video)
- Querying the Signal-Present flag in the video device (*see Manual BT829A*)

Hint:
The function properties are explained in this manual along with the library description.

Create a ***gm4_Test.h86*** Hex file from these sources with *Project ⇨ Rebuild all target files*.

Use PHYTEC FlashTools to load the ***gm4_Test.h86*** file into the grabbMODUL-4. The proper technique for downloading a file is described in *section 2.4, „Installation of the PHYTEC FlashTools and Downloading a Program Code“.*

Since this program generates the „Signal-Present Flag“ continuously on the serial interface following a module reset, you will also need a way to make this information visible. To do this use a terminal program on a PC. The layout and procedure for starting the terminal program are described in *section 2.5.4, „Testing the Module Program with a Terminal Program“.*


Monterm_115200 - HyperTerminal
Datei Bearbeiten Ansicht Anruf Übertragung ?
Verbunden 00:00:22
ANSI
115200 8-N-1

```
Video Init OK
Signal Kanal 4 = 1
```

Figure 52: Terminal Program with Test Output

The Terminal Program should generate a „1“ if a video signal is connected to channel 4. If a „0“ is generated check the video line or the camera signal.

6.3.4 Basic Procedure for Image Capture.

In this section, a few brief examples should serve to clarify the basic procedure for initialization of the module and processing image data.
Since the functions used and the structure of the image data are described in detail in other sections, this section merely explains the sequence of the function calls.

Caution!
This section includes descriptions of some fundamental procedures. Be sure to read this section before you create your own programs.

Initialization of the Microcontroller Kernel:
This occurs primarily in ***„Startup.a65“*** (RAM-variant) or in ***„startfla.a65“*** (FLASH-variant). The modifications are explained in the file headers.
Furthermore, there are specific memory assignments that have to be configured in the project settings of the project interface. Use the example projects supplied by PHYTEC for this (*see sections 6.3.2f.*)

General Initialization:

These are only required if additional functions are to be used.

Ser_Init (B_115200, HANDSHAKE_OFF, 3000, 5000);
- Initializes the serial interface, for example.

Initialization of the Video Devices:

This is required to be able to capture video images and also to access the digitized images.

I2CInit();
- Initializes the I²C routines in order to have access to the video device's setting register. This call **is required** if the grabb-function is to be used!

IO_Init();
- Initialization is only necessary if the I/O ports on the grabbMODUL-4 are to be used.

Video_Init();
- Initialization of all devices necessary for digitization. This call **is required** if the grabb-function is to be used!

Settings Prior to an Image Capture:

The following functions have to be called prior to the first image capture:

Write_ADC_Reg (SRESET, 0x00);
- Brings the video device in a defined initial state. This function should only be used once at the start of the program!

Set_Std (CCIR_PAL);
- Basic initialization of all registers in the video processor BT829. The presettings are made for a PAL video source.

Set_Color_System (3);
- The color system is set and the physical arrangement to the clock sources on the module are established.

Set_Channel (1);
- Setting of the channel that the video source (camera) is connected to.

Set_Image_Size (0 ,0 ,768 ,576 ,768 ,576 ,1);
- This function is used to set the size and form of the image to be digitized (how many pixels with which scaling are stored in the video memory).

Capturing an Image:

The capture procedure has to be started for an image to be digitized and stored in the video memory. During the digitization process no access to the image data is possible. Depending on the image size and the point in time the process starts the digitization can last up to 80 ms (*see Figure 36*).

Start_Grabb (IMAGE_TYPE, RESET);
- Starts the grabb procedure relevant to the video image that is next in line and that meets the IMAGE_TYPE requirement.

while (!noACTIVE);
- Waits until the grabb procedure is finished.

Accessing Image Data:

The structure of the image data is clarified in *section 6.2.2, „Color Transmission and Color Storage“*. To reach a specific pixel position, you will require the vales that were set in the *Set_image_Size()* function.

<u>**Task 1:**</u>
The Y/C value of the pixel on the position x=384 and y=288 of a frame (768 x 576 pixel interlaced image) should be read:

1) Calling the Set_Image Function:

```
Set_Image_Size (0 ,0 ,768 ,576 ,768 ,576 ,1);
```

2) A digitized image is stored in the VIDEO_DATA memory of the module, with the following data format:

   Field one with columns [1,2,3, ...,768] of the odd rows [1,3,5,...,575] and subsequently field 2 with columns [1,2,3, ...,768] of the even rows [2,4,5,...,576]. Every pixel corresponds to a 16-bit value, which is divided into Y [Bits 7..0] and C [Bits 15..8].

3) Since the VIDEO_DATA range is organized by word, the (unsigned int) pixel value can be read directly:

```
Pixel_Value = VIDEO_DATA[Pixel_Position];
```

   with

```
Pixel_Position = 144L*768L + 348L
```

   The pixel value is then split into a Y and a C value:

```
Y_Luma = ((unsigned char)Pixel Value);
C_Chroma = ((unsigned char)(Pixel Value >> 8));
```

How to calculate the position of a pixel in the Video RAM?

It is tricky to determine the memory address for a pixel of a frame (which consists of two consecutive fields).

To calculate the memory address for a certain pixel position, we have to take the following into consideration:

- Regarding memory addresses and arrays, programmers usually start counting with zero. So the first pixel in tv-line no. 1 will be indexed as pixel position (0,0) with x=0 and y=0. In difference counting tv-lines starts with 1.
- The two interlaced fields of an image frame are stored in the Video RAM in sequence (as shown in *Figure 37*). Thus, a field is stored in two portions: The first contains all odd tv-lines, the second all even lines. So we have to work with a case differentiaton.
- The array of image data is mapped to a linar organized memory. All pixels of a line are stored one after another and all lines are also stored one after another. For the calculation of the start of a certain line, the total number of pixels in one line will be essential.

However, since programmers start counting y-positions with 0, all pixels with *even* y-position numbers will be found in the *first* portion of the Video RAM.

given:
P = pixels per line (768 in our example)
L = number of tv-lines (576 in this case)
(x, y) = matrix position of the pixel

wanted:
pixel_position = address of this pixel in the Video-RAM

(a) for ***even*** values of y :

$$pixel\_position = \frac{y}{2} \cdot P + x$$

(b) for ***odd*** values of y :

$$pixel\_position = \frac{y - 1 + L}{2} \cdot P + x$$

In single field mode, the calculation of the corresponding pixel address is much easier since no interlacing has to be taken into consideration:

For non-interlacing mode:

$$pixel\_position = y \cdot L + x$$

- At least 2 sequential Cr/Cb values are required for calculating RGB, *see 6.2.2., „Color Transmission and Color Storage“.*
- If you require the pixel value in the following column of the frame, the pixel position = 144 * 768 + 384 + number of all pixels in the first field (768 * 288).
- Only a read access of the VIDEO_DATA[] range is possible from the controller!

<u>Task 2:</u>
The Y/C value of the pixel should be read from row y=144, column x=192 of a field (384 x 288).

Realization:

1) Calling the Set_Image Function:

```
Set_Image_Size (0 ,0 ,384 ,576 ,384 ,576 ,1);
```

1) The result is a digitized image in the module's VIDEO_DATA memory, which is divided in the following manner:
   Field 1 with scaled columns [1,2,3, ...,384] and the scaled rows [1,2,3,...,288]. Every pixel corresponds to a 16-Bit value, which is divided into Y (Bit7..0) and $C_{R/B}$ (Bit15..8) segments.

2) Since the VIDEO_DATA range is organized by word, the (unsigned int) pixel value can be read directly:

```
 Pixel Value = VIDEO_DATA[Pixel Position];
```

   with

```
 Pixel Position = 144L * 384L + 192L;
```

   The pixel value is then divided into Y and C values.

```
Y_Lumina = ((unsigned char)Pixel Value);
C_Chroma = ((unsigned char)(Pixel Value >> 8));
```

Hint:
- At least 2 sequential Cr/Cb values are required for calculating RGB, *see section 6.2.2, „Color Transmission and Color Storage*.“.
- Only a read access of the VIDEO_DATA[] range is possible from the controller!

Function Characteristics:

This portion of the manual refers to the special characteristics of a few functions and their use.

1) The following functions should only be called once at the start of the program:

```
Write_ADC_Reg (SRESET, 0x00);
Set_Std ();
Set_Color_System ();
```

2) When switching these channels with the *Set_Channel()*-function, there will be level changes on the video device’s analog inputs. The result is that the video signal can only be correctly recognized after a specified time. Therefore an image should not be digitized immediately after a channel has been switched. Wait until:

   a) the signal is detected correctly

   ```
   Set_Channel(1);
   Wait_Luma_OK();
   ```

   b) or a certain time has lapsed (approx. 300 ms):

   ```
   Set_Channel(1);
   Delay(300);
   ```

   If an S-Video source is used, the channel has to be set and the S-Video signal must be set as the signal type:

   ```
   Set_Channel(4);
   Set_S_Video();
   ```

   When switching back the signal type „FBAS“ has to be set, to digitize composite-type signals:

   ```
   Set_Channel(1);
   Set_FBAS();
   ```

3) During the digitization process (max. 80 ms) no access to the image data is possible. The controller can continue to function normally during this time however, and access the current digitization state over the *noACTIVE*-Flag.

   noACTIVE = 0 ➔ Grabb procedure started/active
   noACTIVE = 0 ➔ Grabb procedure ended/not started

   It is then only necessary to wait until the grabb procedure is complete if work with the image data is to begin immediately following *Start_Grabb()*:

```
Start_Grabb();
while (!noACTIVE);
```

   If the processing is split up well, the microcontroller can perform other tasks that do not require access to the stored image data.

6.3.5 Drivers for the Microcontroller C165 Processing Kernel

The grabbMODUL-4 with its C165 processing kernel contains considerably more components and function devices than a module with a single processing kernel. Therefore PHYTEC Messtechnik has summarized the fundamental and general software processes as functions. These are made freely available to the user in corresponding *Libraries*.
The individual *Libraries* are introduced and described in detail in the following sections. To use the functions in the *Libraries*, they have to be linked to the user project along with the corresponding *Header-Files*.
New or expanded Libraries available to you on our homepage.

6.3.6 Video.Lib – Drivers for Framegrabber Functions

The *Video.Lib* contains all functions for initializing and setting the parameters of the BT829A video digitizer. Grabber control is also initialized.
The I²C routines are necessary for the *Video.Lib* to be used. Therefore the *I2C.lib* has to be linked to the project that you are creating and the *i2c_bus.h* must be included as well.
The definition BYTE stands for „unsigned char“ in these libraries and the definition WORD stands for „unsigned int“.

Caution!
The I²C-interface must be initialized before using a function from the *Video.Lib* for the first time. In addition the function **I2Cinit()** must be called one time. Furthermore *Video.h* has to be included in your project.

The following functions are made available in th *Video.Lib*:

- Write_ADC_Reg()
- Read_ADC_Reg()
- Video_Init()
- Start_Grabb()
- Set_FIX_Field()
- Set_Std()
- Set_Color_System()
- Set_Brightness()
- Set_Contrast()
- Set_Saturation()
- Set_Hue(int)
- Get_Brightness()
- Get_Contrast()
- Get_Sat_U()
- Get_Sat_V()
- Get_Hue()
- Set_S_Video()
- Set_FBAS()
- Set_BW()
- Set_LDec()
- Set_AGC()
- Set_CKill()
- LumaControl()
- Set_Channel()
- Set_Image_Size()
- Wait_Luma_OK()

The functions and their parameters are described in detail in the following sections.

6.3.6.1 Write_ADC_Reg ()

Function: writes a Byte to one of the Video processor's registers

int Write_ADC_Reg(BYTE reg_no, BYTE parameter)

reg_no:	Address of the register in the video processor
parameter:	Byte to be written
Return Value:	0 = OK -1 = Error

This function writes a Byte to one of the video processor's registers. The register address and the value to be written are specified. The success of the operation can be checked by the return value.

6.3.6.2 Read_ADC_Reg ()

Function: Reads a video processor register

BYTE Read_ADC_Reg(BYTE reg_no)

reg_no:	Address of the register to be written
Return Value:	Content of the register

This routine enables a video processor register to be read.

6.3.6.3 Video_Init()

Function: Initialization of the Video Framegrabber

void Video_Init(void)

This function initializes the microcontroller's I/O pins required for controlling the Video-Grabber.

