Smart Cameras made in Germany

Vision ®

Components

The Smart Camera People

Programming Tutorial Basics

For VC Smart Cameras with Ti DSP

(VC4XXX VCSBC4XXX and VC2XXX)

Revision 2.2 June 2008 2007
Document name: Prog_Tut.pdf

Vision Components GmbH
Ottostr. 2
76275 Ettlingen
Telefon +49 (0)7243 2167-0
Fax +49 (0)7243 2167-11

Vision Components®
The Smart Camera People
since 1996

Foreword and Disclaimer

This documentation has been prepared with most possible care. However Vision Components GmbH does not take any liability for possible errors. In the interest of progress, Vision Components GmbH reserves the right to perform technical changes without further notice.
Please notify **support@vision-components.com** if you become aware of any errors in this manual or if a certain topic requires more detailed documentation.
This manual is intended for information of Vision Component's customers only. Any publication of this document or parts thereof requires written permission by Vision Components GmbH.

Trademarks

Code Composer Studio and TMS320C6000, Windows XP, Total Commander, Tera Term, Motorola are registered Trademarks. All trademarks are the property of their respective owners.

References

Since the VC4XXX smart camera family employs a TI processor, the programming environment and functions for the VC20XX cameras can be used for this camera.

Further References under "Support + Download" on www.vision-components.com**:**

„Support News" – for up to date information on VC Software and Documentation.

„Knowledge Base / FAQ" - searchable Database with latest software developments, frequently asked questions and demo programs.

"Download Areas" for all documentation and Software downloads – refer to the following table:

Description	Title on Website	Download Area
Schnellstart VC – Deutsche Version dieses Handbuches .	Schnellstart VC Smart Kameras	Registered User Area ▸Getting Started VC SDK TI
Getting Started VC Smart Cameras	Getting Started VC Smart Cameras with TI DSP	Registered User Area ▸Getting Started VC SDK TI
introduction à l'utilisation des caméras Vision Components	Démarrage rapide Smart Cameras Vision Components	Registered User Area ▸Getting Started VC SDK TI
Demo programs and sample code used in the Programming Tutorial	Tutorial_Code	Registered User Area ▸Getting Started VC SDK TI
VC4XXX Hardware Manual	VC4XXX Smart Cameras Hardware Documentation	Public Download Area ▸Hardware Documentation VC Smart Cameras
VCRT Operation System Functions Manual	VCRT 5.0 Software Manual	Registered User Area ▸Software documentation VC Smart Cameras
VCRT Operation System TCP/IP Functions Manual	VCRT 5.0 TCP/IP Manual	Registered User Area ▸Software documentation VC Smart Cameras
VCLIB 2.0 /3.0 Image Processing Library Manual	VCLIB 2.0/ 3.0 Software Manual	Registered User Area ▸Software documentation VC Smart Cameras

The Light bulb highlights hints and ideas that may be helpful for a development.

This warning sign alerts of possible pitfalls to avoid. Please pay careful attention to sections marked with this sign.

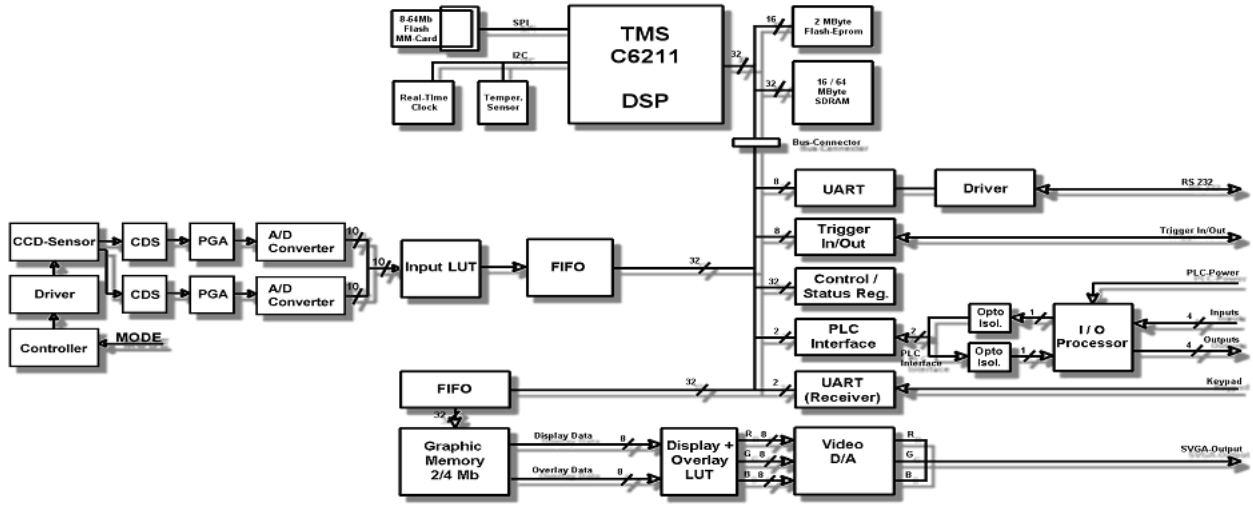
Author: Peter Neuhaus, VC Support, **mailto:support@vision-comp.com**