6.3.6.4 Start_Grabb()

Function: Start Image Capture

int Start_Grabb (BYTE field, BYTE set_count)

field:	Code for the image type to be captured: 0 = FIX_FIELD (default ODD) 1 = ANY_FIELD 2 = FULL_FRAME
set_count:	0 = V-RAM counter continues counting 1 = V-RAM counter is reset before the capture
Return Value:	0 = OK, -1 = Error

This function causes an image to be digitized and stored in the Video-RAM of the grabbMODUL-4. The progress of the Grabb procedure can be followed using the status pin *noACTIVE*:

noACTIVE = 0 ➔ Grabb procedure started/active
noACTIVE = 1 ➔ Grabb procedure ended/not started

The following image data is digitized based on the command given over the *field* variable.

1) **FIX_FIELD:** The call of the routine with this parameter initiates the digitization of a field with the parity defined in the function *Set_FIX_FIELD()* (default: ODD, first field). The next odd or even field is grabbed. The duration of the digitization process depends on the set vertical resolution and scaling. For a complete field it lasts 20 ms. Depending on the starting time and the vertical position of the image outline, up to 40 ms can pass before the capture begins (total time max. 60 ms).

2) **ANY_FIELD:** This parameter initiates the digitization of the next available field. It doesn't make a difference whether it is an odd or even field. The maximum delay time here is approximately 20 ms (if the beginning of the vertical outline is the same as the beginning of the image). The total capture time is max. 40 ms. The application range is any application which requires an extremely fast reaction to a trigger signal.

3) **FULL_FRAME:** A call with this parameter gives the command for the digitization of a frame consisting of two sequential fields. Both fields are stored sequentially in the Video RAM. For frame digitization it is important to start with the ODD field, so that the images can then be correctly interlaced. The duration of the digitization procedure is 40 ms. Depending on the starting time, up to 40 ms can pass before the beginning of the capture procedure (total duration max. 80 ms).

The *set_count* parameter determines where the requested image is stored in the video memory:

1) ***Set_count=0:*** The next image to be digitized is stored one position after the previously grabbed image in the memory. If no image was stored in the previous field then this is the memory position Zero in the Video RAM (start address Video RAM +0x0000). Using this function it is possible to store multiple images one after another in the Video RAM of the grabbMODUL-4.

2) Set_count=1: The next image to be digitized is stored in the Video RAM after memory position Zero.

Hint:
In most applications it is possible to work with the set_count = 1. The image is then written in after the beginning of the Video RAM.

If you work with the parameter set_count = 0, then you can store as many images one after another as there is Video RAM available.
With a memory configuration of 1 M (512k = Luma, 512k = Chroma) It would be possible to store 32 images with 128 x 128 pixel resolution (= 16 kByte per image) in the Video RAM of the grabbMODUL-4. Since Luma and Chroma have separate memory areas, in this example it does not matter of the image is required in black and white or color.

If the upper limit of the Video RAM is exceeded then the write pointer will jump back to the beginning of the image memory and the image data will continue to be written there (ring memory structure). Data that is located in these positions will be overwritten.

Please note when reading data, that if the *set_count* = 0 there is always a blank pixel space between the pixels being saved. This pixel must be skipped over when the information is being read.

By setting and resetting the output bit *INIT* the image data write pointer can be moved to the position Zero in the Video RAM.

6.3.6.5 Set_FIX_Field

Function: Selects a field for FIX-Field function

int Set_FIX_Field (BYTE odd_even)

odd_even:	Coding for the field to be captured 0 = EVEN-Field (2nd field) 1 = ODD-Field (1st field)
Return Value:	0 = OK, -1 = Error

Using this function you can select which of the two fields will be digitized when *Start_Grabb()* is called with the parameter FIX_FIELD.

Hint:
The selection must be made before *Start_Grabb()* is called and cannot be changed until the digitization procedure is complete.
By default the first field (ODD) is selected for digitization.

6.3.6.6 Set_Std

Function: Initializes the video processor with standard values

int Set_Std (BYTE parm_list[])

param_list[]:	Pointer to an array with the contents of the video processor register 0x00..0x1A
Return Value:	0 = OK, -1 = Error

With this function it is possible to set the video processor registers with standard values. A pointer is given indicating the array with the contents of the register 00_{Hex} bis $1A_{Hex}$.

For *param_list[]* the user can access the following predefined values in the file Video.Lib:

1)	**CCIR_PAL[]**	= Settings for 720x576 CCIR PAL
2)	**Square_PAL[]**	= Settings for 768x576 Square-pixel PAL
3)	**CCIR_NTSC[]**	= Settings for 720x480 CCIR-NTSC
4)	**Square_NTSC[]**	= Settings for 640x480 Square-pixel NTSC

Pre-initialization of a video device register should occur at the beginning of the program, even if individual settings (e.g. resolution) have to be changed while the program is running.

The Call

```
#include "Video.h"
...
Set_Std(Square_PAL);
```

sets up the Grabber’s video processor for processing PAL image sources.

Hint:
We recommend that you initialize the device with this routine, even if you want to modify individual values later.
Without the initialization the device will not function correctly and image capture will either not function at all or be disturbed.

6.3.6.7 Set_Color_System

Function: Set the grabber to the applied color system

int Set_Color_System (BYTE colsys)

colsys:	Code for the color system 0 = NTSC 1 = PAL 2 = AUTO 3 = PAL_ONLY
Return Value:	0 = OK, -1 = Error

The color system to be used by the Grabber is set with this function. The video processor's clock frequency and input register are set accordingly.

1) **PAL:**	Sets the Grabber for use with PAL video sources (if only one oscillator is populated).
2) **NTSC:**	Sets the Grabber for use with NTSC video sources.
3) **AUTO:** a	Recognizes the color system automatically (only if video signal is connected and both oscillators are populated).
4) **PAL_ONLY:**	Sets the Grabber explicitly for use with PAL (only if both oscillators are populated).

Hint:

- If the *colsys* – Parameter ‚AUTO' is used, incorrect interpretations of the color system can result. In this case the color system might be set incorrectly. If guaranteed recognition of the system is required, it is recommended that the explicit settings NTSC or PAL_ONLY are used for known video sources.
- In the standard variant VM-004 both oscillators are populated. Here you cannot use the ‚PAL' parameter! This parameter is only meant for special variants equipped with only one oscillator.

6.3.6.8 Set_Brightness

Function: Sets the image brightness

int Set_Brightness (int bright)

bright:	Image brightness [-128...127], Default= 0 (0%)
Return Value:	0 = OK, -1 = Error

Sets the register for image brightness in the video processor. The value determines a constant that is added to the brightness value of a pixel in the video processor. The brightness can vary in a range from –100% to +100%:

bright = Brightness [%] · 1,27 [1/%]

6.3.6.9 Set_Contrast

Function: Sets the image contrast

int Set_Contrast (int contr)

contr:	Image Contrast [0...511], Default = 216 (100%)
Return Value:	0 = OK, -1 = Error

Sets the contrast of the captured image. The contrast value represents a constant factor, multiplied by the brightness value of the pixel data in the video processor (scaled accordingly). The setting can be in a range from 0% to 236.57%:

contr = Contrast [%] 2.16 [1%]

6.3.6.10 Set_Saturation

Function: Sets the color saturation

int Set_Saturation (int sat_u, int sat_v)

sat_u:	Saturation of the U-color portion [0...511], Default = 254
sat_v:	Saturation of the V-color portion [0...511], Default = 180
Return Value:	0 = OK, -1 = Error

With this function the color saturation of the image can be adjusted separately for the U and the V-color portion. Normally the relationship of the values *sat_u* and *sat_v* is equal. Color aberrations (e.g. by unadjusted cameras), can be smoothed out by adjusting the U- and V- values thereby adjusting the color of the image.

sat_u = U-Saturation [%]· 2,54 [1/%]; range from 0% to 201,18 %
sat_v = V-Saturation [%]· 1,8 [1/%]; range from 0% to 283,8 %

Hint:

- Please note that due to the different conversion factors for U and V, the Null setting (100 %) has different register values (for U: 254, for V: 180).
- U and V form a color vector. The length of the vector defines the color saturation, the angle defines the color tone. (The V-axis characterizes the red tones, the U-axis characterizes the blue tones.)

6.3.6.11 Set_Hue

Function: Corrects color tone (only with NTSC)

int Set_Hue (int hue)

hue:	color tone, phase adjustment of the color carrier [-128...127], Default = 0°
Return Value:	0 = OK, -1 = Error

This function sets the hue for digitization of NTSC color images, by varying the phasing of the color carrier. With PAL this value has no meaning, since the color system automatically corrects phase errors. Settings in a range from –90° to +89,3° are possible:

hue = color carrier phase [°]· 1,422 [1/°]

6.3.6.12 Get_Brightness

Function: Reads back the brightness setting

int Get_Brightness (void)

Return Value:	Contents of the video processor’s brightness register Image Brightness [-128...127]

6.3.6.13 Get_Contrast

Function: Reads back the contrast setting

int Get_Contrast (void)

Return Value:	Contents of the video processor’s contrast register Image Contrast [0...511]

The function reads the contents of the contrast register from the video processor and returns it as an integer value.

6.3.6.14 Get_Sat_U

Function: Reads the contents of the U color saturation register

int Get_Sat_U (void)

Return Value:	Value of the current U color saturation [0...511]

The contents of the U color saturation register can be determined by calling this routine.

6.3.6.15 Get_Sat_V

Function: Reads the contents of the V color saturation register

int Get_Sat_V (void)

Return Value:	Value of the current V color saturation [0...511]

The contents of the V color saturation register can be determined by calling this routine.

6.3.6.16 Get_Hue

Function: Reads back the contents of the color tone register

int Get_Hue (void)

Return Value:	equivalent to the set phase length of the color carrier [-128...127],

This function is used for reading back the set hue value.

6.3.6.17 Set_S_Video

Function: Sets the S-Video mode

int Set_S_Video (void)

Return Value:	0 = OK, -1 = Error

When an S-Video source is connected, the video processor's second ADC must be activated. This function switches on the Chroma-ADC and deactivates the now unnecessary color trap in the Luma path, whereby the image becomes sharper. Furthermore, the input channel is automatically switched to the S-Video socket.

S-Video sources are color cameras that are equipped with a special output that sends the signals for brightness and color separately.

6.3.6.18 Set_FBAS

Function: Sets the module to FBAS Mode (Composite Inputs)

int Set_FBAS (void)

Return Value:	0 = OK, -1 = Error

By calling this routine, the Grabber is switched back to FBAS mode: the Chroma-ADC is switched off and the color trap is reactivated. This mode has to be set if the composite signals are to be digitized. The FBAS mode is set in the standard initialization.

Composite-sources are color cameras that transfer color and brightness signals over a shared line (e.g. over a BNC or Cinch socket). The image quality is somewhat less than what can be achieved with S-Video, since interference (Moiré) can occur on certain structures. Composite signals are also described in German according to their components as FBAS signals (<u>F</u>arb-<u>Figure</u>-<u>A</u>ustast-<u>S</u>ynchon-Signal).

6.3.6.19 Set_BW

Function: Switches on/off the color trap for black & white operation

int Set_BW (int on)

on:	0 = color-Signal on input (switch on color trap), Default 1 = b/w-Signal on input (switch off color trap)
Return Value:	0 = OK, -1 = Error

If a black & white camera is connected to the Grabber instead of a color camera, then the color trap which keeps color interference away from the brightness signal (Cross-Color-Effect), is no longer necessary. With this function the color trap can be switched on and off.

It makes sense to shut off the color trap if black & white signals are used, since with the removal of the band filter the image sharpness will increase slightly.

Hint:

- The color trap can only be shut off if a black & white camera is connected. Otherwise interference lines will be visible in the image.
- In S-Video mode the deactivation occurs automatically.
- If you switch back to the composite mode with Set_FBAS, the color trap is activated (color video sources are then supposed).
- If you are not certain which image source will be connected to the input then the color trap should always remain activated. The minor reduction in quality is tolerable for most applications.

6.3.6.20 Set_LDec

Function: Switches the Luma low pass filter on and off

int Set_LDec (int on, int HFilt)

on:	0 = Switch off Luma decimation, default 1 = Switch on Luma decimation
HFilt:	0 = Automatic filter selection, default 1 = CIF Filter (only with active Luma decimation) 2 = QCIF Filter (only with active Luma decimation) 3 = ICON Filter (only with active Luma decimation)
Return Value:	0 = OK -1 = Error -2 = invalid combination (on = 0 and Hfilt = 1,2,3)

With small image formats (smaller than 2:1) you can achieve a better image quality if the input signal resolution is reduced (image sharpness is adapted to reduced image resolution). With this routine it is also possible to integrate a low pass filter in the Luma path.