Table of Contents

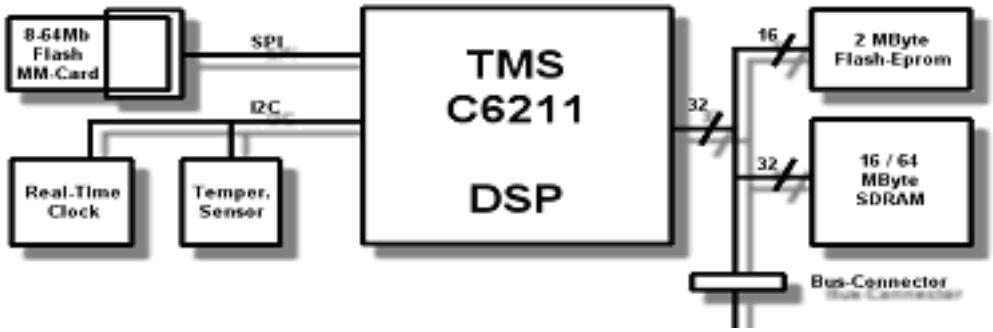
1 How to use this Manual

This programming tutorial has been designed for self- learning of programming Vision Components Smart Cameras with Texas Instrument DSP.
Please read the getting started manual " VC20XX and VC40XX Installation Manual" prior to working through this manual since it includes the basics needed to work with VC Cameras.