With the parameter *Hfilt* the filter switched on by the parameter *on* is adapted to the image size.
Automatic filter selection makes the filter setting dependent on the set image size (see *Set_Image*). The filter can also be adapted to one of the standardized image formats *CIF*, *QCIF* or *ICON*.

Hint:
If the filter type 1, 2 or 3 is selected with *Hfilt*, then this is only possible with the low pass filter *on = 1*.

6.3.6.21 Set_AGC

Function: Sets the various AGC functions

int Set_AGC (int CAGC, int AGC, int Crush)

CAGC:	0 = Non-adaptive Chroma AGC, Default 1 = Adaptive Chroma AGC
AGC:	0 = AGC switched off 1 = AGC switched on, Default
Crush:	0 = Non-adaptive AGC, Default 1 = Adaptive AGC
Return Value:	0 = OK, -1 = Error

The grabbMODUL-4 has two AGCs (automatic gain control loops). The general AGC monitors the signal level of the composite or Y-signals and regulates the input amplitude. In addition, the Chroma-AGC ensures adaptation of the color carrier amplitude.
The composite/Y-AGC can be switched on or off with the parameter AGC.
If *Crush*=1, the composite/Y-AGC uses an adaptive automatic controller action to adapt itself to the input video signal.
The Chroma-AGC uses an adaptive automatic controller action to adapt itself to the color carrier amplitude of the input video signal.

Hint:

- Due to the hardware wiring on the grabbMODUL-4, the AGC always **has to be** switched on.
- By setting the parameters *CAGC* and *Crush*, an optimal high-contrast image can be generated. These parameters should not be used with applications that work with absolute brightness or color (e.g measurement tasks).

6.3.6.22 Set_CKill

Function: Switches the color killer on or off

int Set_CKill (int CKill)

Ckill:	0 = Switches off the color killer, default 1 = Switches on th color killer
Return Value:	0 = OK, -1 = Error

If black & white image sources are connected to a color system, the result can be a slight color noise. The color killer prevents this effect by checking for the presence of the color burst and automatically switches off the color processing if necessary.

In special applications it may be necessary to evaluate a color signal with a weak color carrier. In this case the automatic color killer can be switched off.

6.3.6.23 LumaControl

Function: Sets the output format of the brightness signal.

int LumaControl (int Range, int Core)

Range:	0 = Value range Luma/Y, 16-253, Default 1 = Value range Luma/Y, 0-255
	1 = all brighness values <=8 are set to 0 2 = all brightness values <=16 are set to 0 3 = all brightness values <=32 are set to 0
Return Value:	0 = OK, -1 = Error

With this function the output format of the brightness value can be adapted to the application.
Range determines the value range for brightness (possible gray values).

- *Range* = 0 corresponds to the normal value range, which is specified in CCIR 601. The brightness range is thereby limited to the values [16...253], whereby the value Y = 16 corresponds with black. The color value corresponds to [2...253] with Cr/Cb = 128 as zero (signed representation).
- *Range* = 1 enables the use of the full value range [0...255], whereby the value Y = 0 corresponds with black. The color range corresponds to [2...253] with Cr/Cb = 128 as zero (signed).

Setting the *Core* paremeter presents another possibility of influencing the brightness values. The value ranges determined by *Core* (0-3) are assigned the value of zero (reduces dark image noise).

6.3.6.24 Set_Channel

Function: Sets the input channels

int Set_LDec (BYTE channel)

channel:	input channel to be set [1..4], optional [1..8] Default = 1
Return Value:	0 = OK, -1 = Error

The input channel can be selected with this function. The multiplexer located in the digitizer is switched. Channel 4 is used in S-Video operation to feed Luma components (brightness).
As an option the grabbMODUL-4 can be equipped with an additional multiplexer. In this case the channel range is expanded from 1 to 8. The *Set_Channel* function then takes over the switching of the additional multiplexer.

Hint:

- When switching to the S-Video input (see *Set_S-Video*) the corresponding channel 4 is set automatically.
- Be aware of the switching times when switching from one channel to another (*see section 6.3.4*).

6.3.6.25 Set_Image_Size

Function: Sets image size and scaling

int Set_FIX_Field (int hpos, int vpos, int hsize, int vsize, int ppl, int lines, int col_system)

hpos, vpos:	Position of the upper left corner of the captured image on the video screen: (hpos = horizontal, vpos = vertical)
hsize :	Size of the image in the X-direction
vsize :	Size of the image in the Y-direction
ppl :	Desired image resolution in the X-direction (Pixel Per Line, horizontal resolution)
lines :	Desired line count for the captured image (vertical resolution)
col_system:	Color system used: 0 = NTSC 1 = PAL 2 = automatic recognition
Return Value:	0 = OK, -1 = Error

With the routine *Set_Image_Size()*, the size, position and scaling of an image captured by the grabber can be set.
The parameters *hsize* and *vsize* are used to create a window with the size of the desired image (in pixels). *Hsize* determines the number of pixels in the X-direction that the captured image will have, *vsize* determines the number of pixels in the Y-direction.
The values *ppl* and *lines* determine how many pixels from the original video image should be generated. *ppl* defines the number of pixels per image line, *lines* determines how many lines („pixels in the Y-direction“) are generated. Both values are used to determine the image’s *resolution*.
A frame in PAL format has 768 pixels x 576 lines. In order to digitize the image with the highest resolution possible, *ppl* is assigned the value 768 and *lines* is set at 576. The lowest resolution for PAL is 50 x 40 pixels per frame.

The definition of ppl and lines refers to frames and the real number of lines of the TV image to be digitized is divided into two half images (fields). Therefore, the width to height ratio for the highest possible *half-image resolution* is 720 x 288 pixel and thus distorted by a factor of two. In order to eliminate the distortion, the two half-images (even and odd field) have to be digitized and interlaced line by line (resulting resolution of 768 x 576) or the resolution must be reduced to *ppl*=384, *lines*=576, and uses the horizontal properly proportioned resolution of 384 x 288.

Note:
With measurement and automation applications it is not absolutely necessary that you have a properly proportioned representation, as long as the distortion is accounted for in the algorithm. Therefore it is possible to use the half-image resolution 768 x 288 for measurements, that can be more precise in the X-direction than in the Y-direction.

The window defined by *hsize* and *vsize* is now the image portion seen from the digitized image with the dimensions *ppl x lines*.
If *hsize* = *ppl* and *vsize* = *lines* then the entire digitized image is visible; if the values are smaller then only a cropped image of the size defined is visible. The relationship of *hsize* and *vsize* does not change the proportions of the image, since no scaling occurs here but instead a cropped image is taken from the already scaled image. *Figure 53* and *Figure 54* illustrate the meaning of the parameters.

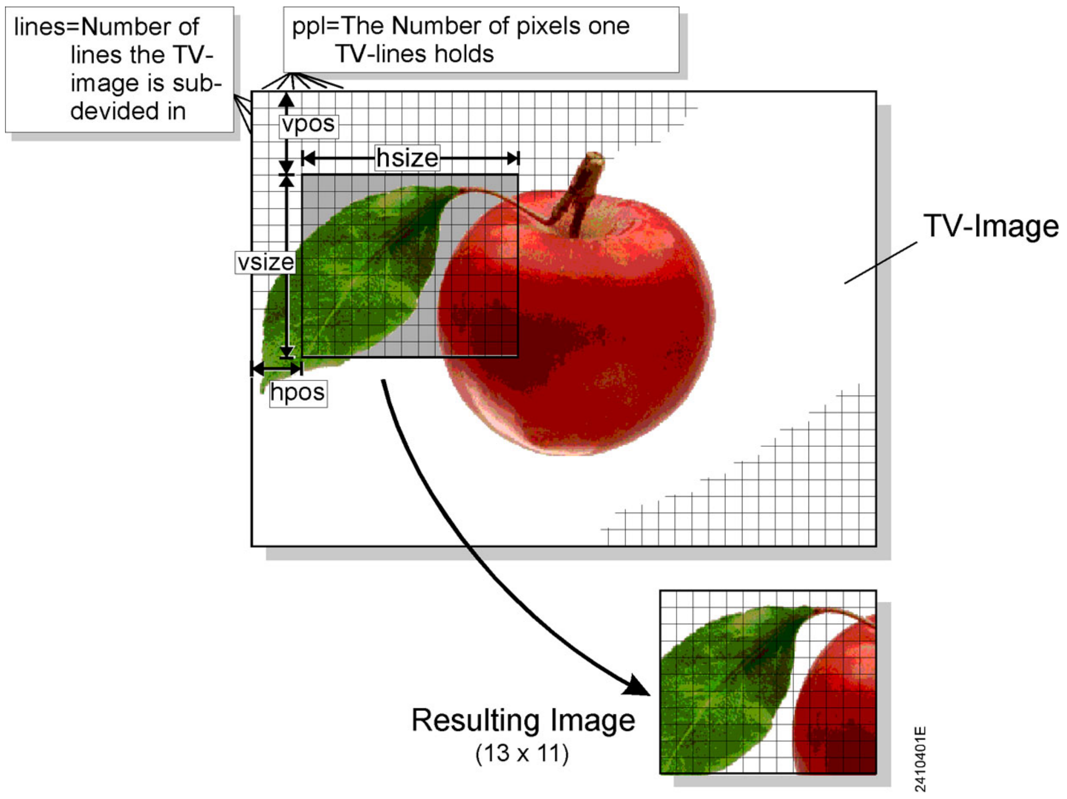

lines=Number of lines the TV-image is sub-devided in
ppl=The Number of pixels one TV-lines holds
vpos
hsize
vsize
hpos
TV-Image
Resulting Image
(13 x 11)
2410401E

Figure 53: Scaling and Cropping an Image

Caution!
vsize and *lines* always have to be given for a frame, even if only a field is digitized and grabbed.

If the image cutout defined by *hsize* and *vsize* is smaller than the image size defined by *ppl* and *lines*, then the window can be moved within the digitized image with the values of *hpos* and *vpos*.
If *hpos*=0 and *vpos*=0 then it will be located in the upper left corner of the digitized image.

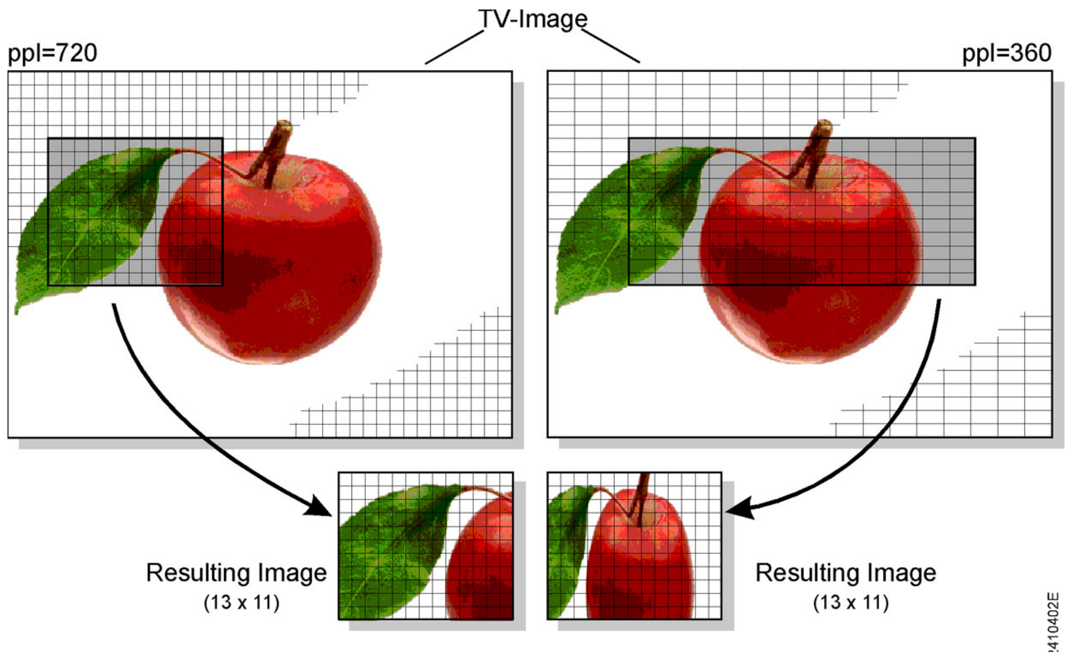


Figure 54: Example for Scaling, All Values are Equal up to ppl

Caution!

- The window area defined in terms of size by *hsize* and *vsize* and in terms of position by *hpos* and *vpos* cannot go outside the range of the digitized image defined by *ppl* and *lines* at any position:

hpos=100, hsize=200, ppl=200:	not allowed, since the last 100 pixels are not defined
hpos=0, hsize=200, ppl=200:	allowed; all pixels are within the image range
hpos=100, hsize=100, ppl=200:	allowed
hpos=100, hsize=200, ppl=300:	allowed
hpos=300, hsize=300, ppl=800:	not allowed: image area has more pixels than can be delivered in TV-norm (corresponding in Y-direction)

- All parameters in the horizontal direction must have *even* values! (ppl=123 is not allowed, ppl=124 is allowed.)

- All parameters in the vertical direction must have even values for frame digitization!

Caution!
Since pixel pairs are always required for color image processing, the number of pixels per line always has to be even.
Be sure to check this when setting the image size!
By definition, the first 16-bit value in the memory begins with the Cb value, followed by the Cr value.

Examples:

- A frame 768 x 576 pixels is to be digitized with a resolution and proportions that correspond to a TV image:

  ```
  Set_Image_Size (0,0,768,576,768,576,1)
  ```

  Caution!
  The digitized image consists of 2 fields, that have to be interlaced manually when displayed.

- A field (CIF) 384 x 288 pixels is to be digitized with a resolution and proportions that correspond to a TV image:

  ```
  Set_Image_Size (0,0,384,576,384,576,1)
  ```

  Hint:
  The digitized image shows the same image range as with the frame digitization, however the resulting image is only half as large (half resolution).

Detailed Example:

- A quadratic image 256 x 256 pixels in size is to be digitized with a resolution and proportions that correspond to a TV image.

(a) Resolution / Scaling

A field with 288 lines is sufficient for digitization. So that the height/width relationship is correct, the resolution in the X-direction also has to be reduced by half (because: *Half* Image = Half Height). Therefore horizontal: 768 : 2 = 384.
The result is that *ppl*=384
Since in the vertical direction the line value always has to be set according to frames, the line value can be calculated as follows:

lines=288 · 2 = 576

From this frame we will only be digitizing a field, i.e. half the line count, whereby the value we want is based effectively on 288 lines.

(b) Window Size

The image should have quadratic measurements of 256 x 256 pixels. From this we can calculate the following directly:

hsize=256, *vsize*=256 · 2 = 512

Note that the vertical value here has also to be calculated according to frames, even if only a field is used later.

(c) Positioning

It is reasonable to center the picture window. In the X-direction only 256 of the 384 pixels will be shown in the window. There will be a border of 384-256=128 pixels, which should be distributed evenly on both sides, i.e. 64 pixels on the left and the right. *hpos* is the size of the left border, therefore *hpos*=64.
Corresondingly in the Y-direction: (576-512):2=32
In reference to a frame: *vpos*=32 · 2=64

```
Set_Image_Size (62,32,256,512,384,576,1)
```

Note:
Here it is incorrect to set ppl=256 and lines=256 · 2=512. This is because these settings would change the width:height ratio to 1:1 (the TV-standard defines 4:3) and the image would become distorted.

The *col_system* parameter defines which TV color system is used. The routine uses the *col_system* indication to correctly set the scaling parameters. Instead of using the counting values 0, 1, 2 the predefined constants *NTSC, PAL, AUTO* can also be used.

6.3.6.26 Wait_Luma_OK

Function: Waits until the connected video signal has been received.

int Wait_Luma_OK (void)

Return Value:	0 = Signal OK, -1 = Timeout Expired

The video device is equipped with an AGC which has a digitization window that automatically adapts to the level of the input signal (Luma components). If a video signal is connected, the automatic adaptation requires a certain time to complete.
As long as the signal has not yet been received, a Luma-ADC-Overflow will continue to be generated. The *Wait_Luma_OK* function waits until this Luma-ADC-Overflow-Flag is no longer set.
This function should be used if an image is to be grabbed immediately after a channel is switched.

Hint:
If you want to grab an image immediately after a channel has been changed, first call *„Wait_Luma_OK()“* and then wait a few milliseconds until the signal is connected and fully stable. Otherwise disturbances in the brightness level can occur.

6.3.7 Serial.lib – Drivers for the Serial Interface

The *Serial.lib* contains functions for initializing and working with the serial interface of the 80C156 microcontroller.
The interface is made available on the D-SUB connector plug described in *section 3.1.2, „Serial Interface“*. The interface can also operate in Hardware-Handshake and characters are received over an interrupt routine.

In this library the definition ‚BYTE' stands for „unsigned char“ and the definition ‚WORD' stands for „unsigned int“.

Caution!
Before using a function from the *Serial.Lib* for the first time, the function **Ser_Init()** must be called once. In addition to this, Serial.h must be included in your project.

The following functions are made available in the *Serial.lib*:

- Ser_Init ()
- Set_Get_Ser_Timeout()
- Init_Buffer ()
- Put_Buffer ()
- Get_Buffer()
- Buffer_Status()
- Char_Present_TimeOut()
- get_ser()
- put_ser()
- read_ser()
- Read_Error_Code_Serial()

6.3.7.1 Ser_Init

Function: Initialization of the serial interface with 8 data bits, one stop bit and without parity.

int Ser_Init (WORD Baud rate, BYTE handshake,
WORD get_ser_timeout, WORD put_ser_timeout)

Baud rate:	desired Baud rate for the serial interface
	B_9600 = 9600 Baud,
	B_57600 = 57600 Baud
	B_115200 = 115200 Baud

handshake:	Setting the Hardware-Handshake (CTS, RTS)
	HANDSHAKE_ON = Hardware-Handshake active
	HANDSHAKE_OFF = Hardware-Handshake inactive

get_ser_timeout:	Timeout time [0..65535] in ms when receiving a character from the serial interface with the function get_ser().
put_ser_timeout:	Timeout Time [0..65535] in ms when sending a character from the serial interface with the function put_ser(). **Caution! This time is only active if the Hardware-Handshake is switched on.**
Return Value:	0 = OK, -1 = Error

With the function **Ser_Init()**, the serial interfaces are initialized with the transmitted Baud rate, the transmitted Handshake, 8 data bits, a stop bit and no parity. Furthermore the timeout times for the *get_ser()* and *put_ser()* functions are set.
A character is received using an interrupt routine, which stores the character in a receipt ring buffer (*In_Buffer*). This buffer can hold up to 250 8-bit characters. The buffer can be accessed with the functions *Init_Buffer(), Put_Buffer(), Get_Buffer()* and *Buffer_Status().*
In the **Ser_init()** function the ring buffer is constructed and cleared and the receive interrupt *(SORIC)* is initialized with interrupt level = 1 and interrupt group level = 1.

Caution!
Before using a function from the *serial.lib* for the first time, the **Ser_Init()** must be called once. The settings: 8 data bits, 1 stop bit and no parity can not be changed with this function.

An input ring buffer identified as *In_Buffer* is generated automatically. Therefore, when using the expanded ring buffer functions described subsequently in combination with the serial interface you will need to enter *In_Buffer* as an identification for the function parameter *Queue*.

In most cases the simple functions for using the serial interface are sufficient:
get_ser(); – get character
put_ser(); – write character
Char_Present_TimeOut(); – wait for character
Read_Error_Code_Serial(); – error handling

The error variable ERROR_CODE_SERIAL (‘unsigned char’ type) is used for error handling while the serial interface is operating. In order to be able to use the serial interface, you have to define this variable **global** in your main program.

6.3.7.2 get_ser

Function: Performs a query to see if a character is present in the receipt ring buffer (*In_Buffer*) within a specified timeout time and then gets the character.

BYTE get_ser (void)

Return Value:	(0 & ERROR_CODE_SERIAL = 0x02) = Timeout
	(0...255) = character from In_Buffer (serial interface)

The function **get_ser()** waits a predefined amount of time (see *get_ser_timeout*) for a character and then gets this character.
If there is already a character present in the buffer it will be read out immediately, otherwise it will wait the set wait time for the arrival of a character.
If the timeout time is exceeded, the Bit_1 (0000 0010) will be set in the error variable ERROR_CODE_SERIAL and the value 0x00 will be returned.
The timeout time can be set with the initialization of the *Ser_init()* or later with the *Set_Get_Ser_Timout()* function.

6.3.7.3 put_ser

Function: sends a character over the serial interface

void put_ser (BYTE send_char)

send_char:	Character that is to be sent over the serial interface.

The function *put_ser()* sends a character over the serial interface with the parameters set in *Ser_Init()*.
If the Hardware-Handshake was switched on in *Ser_Init()* then the character will only be sent if CTS=0. If CTS=1 is maintained during the entire timeout time (put_ser_timeout), then the character is not sent and the Bit 0 (0000 0001) is set in the error variable ERROR_CODE_SERIAL.
The timeout time *put_ser_timeout* can be set with the *Ser_init()* function.

6.3.7.4 read_ser

Function: Reads a character which is present in the receipt ring buffer (*In_Buffer*). The position of the read pointer is not changed by this operation which means that the character is not removed from the buffer.

BYTE read_ser (void)

Return Value:	(0...255) = character from In_Buffer (serial interface)

The function **read_ser()** gets the character from the receipt ring buffer. The character is not removed from the buffer (the position of the read pointer is not changed). Thus, this character can be read several times.
This function does not check the buffer status. This means, that – if the buffer is empty – the returned character is undefined. Use *Buffer_Status()* to determine, whether there is a valid character in the *In_Buffer*.

6.3.7.5 Char_Present_TimeOut

Function: Queries whether a character is present in the receipt ring buffer (In_Buffer) during the timeout time.

BYTE Char_Present_TimeOut (BYTE timeout)

Timeout:	Timeout time [0..255] for receiving a character from the serial interface.
Return Value:	0 = there is at least one character in *(In_Buffer)* 1 = no character was received during the timeout time

This function can be used to check whether a character was received and stored in the ring buffer of the serial interface.
If no character is present in the buffer, the routine will wait the alloted time defined by *timeout* for the entrance of a character.
If the timeout time is exceeded, the bit 1 (0000 0010) gets set in the error variable ERROR_CODE_SERIAL and the value 0x01 is returned.
This function can be used if characters are expected from the serial interface following a predefined time table (protocol).

6.3.7.6 Read_Error_Code_Serial

Function: Reads the ERROR_CODE_SERIAL variables.

BYTE Read_Error_Code_Serial (void)

Return Value:	0x01 = Error when sending a character (only possible if the Hardware-Handshake is active) 0x02 = Error when receiving a character

The errors that occurred in the functions *Char_Present_TimeOut()*, *get_ser()* and *put_ser()* are stored in the ERROR_CODE_SERIAL variable. This information can be read with the function *Read_Error_Code_Serial()*.

The errors are coded bit by bit, which means that combinations of errors can occur as well.
The value of ERROR_CODE_SERIAL is not influenced by a read. An error code must be reset explictly by a write access to ERROR_CODE_SERIAL.

6.3.7.7 Set_Get_Ser_Timeout

Function: Overwrites the *Get_Ser_Timout* time set in the *Ser_Init()* function.

void Set_Get_Ser_Timeout (WORD get_ser_timeout)

get_ser_timeout:	Time_Out time [0..65535] in ms upon receipt of a character from the serial interface with the function get_ser().

With the function **Set_Get_Ser_Timeout()** the maximum time in which the *get_ser()* function will attempt to get a character from the serial interface is determined.
This function is used to change the timeout time while the program is running, without performing a complete initialization with the *Ser_Init()* function.

6.3.7.8 Buffer_Status

Function: Reads the status of the specified ring buffer.

BYTE Buffer_Status (struct Buffer *Queue)

*Queue:	Pointer to a structure that is a buffer
Return Value:	0 = buffer is empty 1 = buffer is partially filled 2 = buffer is at least half full 3 = buffer is full 4 = buffer overflow

The function **Buffer_Status()** is used to ascertain whether a character is present in the buffer in question. It can also determine how full the buffer is.

Hint:
To test the status of the serial interface's input buffer, give the designation *In_Buffer* for *Queue*:

```
status = Buffer_Status(&In_Buffer);
```

6.3.7.9 Init_Buffer

Function: Initializes a struct Buffer type memory area, for example the Receipt Ring Buffer.

void Init_Buffer (struct Buffer *Queue)

*Queue:	Pointer to a buffer with the following structure: - unsigned char queue[BUFFER_SIZE]; = FIFO Buffer - unsigned char write; = Position of the write pointer - unsigned char read; = Position of the read pointer - unsigned char handle; = Status of the FIFO Buffer

With the function **Init_Buffer(),** the buffer elements *write*, *read* and *handle* are set to ZERO. That means that the buffer is cleared (erased). These elements and the following function can be used for definition and management of any desired Ring Buffer structures.
The functions described above for managing the serial interface carry out the buffer management automatically. Therefore in most cases the user need not get involved.