2 Hardware Structure VC Cameras



2.1 Memory Structure

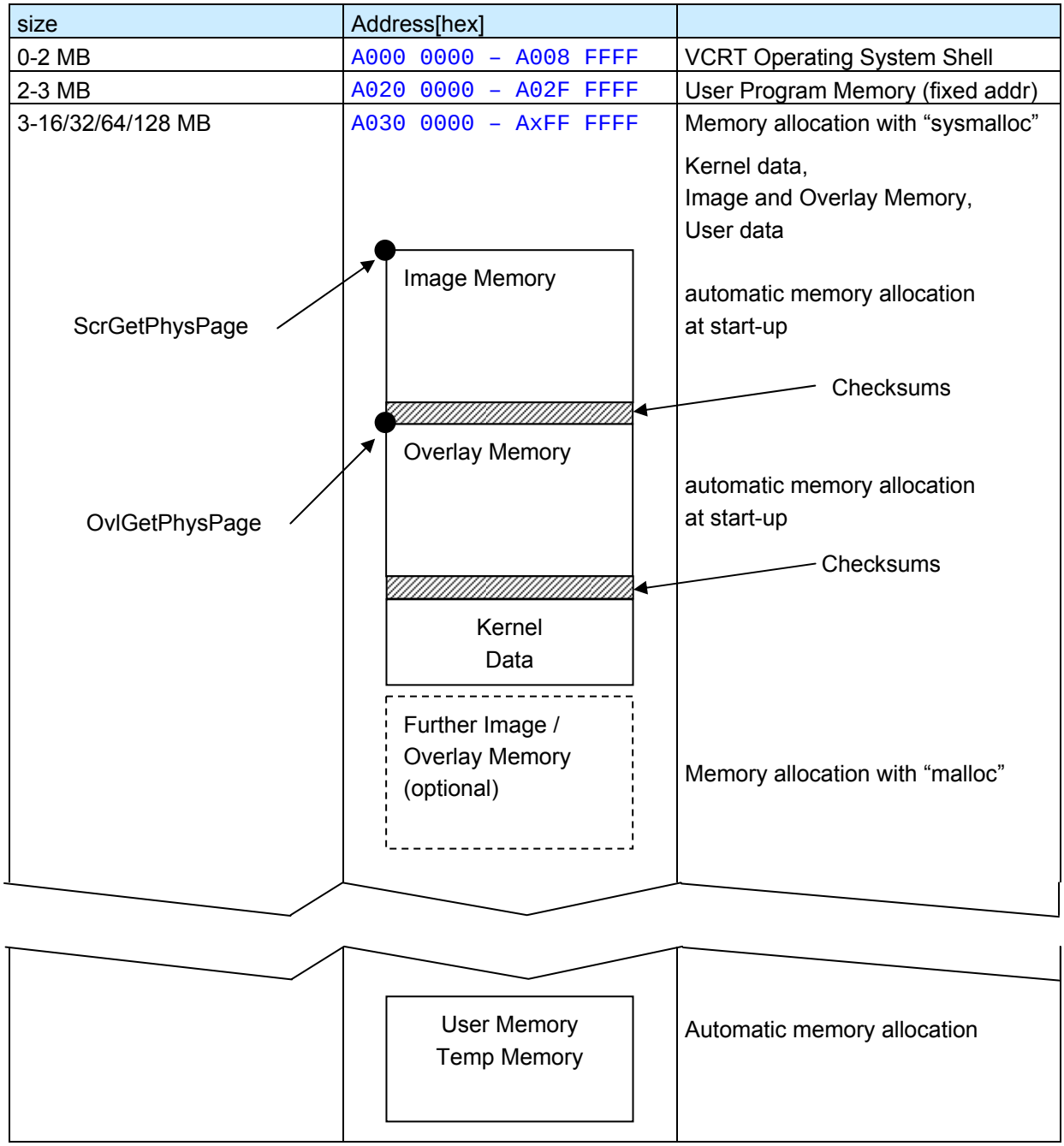


Flash Eprom 4MB	SDRAM 32MB	SD Card 128 MB
Non volatile	Volatile	Non volatile
User programs	Operating system VCRT Program Memory Kernel Image memory Overlay memory	Can be used to store user data: - images - other data

3 Memory structure SDRAM

3.1 Memory structure SDRAM Without Relocation

memory size	16 MBytes	32 MBytes	64 Mbytes	128 MBytes
start address	0xA0000000	0xA0000000	0xA0000000	0xA0000000
end address	0xA0FFFFFF	0xA1FFFFFF	0xA3FFFFFF	0xA7FFFFFF
size (hex)	0x01000000	0x02000000	0x04000000	0x08000000



Also see the “cc.cmd” file in C:\ti\work

Remark: calling the function `malloc()` allocates memory in the program heap (TI-function)

3.2 Memory structure with Relocation since VCRT 5.23

- flexible Memory structure (Program memory, image pages and user memory)
- no limitation to 1 MB Program memory
- Programs automatically relocated and loaded to next available start address – manual adjustment of program start addresses no longer required.
- File size increases due to relocation information
- Shell command `mdir` lists all loaded relocatable programs
- On program start relocation offsets are printed when print option is set

ccr.bat

```
cl6x -o3 -mi100000 -pl %1.c
lnk6x -ar -u _c_int01 %1.obj -m %1.map -o %1.out ccr.cmd
strip6x %1.out

copy %1.out exec.out
..\util\econv %1
..\util\scvt
copy adsp.msf %1.msf
```

ccr.cmd

```
-c
/* -priority */ /* CCS 3.0 and above */
-l vcrt4.lib
-l vclib.lib
-l extlib.lib
-l colorlib.lib
-l flib.lib
-l rts6200.lib
-u _c_int01
-e _c_int01
-stack 0x4000 /* adjust appropriate - stack size: min=0x4000 max depends on camera max mem */
-heap  0x400  /* adjust appropriate - heap size : min=0x400  max depends on camera max mem */

MEMORY
{
    PMEM:   o = 0h       l = 0ffffffffh
}

SECTIONS
{
.text   : ALIGN(32) { *(.text) } > PMEM
.const  : ALIGN(8) {} > PMEM
.data   : ALIGN(8) {} > PMEM
.bss    : ALIGN(8) { *(.bss)   } > PMEM
.cinit  : ALIGN(4) { *(.cinit) } > PMEM /* cflag option only */
.pinit  : ALIGN(4) {} > PMEM /* cflag option only */
.stack  : ALIGN(8) {} > PMEM /* cflag option only */
.far    : ALIGN(8) {} > PMEM /* cflag option only */
.sysmem: ALIGN(8) {} > PMEM /* cflag option only */
.switch: ALIGN(4) {} > PMEM /* cflag option only */
.cio    : ALIGN(4) {} > PMEM /* cflag option only */
}
```

Note: Using Auto Relocation, avaliable since VCRT 5.23 / VCLIB 3.01 the memory allocation is no longer fixed – see the release notes of the VCRT 5.23 / VCLIB 3.01 setup. For Auto Relocation use the project file: new5XX_CCS31r.pjt. This project file links the ccr.cmd file that contains the right settings for using auto relocation.

3.2.1 Image Page definitions (also refer to section 3.3):

Capture Page:

- Page where the image is stored after image capture. This usually is the first image memory page allocated automatically by the system.

Display:

- Page from where the image is displayed. This page is usually identical with the Capture page, as it is in general desired to display the captured image. The Display page can be set however to a different page as well. This is usefull in “Ping Pong” mode, where only the processed image is displayed.