6.3.7.10 Get_Buffer

Function: Reads a character from the buffer.

BYTE Get_Buffer (struct Buffer *Queue)

*Queue:	Pointer to a Buffer structure (see *serial.h*).
Return Value:	Characters in the buffer at the current position of the read pointer.

With the function *Get_Buffer(),* a character is read from the buffer from the position: *Queue->queue[Queue->read]*. The buffer is managed as a ring buffer and the buffer level can be queried with the function *Buffer_Status()*.

Caution!
The *Get_Buffer()* function should only be called if there is a character present in the ring buffer. This can be checked with the function *Buffer_Status()*. If you try to use the *Get_Buffer()* function with no character in the ring buffer, a 0x00 will be returned.

Hint:
To get a character from the serial interface's input buffer, give the indication *In_Buffer* for *Queue*:

```
Character = GetBuffer(&In_Buffer);
```

It is easier to use the function *get_ser(),* which has the same functionality in most cases.

6.3.7.11 Put_Buffer

Function: Store a character in the specified buffer.

void Put_Buffer (struct Buffer *Queue, BYTE contents)

*Queue:	Pointer to a Buffer structure (see serial.h):
Contents:	Character that is to be stored in the ring buffer.

With the function **Put_Buffer()** a character is stored in the buffer at the position: *Queue->queue[Queue->write]*. The buffer is managed as a ring buffer and the buffer level can be queried with the function *Buffer_Status()*.
The receipt ring buffer (*In_Buffer*) defined in *Serial.h* is written to automatically from the Interrupt-Routine.

6.3.8 InOut.lib – Driver for Inputs and Outputs

The *In_Out.lib* contains all functions for initialization and port pin settings of the 80C156 microcontroller, which are assigned to the optically isolated I/O ports described in *section 3.1.2, „Serial Interface“*.

Caution!
Before using a function from the *In_Out.lib* for the first time, the function **IO_Init()** must be called one time. Furthermore *In_Out.h* must be included in your project.

The following functions are made available in the *In_Out.lib*:

- IO_Init()
- Read_IO()
- Write_IO()

In addition, bit variables are made available in the *In_Out.h* that can be used to directly control the individual ports.

Variable Name	Label on Connector	Description
IN_0	I/O1	INPUT 0
IN_1	I/O2	INPUT 1
IN_2	I/O3	INPUT 2
IN_3	I/O4	INPUT 3
OUT_0	I/O5	OUTPUT 0
OUT_1	I/O6	OUTPUT 1
OUT_2	I/O7	OUTPUT 2
OUT_3	I/O8	OUTPUT 3

6.3.8.1 IO_Init

Function: Initializing the optically isolated I/O ports.

void IO_Init (void)

This function initializes the microcontroller pins that are necessary for the control of the optically isolated I/O ports.

Caution!
Before using a function from the *In_Out.lib* for the first time, the function **IO_Init()** must be called one time.

6.3.8.2 Read_IO

Function: Reads the port pins I/O_1 through I/O_4.

BYTE Read_IO (void)

Return Value:	Input port status as a hexidecimal value, binary encoded. Format of the return value:

Bit Priority *nn*								
	-	-	-	-	I/O 4	I/O 3	I/O 2	I/O 1
nn Binary	-	-	-	-	Bit 3	Bit 2	Bit 1	Bit 0
nn Decimal	-	-	-	-	8	4	2	1
Bit set = Level present at Input *m*								

Example: nn = Read_IO();
nn = 06_{HEX} = 06_{DEC} = $0000\ 0110_{BIN}$ = 4 + 2 ⇨ High-level present at I/O3 and I/O2!

With this function the current status of the optically isolated I/O inputs can be queried (*see section 3.1.4, „Optically Isolated I/O Ports“*).

Hint:
In a special configuration I/O_5 through I/O_8 can be used as inputs. In this case Bit_4 through Bit_7 in the return value contain the additional input information.

6.3.8.3 Write_IO

Function: Control of port pins I/O_5 through I/O_8.

void Write_IO (BYTE out_byte)

out_byte: Status of the optical isolator's transistors at the outputs as a hexidecimal value, binary encoded. Format of the transmission value:

Bit Priority *out_byte*				
	I/O8	I/O 7	I/O 6	I/O 5
out_byte **binary**	Bit 3	Bit 2	Bit 1	Bit 0
out_byte **decimal**	8	4	2	1
Bit value = 1 = Opto-coupler's transistor in Hi-Z state Bit value = 0 = Opto-coupler's transistor conductive				

*Example:*Write_IO(out_byte);
Out_byte= E_{HEX} = 14_{DEZ} = 1110_{BIN} = ⇨ Transistor at I/O8, I/O7 and I/O6 on (conductive), Transistor at I/O5 off (Hi-Z-state).

With this function the optically isolated I/O outputs I/O_5 through I/O_8 *(see section 3.1.4, „Optically Isolated I/O Ports*") can be connected. This can be useful for controlling external circuitry.

Hint:
On power-up, the transistors of the optically isolated I/O outputs are in off-state = Hi-Z-state (corresponds to *oub_byte* = F_{HEX}).
If the transistors of the outputs are turned on (conductive), then the output pin is connected to the I/O plug's isolated Ground connectors GND.

6.3.9 I2C.lib – Driver for the I2C Devices

The *I2C.lib* contains functions for working with the I²C-Periphery on the board (video processor, EEPROM, RTC), as well as I²C devices that are connected over the OPTION-PORT (*see section 3.1.5*).

The I²C protocol is emulated using software with the microcontroller’s port pins P3.3 (SDA) and P3.4 (SCL).
The definition BYTE and BCD8 stands for „unsigned char“ and the definition WORD and BCD16 for „unsigned int“.

Caution!
Before using a function from the *I2C.lib* for the first time, the function **I2C_Init()** must be called. In addition the *I2C_Bus.h* must be included in your project.

The following functions are made available in the *I2C.lib*:

▪ I2CInit ()	I2C_Bus.h
▪ I2CWrite ()	I2C_Bus.h
▪ I2CRead ()	I2C_Bus.h
▪ BCD2INT ()	Misc.h
▪ INT2BCD ()	Misc.h
▪ RTC_Test ()	RTC_8564.h
▪ RTCSetTime ()	RTC_8564.h
▪ RTCGetTime ()	RTC_8564.h
▪ RTCSetAlarm ()	RTC_8564.h
▪ RTCGetAlarm ()	RTC_8564.h
▪ RTCGetAlarmStatus ()	RTC_8564.h

The functions of the *I2C.lib* are logically assigned to different header files (*I2C_Bus.h, I2C_Hard.h, Misc.h* and *RTC_8564.h*). A few functions in the *I2C_Bus.h* and in the *I2C_Hard.h* are low-level functions of the I²C protocol and are not described in subsequent sections.

6.3.9.1 I2C_Init

Function: Initialization of the I²C Interface (I2C_Bus.h)

void I2C_Init (void)

The I²C interface is initialized for Master/Slave operation with the function **I2C_Init(),** whereby the C165 microcontroller is the Master and all devices connected to the I²C port (SCL, SDA) are considered Slaves.

Caution!

- Before using a function from the I2C.lib, the **I2C_Init()** function must be called once.
- Be sure when connecting external I²C devices that no address conflicts result (*see section 3.1.5*), otherwise the devices will not function correctly.

6.3.9.2 I2CWrite

Function: Writes one or more Bytes to a selected device that is connected to the I²C bus. (I2C_Bus.h)

BYTE I2CWrite (BYTE *SourcePtr, BYTE DeviceID,
BYTE DestAddr, WORD Size)

*SourcePtr:	Address of a memory area that is to be written byte-wise to the I²C device.
DeviceID:	8-bit Device-ID of the I²C device.
DestAddr:	8-bit target address in the I²C device.
Size:	Number of Bytes to be written to the I²C device.
Return Value:	0x00 = Write OK 0xFE = Write not successful 0xFF = No I²C device found

With the function **I2Cwrite()** memory areas ranging from 1 byte to 65535 Bytes can be written from the microcontroller to the I²C device. The number of bytes determined in *Size* are written from the memory area of the **SourcePtr* to the memory area *DestAddr* of the I²C device specified in DeviceID.

Caution!
Never execute two write accesses in immediate succession, instead allow a delay of approximately 3 ms to occur.
If char, in or long variables are written to an I²C device, be aware that the function expects a BYTE pointer.

6.3.9.3 I2CRead

Function: Reads one or more Bytes from a selected device, which is connected to the I²C bus. (I2C_Bus.h)

```
BYTE I2CRead (BYTE *DestPtr, BYTE DeviceID,
              BYTE SourceAddr, WORD Size)
```

*DestPtr:	Address of a memory area to be written byte-wise with the contents of the I²C device
DeviceID:	8-bit Device-ID of the I²C device
SourceAddr:	8-bit read address of the I²C device
Size:	Number of Bytes to be read from the I²C device
Return Value:	0x00 = Read OK 0xFE = No I²C device found

With the function **I2CRead()** memory areas ranging from 1 Byte to 65545 Bytes can be read from the I²C device to the microcontroller memory.
The number of bytes determined in *Size* are written from the memory area of the **SourceAddr* to the memory area **DestPtr* of the I²C device specified in DeviceID.

Caution!
If char, int or long variables are to be read from an I²C device, be aware that the function expects a BYTE pointer.

6.3.9.4 BCD2INT

Function: Conversion of a BCD value to an integer number (Misc.h)

int BCD2INT(BCD16 BCDValue)

BCDValue:	16-bit (unsigned int) BCD value to be converted.
Return Value:	Converted BCD value as an integer number

With the function **BCD2INT()** 16-bit BCD values can be converted to an integer number. This is necessary to prepare values in BCD format for display or mathematical operations.

Example RTC (Realtime-Clock):

BCD value in RTC for seconds		**Corresponds to Hex Value**	**Corresponds to decimal value**	**Decimal value after conversion BCD2INT**
5	5	0x55 hex	85 dec	55 dec
10's column	1's column			

The values in the RTC are present in BCD format. Therefore for example the value '55 seconds' is stored as $0x55_{hex}$ or $85_{dec.}$ If the seconds are required as decimal values, the $0x55_{hex}$ or 85_{dec} must be converted to 55_{dec} with the BCD2INT function.

6.3.9.5 INT2BCD

Function: Conversion of an integer number to a BCD value. (Misc.h)

BCD16 INT2BCD(int INTValue)

INTValue:	16-bit integer value to be converted.
Return Value:	Converted integer number as (unsigned int) BCD value

With the function **INT2BCD()** integer numbers are converted to a 16-bit BCD value. This is necessary since some devices expect numbers in BCD format.

Example RTC:

Decimal value in seconds	**corresponds to hex value**	**Hex value after conversion INT2BCD**	**BCD values in RTC for seconds**	
55 $_{dec}$	0x37 $_{hex}$	0x55 $_{hex}$	5	5
			10’s column	1’s column

The values in the RTC are expected in BCD format. Now the value 55_{dec} , which corresponds to 55 seconds is to be stored in the RTC. 55_{dec} corresponds to $0x37_{hex}$, therefore the 55 $_{dec}$ must be converted to 0x55 $_{hex}$ with the function INT2BCD().

6.3.9.6 RTC_Test

Function: Tests whether the RTC (Real-Time Clock) is present and ready for operation. (RTC_8564.h)

BYTE RTC_Test (BYTE DeviceID)

DeviceID:	8-bit DeviceID of the I²C device (RTC 8564).
Return Value:	0x00 = Read OK 0xFF = No I²C device found

The function **RTC_Test()** performs a read access to the Control_1 register of the RTC. If after 3 seconds no valid read access could be carried out, the function will be aborted with an error code.

Caution!

- Since the RTC is only ready for use approximately 3 seconds after start-up, this function should be carried out before the first access to the RTC.
- The RTC's device ID can be found in the technical data (*section 5.3*). The symbol *RTC8564_ID* which is predefined in the header file can also be used.