Physical Page:

- The physical page is usually set to the captured and display page (see SetPhysPage macro in macros.h) if it is worked on the captured image.
- Page where image or overlay data is stored (memory address)
  - Physical Screen page
  - Physical Overlay page

Logical Page:

- Page currently worked on. For example ScrByteAddr(x,y) always refers to the logical page.
  - Logical Screen page
  - Logical Overlay page

Normaly logical and physical page are the same.

But it is possible to display an image and work on an
other image (ping pong)

3.3 Programming Macros

See also "**macros.h**" under C:\ti\c6000\cgtools\include and the "Macros" section in the VCLIB manual.

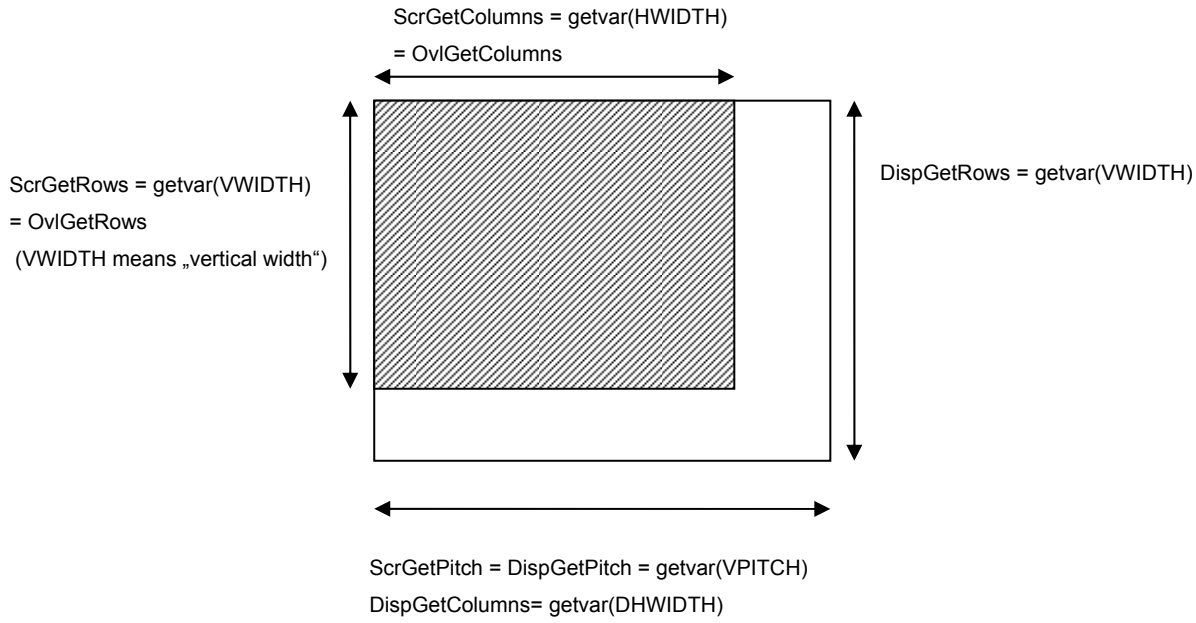
The VCRT and VCLIB libraries contain a number of macros that should be used for convenient and safe programming.

The following types of macros are available:

- definition of bits, bytes, words, pages
- aliases for video modi
- conversion macros
- image variable macros
- screen macros
- overlay macros
- utility macros

Refer to the "Macros" section of the VCLIB documentation for further reference.