6.3.9.7 RTCSetTime

Function: Sets the date and time of the RTC (RTC_8564.h)

BYTE RTCSetTime (REAL_TIME *RealTime, BYTE DeviceID)

*RealTime:	Pointer to a REAL_TIME structure:

```
{ BCD8  Second;
  BCD8  Minute;
  BCD8  Hour;
  BCD8  Day;
  BCD8  Weekday;  // day of week (sun=0 to sat=6)
  BCD8  Month;
  BCD16 Year; }
```

DeviceID:	8-bit Device-ID of the I²C-device (RTC 8564).
Return Value:	0x00 = success 0xFE = Write not successful

With the function **RTCSetTime()**, the date and time data of a structure (from the address of the pointer *RealTime) is written to the RTC.

Example:

```
REAL_TIME time;
time.Hour      = 0x23;
time.Minute    = 0x58;
time.Second    = 0x55;.
time.Weekday   = THURSDAY;
time.Day       = 0x31;
time.Month     = 0x12;
time.Year      = 0x2099;
RTCSetTime  (&Time, RTC8564_ID);
```

sets time = 23:58:55 and date = Thursday, 12/31/2099.

Caution!
The weekdays are sorted numerically, beginning with Sunday:

SUNDAY	= 0
MONDAY	= 1
TUESDAY	= 2
WEDNESDAY	= 3
THURSDAY	= 4
FRIDAY	= 5
SATURDAY	= 6

6.3.9.8 RTCGetTime

Function: Reads date and time from the RTC (RTC_8564.h)

BYTE RTCGetTime (REAL_TIME *RealTime, BYTE DeviceID)

*RealTime:	Pointer to a REAL_TIME structure:
	{ BCD8 Second;
	BCD8 Minute;
	BCD8 Hour;
	BCD8 Day;
	BCD8 Weekday; // day of week (sun=0 to sat=6)
	BCD8 Month;
	BCD16 Year; }
DeviceID:	8-bit DeviceID of the I²C device (RTC 8564)
Return value:	0x00 = success
	0xFF = Read not successful

With the function **RTCGetTime()** the current date and time data is read from the RTC and stored in the structure (at the address of the **RealTime* pointer).

Example: REAL_TIME time;
RTCGetTime (&Time, RTC8564_ID);

If the variables have the following values:

Time.Hour	= 0x23,
Time.Minute	= 0x58,
Time.Second	= 0x55,.
Time.Weekday	= THURSDAY,
Time.Day	= 0x31,
Time.Month	= 0x12,
Time.Year	= 0x2099,

This corresponds to the time 23:58:55 and the date – Thursday 12/31/2099.

6.3.9.9 RTCSetAlarm

Function: sets the alarm day and alarm time for the RTC. (RTC_8564.h)

**BYTE RTCSetAlarm (ALARM_TIME *AlarmTime,
BYTE EnableExtInt, BYTE DeviceID)**

*AlarmTime:	Pointer to ALARM_TIME structure:
	{ BYTE Minute_Ignore;
	BCD8 Minute;
	BYTE Hour_Ignore;
	BCD8 Hour;
	BYTE Day_Ignore;
	BCD8 Day;
	BYTE Weekday_Ignore;
	BCD8 Weekday; // sun = 0 to sat = 6
	}

EnableExtInt:	Activate or deactivate the external interrupt of the RTC at port pin 10 (/INT). 0 = deactivated, 1= activated
DeviceID:	8-bit Device-ID of the I²C device (RTC 8564).
Return Value:	0x00 = success 0xFE = Write not successful 0xFF = Read not successful

With the function **RTCSetAlarm()** the data of a structure (from the address of the pointer **AlarmTime*) for alarm day and alarm time is written to the RTC. The variables Minute_Ignore, Hour_Ignore, Day_Ignore and Weekday_Ignore can influence the validity of the individual time entries.

Example:

- Day_Ingore = 0: Alarm is generated on set Alarm Day
- Day_Ignore = 1: Alarm is generated independently of the set Alarm Day.

Furthermore, the RTC Interrupt Pin 10 (/INT) can be activated or deactivated with the variable *EnableExtInt*. If the function is activated, then when the Timer-Flag (*TF, see Control2-Register in RTC8564*) or the Alarm-Flag (*AF, see Control2-Register in RTC8564*) is set an interrupt is generated.

Example.:

```
ALARM_TIME AlarmTime;
AlarmTime.Hour              = 0x23;
AlarmTime.Hour_Ignore       = 0;
AlarmTime.Minute            = 0x58;
AlarmTime.Minute_Ignore     = 0;
AlarmTime.Second            = 0x55;
AlarmTime.Second_Ignore     = 0;
AlarmTime.Weekday           = THURSDAY;
AlarmTime.Weekday_Ignore    = 0;
AlarmTime.Day               = 0x31;
AlarmTime.Day_Ignore        = 0;
RTCSetAlarmTime  (&AlarmTime,0, RTC8564_ID);
```

Sets the Alarm Time to 23:58:55 on Thursday the 31st. All Alarm Time settings are valid and the external interrupt is not activated.

Caution!
Jumper J7 must be closed on the grabbMODUL so that the RTC can generate an interrupt (*see section 5.2*).

6.3.9.10 RTCGetAlarm

Function: Reads Alarm Day and Alarm Time from the RTC. (RTC_8564.h)

**BYTE RTCGetAlarm (ALARM_TIME *AlarmTime,
BYTE DeviceID)**

*AlarmTime:	Pointer to an ALARM_TIME structure.

```
{  BYTE  Minute_Ignore;
   BCD8  Minute;
   BYTE  Hour_Ignore;
   BCD8  Hour;
   BYTE  Day_Ignore;
   BCD8  Day;
   BYTE  Weekday_Ignore;
   BCD8  Weekday;                  // sun = 0 to sat = 6
}
```

DeviceID:	8-bit DeviceID of the I²C device (RTC 8564).
Return Value:	0x00 = success 0xFF = Read not successful

With the function **RTCGetAlarm()** the current data for Alarm Day and Alarm Time is read from the RTC and stored in the structure (at the address of the pointer **AlarmTime*).
The variables Minute_Ignore, Hour_Ignore, Day_Ignore and Weekday_Ignore contain information about the validity of the individual time inputs.

Example:

- Day_Ignore = 0,	Alarm is generated on the set Alarm Day
- Day_Ignore = 1,	Alarm is generated independently of the set Alarm Day.

```
Example: ALARM_TIME AlarmTime;
         RTCGetAlarmTime (&AlarmTime, RTC8564_ID);
```

If the variables have the following values:

```
AlarmTime.Hour              = 0x23,
AlarmTime.Hour_Ignore       = 0,
AlarmTime.Minute            = 0x58,
AlarmTime.Minute_Ignore     = 0,
AlarmTime.Second            = 0x55,
AlarmTime.Second_Ignore     = 0,
AlarmTime.Weekday           = THURSDAY,
AlarmTime.Weekday_Ignore    = 0,
AlarmTime.Day               = 0x31,
AlarmTime.Day_Ignore        = 0,
```

An alarm will be generated on Thursday the 31st at 23:58:55.

6.3.9.11 RTCGetAlarmStatus

Function: With this function the current Alarm Flag status can be read and reset in the RTC (RTC_8564.h)

BYTE RTCGetAlarmStatus (BYTE ResetAlarm,
BYTE DeviceID)

ResetAlarm:	0 = AF Flag is not reset after the read
	1 = AF Flag is reset after the read
DeviceID:	8-bit DeviceID of the I²C device (RTC 8564).
Return Value:	0x00 = AF-Flag is not reset
	0x01 = AF-Flag is reset
	0xFE = Write not successful
	0xFF = Read not successful

With the function **RTCGetAlarmStatus()** the status of the Alarm-Flag (AF, see Control2-Register of the RTC8564) is read and the result is returned. This function can also reset the Alarm-Flag after the read.

6.4 Using PHYTEC Firmware

6.4.1 LOCAL_COM: Firmware for Local Image Transferrence

With the firmware *LOCAL_COM* PHYTEC offers you a fully-functioning module software that can be used to control the grabbMODUL-4 with a host PC via the serial interface and transfer image data from the module (*Figure 55*).
Furthermore, image captures can be generated over the optically isolated inputs by external events (up to four inputs can support four cameras). The time of the generated image capture can be set by a time stamp.

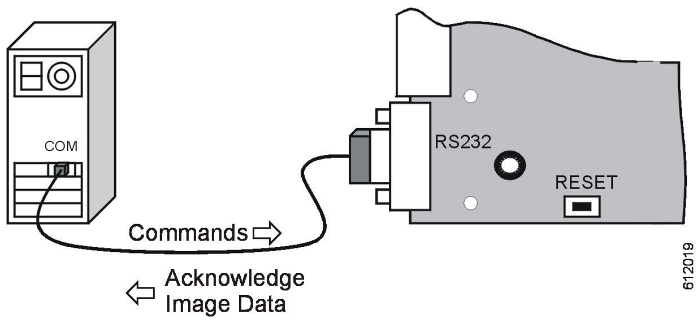


Figure 55: Device Configuration for LOCAL_COM

Please Note:

- The firmware LOCAL_COM has to be installed on the grabbMODUL-4. To do this you will have to program the file *local_com_Vxx.h86* into the module's Flash memory with the help of PHYTEC FlashTools. (*xx* represents the current version number.). Upon delivery, a current version of „LOCAL_COM“ is already loaded in the Flash memory of the grabbMODUL-4.
- The grabbMODUL-4 and the Host PC have to be connected in the application environment with a *Nullmodem* cable. With a Nullmodem cable, the signal lines cross internally. If possible use a pre-fabricated Nullmodem cable, in order to prevent errors. *Information on construction of a Nullmodem cable can be found on the FAQ page of the PHYTEC website.*

- You are required to write your own application software for the host PC that will be used to control the grabbMODUL-4, capture image data and transmit it over the serial interface.
  This software can be structured however you like. In the following section there is a description of the commands that can be sent to the module over the serial interface and the format of the data transfer.

- After being downloaded to the grabbMODUL-4 the software can be tested with a terminal program (e.g. HyperTerminal) or the Windows software „***PC_Vxx.exe***“. The test procedure is described in *section 2.4, „Installation of the PHYTEC FlashTools and Downloading a Program Code*“.

6.4.2 Hints about Firmware Functions

The software on the grabbMODUL-4 initializes the serial interface with 115200 Baud, 8 Data bits, no parity, one stop bit and no protocol. Afterwards the software waits for a character from the serial interface to execute the corresponding function. If a valid protocol sequence is introduced by the host PC (e.g.: S), the module expects the following characters. If the individual characters are not sent by the host within 3 seconds (preset), then the module will proceed as if there was interference on the serial interface. If this occurs the module will generate an error message (e.g. g,S,1#), set the error bit (0x08 „received character failed“) in the error status register and wait for a new protocol sequence that is valid.

If a protocol sequence has been completely received, the grabbMODUL-4 answers with a delay of approximately 10…500ms. Larger delays can occur with the commands „G“ (up to 3 s) and „V“ (up to 5 s).
If a level-change (rising or falling edge) is recognized on one of the input pins while waiting for a protocol sequence then the software will grab an image from the corresponding channel (Input 1 = Channel 1, ...). The image size correponds to the values set by the parameter „P“ (default: 180 x 144).

6.4.3 Overview of Commands and Command Structures

Commands and responses are transmitted as ASCII codes. A command consists of an introductory command letter, followed by the required parameters. A comma „ , “ is sent as a character separation between command letters and parameters. Every command ends with a number sign „#“.
Responses begin with the response letter Q/q. An error number is transferred as a parameter. The error number 0 means „No Error“. The response ends with a number sign „#“.

The firmware contains the following commands:
- **`C`** - Request input status
- **`E`** - Reserved
- **`F`** - Request error status
- **`G`** - Grab camera image from Channel *k*
- **`I`** - Request image data
- **`J`** - Reserved
- **`L`** - Reserved
- **`O`** - Set output status
- **`P`** - Set image size and scaling
- **`R`** - Reset unit
- **`S`** - Query software ID
- **`T`** - Read/Reset time stamp
- **`V`** - Request camera status
- **`X`** - Reserved
- **`Z`** - Set receive timeout for the interface
- **`0x0A`** - Reserved
- **`0x0D`** - Reserved

6.4.4 Query Software ID

Command Code:	`S#`
Parameter:	None
Example:	`S#`
Response:	`q,S,0#s,`*xx*`,`*yy*`#` = Confirmation, Version code xx.yy e.g. V1.02= 01.02
	`q,S,1#` = Error

6.4.5 Reset Unit

Command Code:	`R,`*x*`#`
Parameter:	`1` = Carry out software reset all others: abort command
Example:	`R,1#`
Response:	`q,R,0#` = Confirmation, return to default state `q,R,1#` = Parameter error, no reset
Note:	A software reset will put the grabbMODUL-4 back in its default state. All new value settings will be lost. After the Reset these values will have to be reentered.

6.4.6 Request Camera Status

Command Code:	`V#`
Parameter:	None
Beispiel:	`V#`
Response:	`q,V,0#v,`*nn*`#` = confirmation, status is sent: *nn* = Camera status as hexidecimal value, binary encryption `q,V,1#` = Parameter error
Note:	Between `q,V,0#` and `v,nn#` there can be a processing time of up to 5 sec.

Format of Response:

Bit Priority *nn*								
	-	-	-	-	I/O 4	I/O 3	I/O 2	I/O 1
nn Binary	-	-	-	-	Bit 3	Bit 2	Bit 1	Bit 0
nn Decimal	-	-	-	-	8	4	2	1
Bit Set = Video signal present on channel *m*								

Example:	Returned response = `q,V,0#v,0A#` *nn* = $0A_{HEX}$ = 10_{DEZ} = $0000\ 1010_{BIN}$ = 8 + 2 ⇨ Signal present on channels 4 and 2.

6.4.7 Grab Camera Image from Camera K

Command Code:	`G,`*kk#*
Parameter:	*kk* = Channel number, decimal 00...04
Beispiel:	`G,02#`
Quittung:	`q,G,0#` = Confirmation
	`q,G,1#` = Parameter error

Channel Number = 0: The current image is grabbed by the most recently set channel. Since the channel can also be set by an external event, a signal transition at input 2 (=Camera 2) would mean that the image would be grabbed again by channel 2.

Channel Number = 1..4: The current image will be grabbed by the given channel

Note: A processing time of up to 3 sec can lie between q,G, and 0#. In this time period the grabbMODUL-4 will be set to the correct input channel and the image will be digitized. The default image size is 180x144 pixels. The image size can be modified with the command P.

6.4.8 Requesting Image Data

Command code: `I,`*ooooo*`,`*ss*`,`*nnnn*`,`*m*`#`

Parameter: *ooooo* = Offset in image memory, hexadecimal, 5-position

ss = Size of the data block, 2-digit hex value

nnnn = Number of data blocks, hex

m = Mode:

-- Grayscale images --
1 = Send Low-Nibbles
2 = Send High-Nibbles
3 = Send all bytes
-- Color images --
4 = Send color data YCrCb 4:2:2
(**Note: In mode 4 only even numbered block sizes are possible!)**
5 = reserved
6 = reserved
7 = Only send CrCb color information
8 = reserved

Examples:

(a) Image in memory 180 x 144 pixels, transmitted as byte (256 Grayscale)
Use maximum block size:
`I, 00000, FF, 0066, 3#`
From the module side 101 blocks of 255 pixels each are sent and 1 block with 165 pixels is sent
(101 x 255 + 165 = 180 x 144)
Note: 102 = 0x66

(b) Image in memory 180 x 144 pixels, transmission of the upper 4-bits of each pixel (packed), in order to reduce the amount of data by half:
`I, 00000, FF, 0033, 2#`
From the module side 50 blocks of 255 bytes each are sent and 1 block with 210 pixels is sent
(50 x 255 + 210 = 180 x 144 : 2)
The grayscale resolution can be completed subsequently by sending the command
`I, 00000, FF, 0033, 1#.`

(c) Image in memory 180 x 144 pixels, transmitted as complete color image, use maximum block size:
`I, 00000, FE, 00CD, 4#`
From the module side 204 blocks of 254 bytes each are sent and 1 block with 24 pixels is sent
(204 x 254 +24 = 180 x 144 x 2)
Note: 204 = 0xCD

(d) Image in memory 180 x 144 pixels, transmission of the color information CrCb, use maximum block size:
I, 00000, FF, 0066, 7#
From the module side 101 blocks of 255 pixels each are sent and 1 block with 165 pixels
(101 x 255 +165 = 180 x 144)
Note: 102 = 0x66

Response:	q, I, 0#	= OK, transmission follows
	q, I, 0#	= Parameter error
	q, I, 0#	= invalid value range for a parameter

After the response „q, I, 0#“ the image data will be transmitted from the grabbMODUL-4 as described in *section 6.4.9, „Image Data Format“*. This function transfers the image data from the module’s video RAM over the serial interface. Since the data transfer can take a long time, there are various modes allowing flexible data handling.

Caution!
In mode 4 the block sizes (Parameter *ss*) have to be even!

Hints:

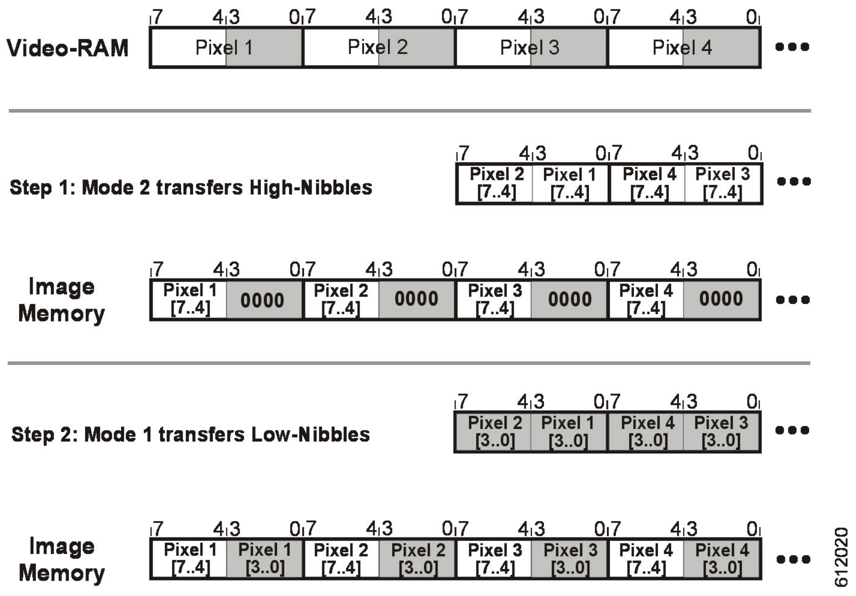
No image is captured with this function, instead the image in the memory is transmitted. To capture an image the command G must be used or an alarm signal has to be entered in the alarm inputs.

The image data is transmitted in the order it is arranged in the memory. This means that for resolutions greater than 288 lines, the two fields are transmitted separately one after the other (*see section 6.2.1*). This is advantageous because when data is being displayed on the screen, an image with half the line-resolution can be displayed quickly by displaying all received lines twice. When the second half-image is transferred it can be added. The original information contained in these lines is then overwritten.
For the observer the image appears to undergo progressive construction.
Details about the format of the transmitted image data can be found in section 6.4.9.

Mode 3 (standard mode) initiates the transmission of the grayscale images. The image data is transmitted line by line (from the top left to the bottom right).

To speed up the image construction, the Nibble modes (Mode 1 and 2) can be used. The function principle of the Nibble mode is based on another transmission of the brightness information. The brightness information for each pixel is transmitted roughly at first (16 gray levels). Since this reduces the information density to 4-bits per pixel, two pixels can be packed in on byte and the image transmission runs twice as fast. In a second run the other half of the brightness information can be transmitted in the same way, whereby the grayscale resolution of 8-bits (256 levels) is achieved again.

The entire transmission time corresponds to the duration of the normal grayscale transmission (Mode 3). The advantage of this is that a complete preview image is already available in half the time.



Video-RAM
Pixel 1
Pixel 2
Pixel 3
Pixel 4
Step 1: Mode 2 transfers High-Nibbles
Image Memory
0000
Step 2: Mode 1 transfers Low-Nibbles
[7..4]
[3..0]
612020

Figure 56: Data Transmission in Nibble Mode

There are two modes available for the transmission of color images. Mode 4 transmits a complete color image inYCrCb-4:2:2 format. The data is transmitted sequentially, i.e. Y_1, $Cb_{1/2}$, Y_2, $Cr_{1/2}$ etc. In order to be displayed the data has usually to be converted to RGB format.
Since two bytes are logically linked here, an even number of bytes must be requested for each block (which means that the parameter *ss* must be even).
Mode 7 only transmits the image's Cr/Cb color information. The color information is transmitted as follows: $Cb_{1/2}$, $Cr_{1/2}$, $Cb_{3/4}$, $Cr_{3/4}$ etc.

This mode was introduced to enable a faster image preview. A grayscale image can be transmitted for preview with mode 3 in the first step. In step 2 mode 7 can be used to get the color information and calculate it into the existing grayscale information, thereby effectively coloring the image.
Combinations of modes 2, 1 and 7 are also possible.

6.4.9 Image Data Format

Image data is transmitted in binary format. The transmission is initiated with the command sequence `i`. The image data is then transmitted in blocks according to the request. At the beginning of each block the actual block size is transmitted as a two digit Hexidecimal value in ASCII code. The specified number of data bytes then follow in binary form. Finally a check sum (binary) is sent over the binary data bytes. The check sum is supplemented to 00. A complete data block is thus constructed as follows:

i, ll$d_1d_2d_3d_4d_5$.....d_{ll}c [Response] ll$d_1d_2d_3d_4d_5$.....d_{ll}c [Response] ...

ll = Number of actual data bytes in a block (hex)
d_n = Data bytes (binary)
c = Check sum (binary)

The check sum „c“ is created for each block and transmitted at the end of the block. The check sum is calculated by adding all the data ($d_1+d_2+d_3+d_4+d_5+.....d_{ll}$) and then transmitting the difference of the result’s low byte to 0x0100.
Example: 2 databytes d1 = 120 and d2 = 160 are sent in one block.
The sum of d1 and d2 = 120 + 160 = 280 = 0000 0001 0001 1000b. The low-byte portion is 0001 1000b = 0x18 = 24. Thus, the checksum c = 0x100 - 0x18 = 0xE8 = 232.

The transmission ends when the specified number of blocks has been transmitted. If data outside the image data memory is requested then the send request is answered with an error message.

The unit waits for a response from the receiver after each block:

Response from the receiver (host PC)

`q,i,0#` = OK, the module sends the next data block
`q,i,1#` = Error / the transmission is aborted by the module

If no valid response is received, then the transmission is aborted. If after a certain timeout time (preset: 3 seconds) no response is received then the transmission is aborted.

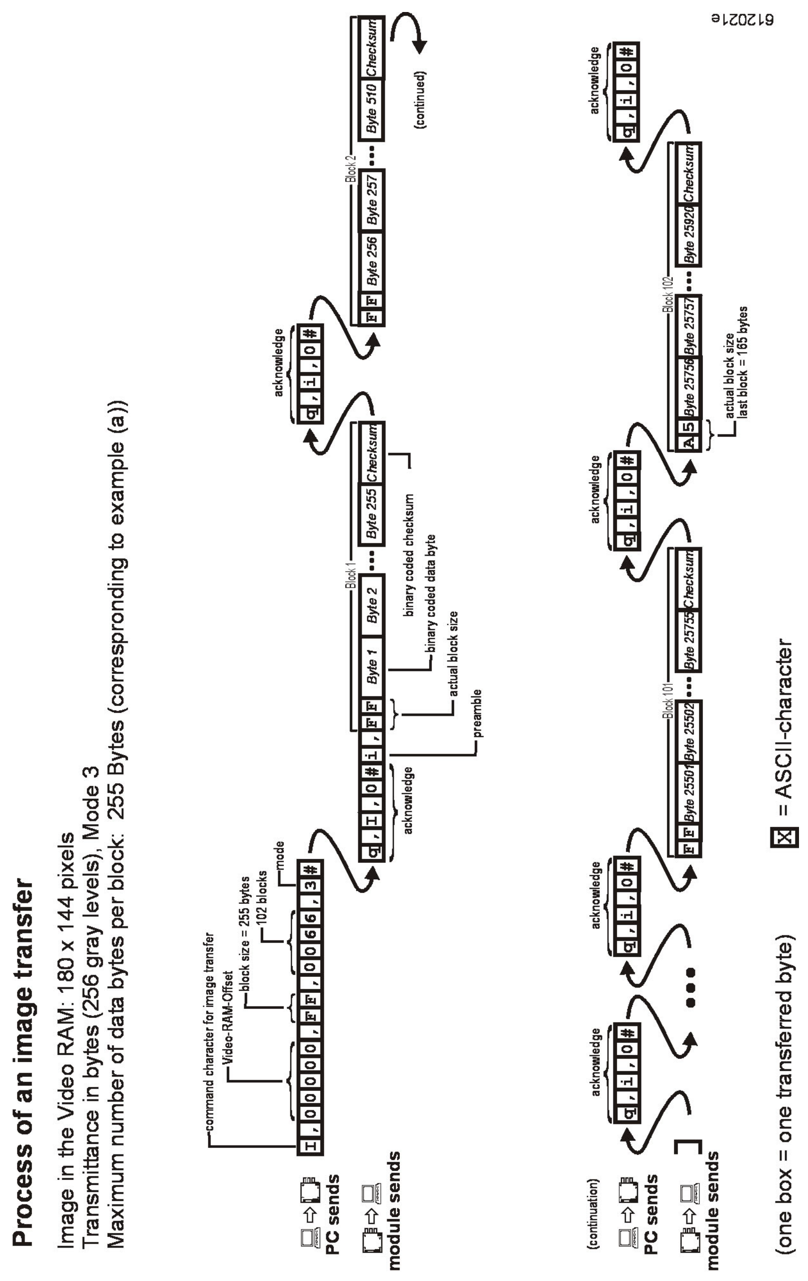


Figure 57: Sequence of an Image Transmission

Hint:
Since the data transmission is binary the interface cannot be configured to „Software Handshake“ (such as XON/XOFF). It is better if the buffer size is adapted to the data block size and the flow control is realized on a higher layer via the response transmission.