3.3.1 Screen and Display Macros

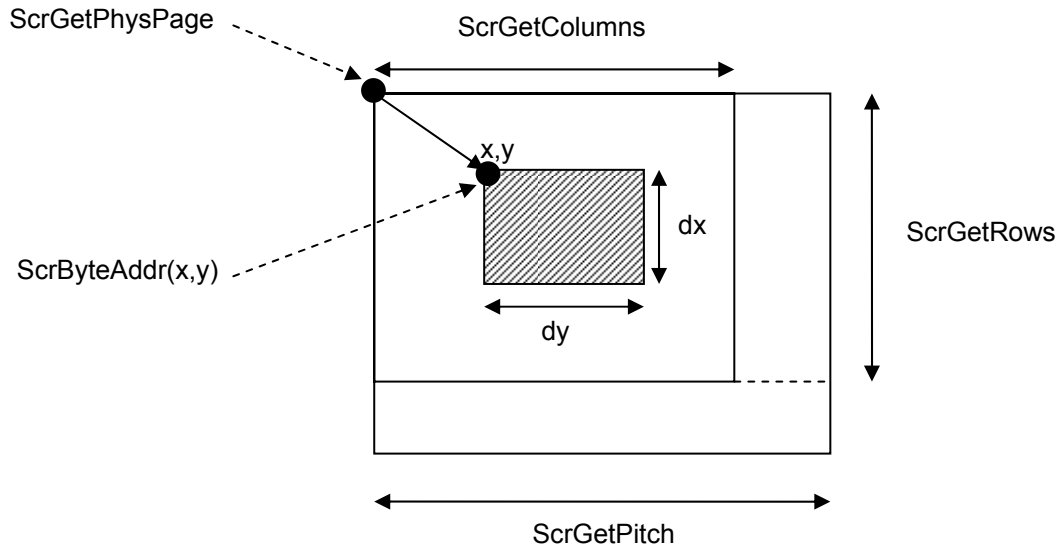


3.3.2 Screen, Display and Pich of different VC Camera Models

	VC4018 VCSBC4018	VC4038 VC4458	VC4065 VC4465	VC4066 VC4466	VC4068 VC4468	VC4472
Image Resolution	**640x480**	**640x480**	**768x582**	**1024x768**	**1280x1024**	**1550x1200**
ScrGetRows = getvar(VWIDTH)	480	480	582	768	1024	
ScrGetColumns = getvar(HWIDTH)	640	640	768	1024	1280	
ScrGetPitch = DispGetPitch = getvar(VPITCH)	640	800	800	1280	1280	
SizeOfScreen (ScrGetPitch*ScrGetRows)	640x480 = 307200	800x480 = 384000	800x582 465600	1280x768 983040	1280x1024 1310720	
Display Resolution	**N / A**	**800x600**	**800x600**	**1280x1024**	**1280x1024**	**1600x1200**
DispGetRows = getvar(VWIDTH)	**0**	600	600	1024	1024	
DispGetColumns= getvar(DHWIDTH)	**0**	800	800	1280	1280	

3.3.3 Assignment of Image Variables

In order to do any processing an ROI (region of interest) has to be defined.



assignment of a whole image variable in just one statement

```
#define ImageAssign (a,newst,newdx,newdy,newpitch)
{(a)->st=(long)(newst);(a)->dx=(I32)(newdx);(a)->dy=(I32)(newdy);
(a)->pitch=(I32)(newpitch);}
```

For example:

```
Image window;          // Declaration of image variable window
ImageAssign(&window, ScrByteAddr(100,100),400,400,ScrGetPitch);
```

3.4 System Variables

See **sysvar.h -** also see the “Appendix Sysvar” in the VCRT5 manual.

VC/RT allows access to a series of system variables. Their addresses are defined in a header file called **`sysvar.h`**. Please always use the names in this header file as a reference. Do not use absolute addresses, as they may be changed while the development of the cameras continues. System variables may be accessed using the functions `getvar()` , `setvar()` , `getlvar()` and `setlvar()`.

The following is a list of the most important system variables:

Number	Name	Description
0	DISP_PERIOD	refresh rate for display & overlay
1	DISP_CNT	counter for refresh rate
2	DISP_START	start address for display
3	OVLY_START	start address for overlay
4	DISP_ACTIVE	0: no refresh / 1: refresh (display)
5	OVLY_ACTIVE	0: no refresh / 1: refresh (overlay)
6	CAPT_START	start address for image capture
7	HWIDTH	active horizontal pixels
8	VWIDTH	number of active vertical lines
9	VPITCH	video pitch
10	EXPCNT	number of exposure cycles (lines)
11	GAIN	video gain value
12	IMODE	video mode
13	VSTAT	video status 0=idle 1=capture busy
14	INTFL	interrupt flag
15	CPUCLK	master cpu clock frequency
16	MSEC	real-time clock: millisecond
17	SEC	real-time clock: seconds since 1900
19	EXUNIT	time unit for exposure control [usec]
20	TIMESTAMP	timestamp for last captured image[ms]
21	DAYLIGHT	daylight savings time flag
22	TIMEZONE	real-time clock: timezone
23	LOWBAT	low battery voltage: time invalid
24	TEMP	cpu board temperature
25	VERSION	VCRT software version
26	DRAMSIZE	size of main SDRAM
27	HEAP	start address of heap
29	TOPSTK	top of stack
30	STSIZE	stack size - linker info only
31	PLCOUT	PLC output value
32	PLCIN	PLC input value
33	EXPOSING	image number of image being exposed
34	STORING	image number of image being stored

35	IMGREADY	image number of image ready for use
36	POWFAIL	1: PLC power failure / 0: power ok
37	LATENCY	maximum interrupt latency
38	MMC	missing multi-media card: -1
39	IPADDR	IP Address (ethernet version) */
40	IPMASK	IP mask (ethernet version) */
41	IPGATE	IP gateway (ethernet version)
42	DHCP	dhcp 1=on 0=off -1=failure
43	TPRIORITY	exec1 task priority default=9
46	CAPT_ERR	capture error counter (DM640 only)
49	MODEL	camera model
50	DHWIDTH	display horizontal width
51	DVWIDTH	display vertical width
53	PRIVATE	index for private sysvars
72	TIME_SLICE	time_slice
73	SENSORID	sensor id of camera head
74	PRIVATESYS	storage for 50 private sysvars

Example of how to use a system variable

```
#include <sysvar.h>

void set_display_start(int addr)
{
setvar(DISP_START, addr);     /* Use of system variable DISP_START */
}
```

Also See the "Sysvar.c" Demo file that is described in further detail in section 11.1 of this manual.

3.4.1 Example for hardware dependent System Variables:

	VC4018 VCSBC4018	VC4038 VC4458	VC4065 VC4465	VC4066 VC4466	VC4068 VC4468	VC4472
EXPCNT	10ms = 161 33.3 (31) = 0 95.7 (98) = 1 158.1 (160) = 2	10ms =322 5[1] (4) µs = -3 10 (8) µs = -2 15 (12) µs = -1 20 (16) µs = 0 50.5 (50)µs =1	10ms = 322 5[2] (4) µs = -3 10 (8) µs = -2 15 (12) µs = -1 20 (16) µs = 0 50.5 (52)µs =1	10ms = 238 10µs = -2 15µs = -1 20 = 0 62.3= 1	149 5 (8) = -3 10 (17) = -2 15 (25) = -1 20 (34) = 0 87 (87) = 1	
EXUNIT	62	31	31	42	67	
GAIN (default)	256	96	96	96	96	
MODEL	4018	4038	4065	4066	4468	
IPADDR (example)	0xC0A80053 = 192.168.0.83	0xC0A80053 = 192.168.0.83	0xC0A80053 = 192.168.0.83	0xC0A80053 = 192.168.0.83	0xC0A80053 = 192.168.0.83	
SESORID	0 (N / A)	0	16	66	81	

[1] Shutter times in brackets according to value displayed when using shell command „sh"

4 Setup Test – Hello World

The basic function of Vision Components' programming environment are best explained with help of demonstration programs. This is done in the following sections.

4.1 Hello World

This is an exercise for source code compilation, uploading of the program into the camera memory and program execution. All demo Programs covered in this totorial are copied to the following folder upon installation of the VC SDK-Ti PC Libraries (VCRT525_VCLIB303_Setup.exe) to the default directory:

C:\ti\MyProjects\Demo Programs\Program_Tut\...

1. Copy the folder "hello World" from C:\ti\MyProjects\Demo Programs\Program_Tut\2.1 hello to "C:\ti\myprojects\hello" in order to avoid overwriting of the original files.
2. Copy the following file: "new5XX_CCS31r.pjt" from "C:\ti\MyProjects\NewProjects" into your project directory "hello".
3. Launch Code Composer Studio and open "new5XX_CCS31r.pjt" using the pull down menu "Project" and then "Open"

Refer to the "Getting Started VC.." or "Schnellstart VC" or "Démarrage rapide Smart Cameras VC**"** for detailed instructions on program compilation, licencing and SW installation.

```
/*********************************************************************/

#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>

/*********************************************************************/

void main(void)

 {
 print("Hello World\n");          /*the "print function" sends strings to the PC via serial RS232 or Telnet
                                  port 23 with Ethernet cameras – this function is the substitution for the
                                  "printf" AINSI C function */
 }

/*********************************************************************/
```

This program can be compiled and uploaded quickly to the camera for testing purposes.

5 First Image Processing on Image and Overlay

5.1 Example 1: Drawing into the image or into the overlay:

Step number	Drawing into the image	Drawing into the overlay
Source File:	Draw_image.c	Draw_Overlay.c
1	#include <vcrt.h> #include <vclib.h> #include <macros.h> #include <sysvar.h>	
2	void main(void) { int Image; image window;	void main(void) { int Over; image window;
3	init_licence("TXXXXXXXXXX")	init_licence("TXXXXXXXXXX")
4	vmode(vmLive); /*switch camera to life mode*/	Vmode(vmOvlLive); /*switch camera to Overlay life mode*/
5	/* get the actual memory address for the physical page */ Image = (int)ScrGetPhysPage; /*setting the working page to the physical page*/ ScrSetLogPage(Image);	/* get the actual memory address for the overlay page*/ Over = (int)OvlGetPhysPage; /*setting the working page to the overlay page*/ OvlSetLogPage (Over);
6	/* Take pictures */ tpict();	
7		OvlClearAll;
8	/* Display pictures – assign image variable*/ ImageAssign(&window, ScrByteAddr(100,100),400,400,ScrGetPitch);	

9	/* Using VCLIB image processing functions*/
	framed(&window, 128);
	markerd(&window, 64);
	ellipsed(&window, 32);	

10		/*set overlay colors*/
		set_overlay_bit(7, 255, 0, 0);
		set_overlay_bit(6, 0, 255, 0);
		set_overlay_bit(5, 0, 0, 255);

11		set_ovlmask(255); /*display overlay*/
12	vmode(vmStill); /*Still Mode displays memory* }	vmode(vmOvlLive); /*Live mode display*/ }

Explanation of the steps in above program:

Step 3: Iinitialise T…Licence (see section 4, “Getting Started VC Guide”)

Step 4: Switching the Camera to Life mode

Step 4: Setting the logical page

Step 5: Clear Overlay

Step 6: Taking an Image

Step 7: Assigning the variable “image”

Step 8: Using image processing functions

Step 9: Setting the overlay colors

A maximum of 7 solid overlay colors and 3 translucent overlay colors that can be used in one overlay using the set_overlay_bit functions. No further programming is required in this case.