6.4.10 Setting Image Size and Scaling

Command Code: `P,`*hhh*`,`*vvv*`,`*xxx*`,`*yyy*`,`*ppp*`,`*lll*`,`*c*`,`*i*`#`

Parameter:	*hhh*	= Left start position of the image outline in video field (horizontal), 3-digit hex value
	vvv	= Upper start position of the image outline in the video field (vertical) 3-digit hex value
	xxx	= Size of the image outline in x-direction, 3-digit hex value
	yyy	= Size of the image outline in y-direction, 3-digit hex value
	ppp	= desired image size in the x-direction (Pixels per line), 3-digit hex value
	lll	= desired number of image lines (y-direction) 3-digit hex value
	c	= color system used, 1-digit hex value 0 = Reserved 1 = PAL 2 = Reserved
	i	= Full or half-image mode: 0 = Field 1 = Reserved 2 = Frame

Note: The parameters transmitted with „P“ (up to i) correspond to the parameters of the „Set_Image_Size“ function. The function and its parameters are described in detail in *section 6.3.6.25„Set_Image_Size“.* The parameter „i“ must be transmitted to inform the Grabber if a field (odd, preset) or a frame (odd and even field) should be grabbed.

Response:	`q,P,0#`	= OK
	`q,P,1#`	= Parameter Error
	`q,P,2#`	= Invalid value range for a parameter

Examples for image size settings:

(a) The image to be grabbed should have the following properties:
- No change of the aspect ratio
- 1:4 scale
- 180 x 144 pixel resolution
- Field resolution

The following parameters result:
Decimal: 0,0,180,288,192,288,1,0
Hex/Command:
```
P,000,000,0B4,120,0C0,120,1,0#
```

(b) The image to be grabbed should have the following properties:
- No change of the aspect ratio
- 1:2 scale
- 360 x 288 pixel resolution
- Field resolution

The following parameters result :
Decimal: 0,0,360,576,384,567,1,0
Hex/Command:
```
P,000,000,168,240,180,240,1,0#
```

(c) The image to be grabbed should have the following properties
- No side change of the aspect ratio
- 1:1 scale
- 768 x 576 pixel resolution (maximum resolution)
- Frame resolution

The following parameters result:
Decimal: 0,0,768,576,768,567,1,2
Hex/Command:
```
P,000,000,300,240,300,240,1,2#
```

6.4.11 Image Capture Triggered by an Alarm Input

An image capture can not only be triggered by the `G` command, but also by an event on one of the optically decoupled signal inputs (*see section 3.1.4)*
An event can be any signal transition, in other words a level change from a logical 0→1 or from a logical 1→0.

If input X is recognized as active, then the unit will digitize the video signal from video input X and then store the image in the image memory. Afterwards a status flag is set, which displays, which video channel sent the image. If multiple trigger signals are active at the same time, then the camera image triggered by the lowest priority channel will be stored.

Note:
This procedure ensures that the image memory retains an image of the last input channel for which the trigger signal has indicated a level change.

The status flag can be queried with the command `T`. The host PC can then decide whether it wants to get the stored image or not. The flag can be deleted by a command parameter.

Hint:
The status flag then receives a new value (channel number), if the user has explicitly used another channel to grab a new image with the command „`G`“.

A time stamp is generated with the image.
For the time stamp generation an internal 16-bit counter counts up approximately one second at a time. The counter begins to count when the status flag is set. The counter status can be transmitted to the PC. If the maximum counter level is exceeded then the maximum counter value is returned. This value is then to be interpreted as a „capture before more than x seconds“.
The counter is automatically stopped and returned to its rest state when the status flag is reset by the host PC.

6.4.12 Read/Reset Time Stamp

Command Code:	`T,`*r*`#`
Parameter:	r Reset: 0 = Only read time stamp 1 = Read time stamp and then reset
Example:	`T,0#`
Response:	`q,T,1#` = Parameter error `q,T,0#t,`*kk*`,`*zzzzz*`#` *kk* = triggering channel *zzzzz* = time in seconds since triggering, max. 65535 sec.
Example:	`q,T,0#t,03,01800#` = Input 3 triggered an image capture 30 minutes ago (1800 sec / 60 = 30 minutes)

An image capture can be generated by a level transition on an alarm input. The point in time when the alarm generation occurred and the triggering channel can be found out by reading the time stamp. For information on the layout of the alarm inputs, *refer to section 3.1.4*, „Optically Isolated I/O Ports“). The time stamp determines the time span that has elapsed since the image capture. The maximum time is approximately 18 hours.

Hint:
The triggering channel also receives a new value, if the user has explicitly used another channel to grab a new image with the command „`G`“.
The time display is not changed by a manually generated image capture.

6.4.13 Request Input Status

Command Code:	`C#`
Parameter:	none
Example:	`C#`
Response:	`q,C,0#c,`*nn*`#` = Confirmation, status is sent: *nn* = Input status as hexadecimal value, binary coded `q,C,1#` = Parameter error

Note:

Format of Response:

Bit Priority *nn*								
	-	-	-	-	I/O 4	I/O 3	I/O 2	I/O 1
***nn* Binary**	-	-	-	-	Bit 3	Bit 2	Bit 1	Bit 0
***nn* Decimal**	-	-	-	-	8	4	2	1
Bit set = High-Level present at Input *m*								

Example:
Returned Response = `q,C,0#c,06#`
nn = 06_{HEX} = 06_{DEZ} = 0000 0110_{BIN} = 4 + 2 ⇨ High-level present at I/O3 and I/O2

With this function the momentary status of the optically isolated I/O inputs *(see section 3.1.4, „Optically Isolated I/O Ports“*) can be queried. This can be used to test connected low-active alarm loops.

6.4.14 Set Output Status

Command Code:	`O,`*s*`#`
Parameter:	s = bit-mask für output ports, 1-digit hex value
Example:	`O,0#`
Response:	`q,O,0#` = OK
	`q,O,1#` = Parameter error
Example:	`q,O,0#` = all 4 transistor outputs of the module become conductive.
Note:	

Format of the bit mask „s":

Bit Priority *s*				
	I/O8	I/O 7	I/O 6	I/O 5
s binary	Bit 3	Bit 2	Bit 1	Bit 0
s decimal	8	4	2	1
Bit Value = 1 = Opto-coupler's transistor is in Hi-Z-state Bit Value = 0 = Opto-coupler's transistor is conductive				

Example:
Sent Parameter = `O,E#`
s = E_{HEX} = 14_{DEZ} = 1110_{BIN} = ⇨ Transistor at I/O8, I/O7 and I/O6 Hi-Z (isolated), Transistor at I/O5 conductive

With this function the optically isolated I/O outputs I/O5-I/O8 *(see section 3.1.4, „Optically Isolated I/O Ports*") can be switched. This can be used to control external circuitry.

Hint:
After power-up the optically-isolated I/O outputs are in Hi-Z – state (s = F_{HEX}).
If the output transistors are conductive, the output pin is connected to Ground (GND).

6.4.15 Request Error Status

Command Code:	`F#`
Parameter:	none
Example:	`F#`
Response:	`q,F,0#f,`*nn*`#` = Confirmation, status is sent:
	nn = Error status as hexadecimal value, binary coding

$0x01_{Hex}$ =	Initialization or setting of video parameters (BT829) failed
$0x02_{Hex}$ =	Start of grab procedure failed
$0x04_{Hex}$ =	Transmission of a character on the serial interface failed
$0x08_{Hex}$ =	Receipt of a character from the serial interface failed
$0x10_{Hex}$ =	Faulty answer from host during image transmission
$0x20_{Hex}$ =	Reserved
$0x40_{Hex}$ =	Reserved

`q,F,1#` = Parameter error

Note:

Format of Response:

	Bit Priority *nn*							
***nn* Binary**	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
***nn* decimal**	128	64	32	16	8	4	2	1
Bit Set = Error code present corresponding to error								

Example:

Returned Response = q, F, 0#f, 0A#

nn = $0A_{HEX}$ = 10_{DEZ} = $0000\ 1010_{BIN}$ = 8 + 2

⇨ Error $0x02_{Hex}$ and $0x08_{Hex}$ occured

With this function the internal variable „Error-Code“ can be read. With the help of the error number the cause of the module’s error condition can be determined.

$0x01_{Hex}$ = Initialization or configuration of video parameters (BT829) was faulty. In this case there was either an error in the I²C control on the module or a defect present in the video device BT829.

$0x02_{Hex}$ =	The start of the grab procedure was faulty. In this case the video control device on the module is defective.
$0x04_{Hex}$ =	An error occured when sending a character on the serial interface. This error appears during Handshake operation if the CTS signal was not retrieved by the host within a specified time (preset: 5 sec.). Check the CTS signal.
$0x08_{Hex}$ =	An error occured during receipt of a character from the serial interface. If the protocol expects the reception of characters (handshake or other requested characters), the module will wait a specified time (preset: 3 sec.) for the character. If no character is received within the set receipt-timeout time, then this error is set and a „0_{dec}" is received. In this case there is either interference in the serial connection or the response character was sent too late. Use the function „Z" (*see section 6.4.16, „Setting the Interface's Receive Timeoute"*) to set a new timeout time or check the serial connection.
$0x10_{Hex}$ =	Host's handshake for image transfer was faulty. This error is set if a faulty handshake was sent by the host as an answer to the image transfer request „`I`, request image data" (*see section 6.4.9, „Image Data Format*"). Check the response in the host to see if it is correct.
$0x20_{Hex}$ =	Reserved
$0x40_{Hex}$ =	Reserved

6.4.16 Setting the Interface's Receive Timeout

Command Code:	`Z,`*tt*`#`
Parameter:	*tt* = Timeout time in seconds for an individual character from the serial interface, 2-digit hex value. Maximum value range [0...0x40$_{Hex}$] = [0...64$_{Dec}$] seconds.
Example:	`Z,08#`
Response:	`q,Z,0#` = OK
	`q,Z,1#` = Parameter error

Note:

Format of the bitmask „tt":

	Bit Priority *tt*							
tt binary	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
tt decimal	-	64	32	16	8	4	2	1

Example:
Sent parameter = Z,0A#
tt =0A$_{HEX}$ = 10$_{DEc}$ = 0000 1010$_{BIN}$ = Timeout time for the receipt of a serial character is set to 10 seconds.

In its idle state the module waits for a character that is defined in the protocol (protocol preamble character). Then, according to the protocol, additional characters or responses will follow. Each of these characters or responses is expected within a specific time frame (timeout time). If these characters are not sent within the timeout time then the module will return to its idle state (waits for the protocol character).
By changing the timeout time you can determine the wait time of the module and also establish how quickly the module returns to its initial state after an interference in the data stream.

Hint:
It makes sense to set the timeout time relatively long if you are performing an initial test with a terminal (e.g. HyperTerminal) and want to enter the protocol commands and responses manually.

Document:	**grabbMODUL-4**
Dokument No.:	**L-612e_1, February 2005**

How would you improve this manual?

____

____

____

Did you find any mistakes in this manual? page

____

____

____

Submitted by:

customer no.:	______________________
name:	______________________
company:	______________________
address:	______________________
	______________________

return to:

PHYTEC Technologie Holding AG
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-33

Published by PHYTEC

© PHYTEC Messtechnik GmbH 2005

Ordering No. L-612e_1
Printed in Germany