Setting solid overlay colors:

The bitplanes for solid overlay colors range from 2 to 7, representing 0000 0000

overlay colors can be used at the same time using this function (bit=2… bit=7) plus 2 additional translucent overlay colors (bit = 0 and bit = 1). See section 11.13.1 of the VCRT5 manual.

The “grey values” set in step 8 represent the following bit levels (solid colors):

Grey Values	Bit	Bit Level
4	0000 0100	2
8	0000 1000	3
16	0001 0000	4
32	0010 0000	5
64	0100 0000	6
128	1000 0000	7

Solid (covering) overlay colors are assigned to the bit level number, not the corresponding grey level!

Step 10: Overlay display on

Step 11: Switching to life mode or still mode

6 Image Acquisition

6.1 Image Acquisition Functions

Low level image acquisition function:

```
int capture_request (int exp, int gain, int *start, int mode)
```

allows to set the following parameters with the function call:

- Exposure time (shutter speed)
- Gain
- Capture page (image page where the image is stored)
- Hardware or software trigger
- Full frame or binning mode

Capture_request returns a unique, incremented tracking number for each image.

High level image taking function – software trigger:

Software trigger means the image acquisition is program controlled.

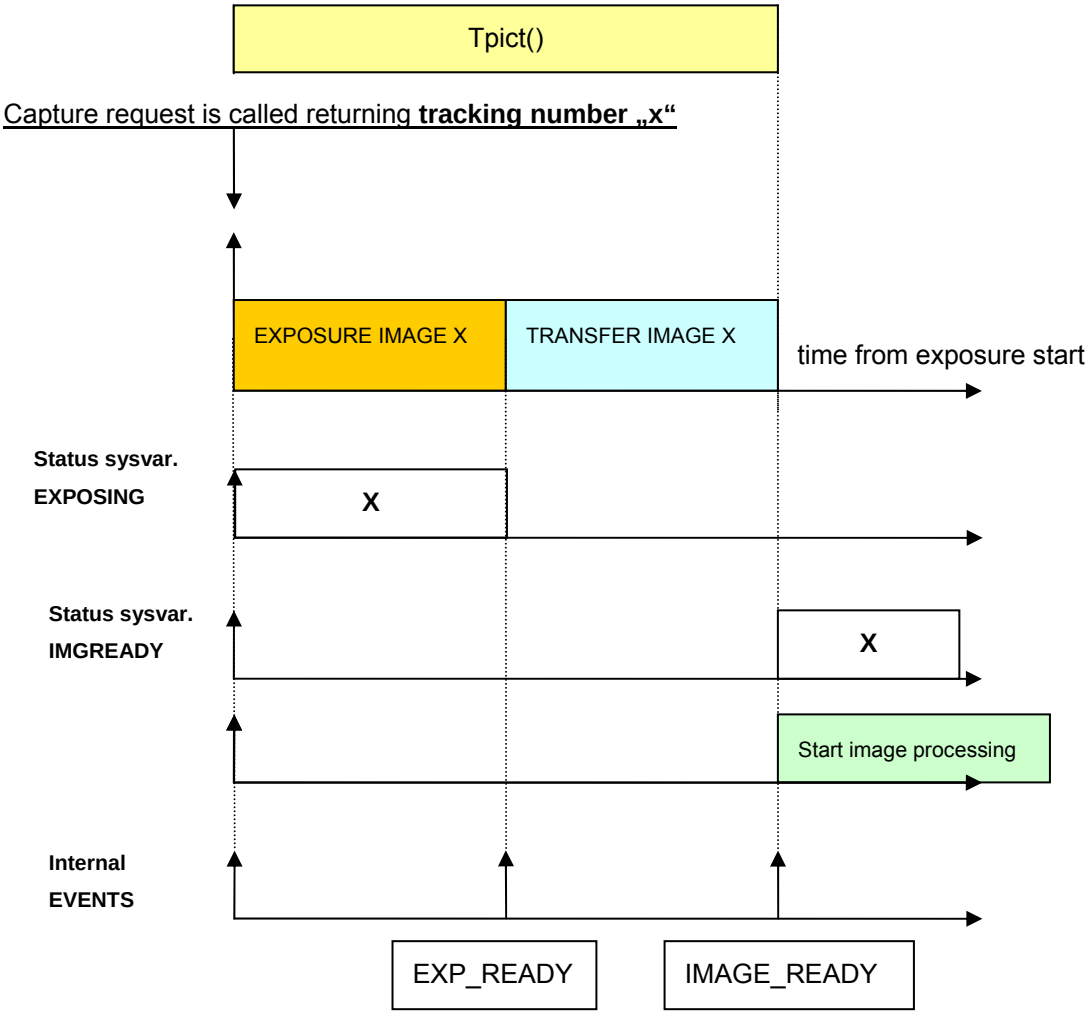
	Software Trigger
High Level Functon	`tpict()`
Equivalent low Level Function	`Capture_request(getvar(EXPCNT), getvar(GAIN), ScrGetCaptPage, 0)`

High level image taking function – hardware trigger:

Hardware trigger means the image acquisition is controlled by the trigger input signal.

	Hardware Trigger
High Level Functon	`tenable()`
Equivalent low Level Function	`Capture_request(getvar(EXPCNT), getvar(GAIN), ScrGetCaptPage, 1)`

6.2 Timing during Image Acquisition



6.3 Timetest

This program consists of 4 parts:

1. Measurement of the tpict() function execution time
2. Measurement of the function call capture_request, exposure and transfer time by polling System variables “EXPOSING” and “IMGREADY”
3. Time measurement of the same steps using events
4. Image acquisition loop polling system variables
5. Image acquisition loop using events (frees CPU time for parallel image transfer)

Execution of this demo with parallel image acquisition:

1. Start img3r as background task $img3par &
2. Start “ATX Client”
3. Start “timetest.c”
4. The program “timetest.c” outputs the following:

Program Output	Explanation
tpict = shutter speed + transfer time t=26ms Press any key - tc_rq = 0, texp = 10, ttr = 15, ttot = 25 Press any key - tc_rq = 0, texp = 10, ttr = 15, ttot = 25 tracking No 1 = 9738 , tracking No.2 = 9739 exposure ready event occurred image ready event occurred press any key to start image loop polling Press 'q' to stop Press any key to start image loop using events Press 'q' to stop	Executing time of “tpict” Exposure time, transfer time and total time polling system variables Exposure time, transfer time and total time using wait(Event, timeout). If sh > 100ms the exp ready event times out, since while(wait(EXP_READY,100) != 1) Image loop using System Variables → No parallel image transfer Image loop using the wait() function → live images are transferred to ATX client)

Without events – checking if an image had been stored in memory had to be done polling the corresponding system variable **IMGREADY**:

```
ImgNr = capture_request(getvar(EXPCNT), getvar(GAIN),(int*)Screen0,0);
while(ImgNr != getvar(IMGREADY));  //Wait until complete image has been stored in memory
```

→ Now the image in memory can be processed.

No CPU time is available for allowing another process to execute during polling the system variable IMGREADY !

Using the internal event “IMAGE_READY” together with the wait function frees CPU time:
A background process for instance can transfer images via TCP/ IP during image transfer.

timetest.c

```
/*****************************************************************/
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
#include <flib.h>
#include <licence.h>
/*****************************************************************/

void main(void)
{
  int ms=0, ms1, ms2, ms3, ms4, ms5, tc_rq, texp, ttr, ttot, x =3, y=3;
  int TrackNr[3];
  int key = 0, ImgNr ;

  init_licence("TXXXXXXXXXX");

  tpict();

////////////////////////////////////
// 1. image aquisition with tpict //
////////////////////////////////////
         ms = getvar(MSEC);
  tpict();
         ms = getvar(MSEC)-ms;
  if(ms<0) ms+=1000;
  print("tpict = shutter speed + transfer time t=%dms\n",ms);
  print("\nPress any key - \n "); getchar();


//////////////////////////////////////////////////////////////////
// 2. image aquisition with capture_request polling tracking number //
//////////////////////////////////////////////////////////////////
         ms1 = getvar(MSEC);

  TrackNr[0] = capture_request(getvar(EXPCNT), getvar(GAIN),(int*)ScrGetCaptPage,0);
         ms2 = getvar(MSEC);
  while(TrackNr[0]== getvar(EXPOSING));
         ms3 = getvar(MSEC);
  while(TrackNr[0] == getvar(STORING));
         ms4 = getvar(MSEC);
  while(TrackNr[0] != getvar(IMGREADY));
         ms5 = getvar(MSEC);
         tc_rq = ms2 - ms1;
         if (tc_rq<0) tc_rq+=1000;
         texp = ms3 - ms2;
         if (texp<0) texp+=1000;
         ttr = ms4 -ms3;
         if (ttr<0) ttr+=1000;
         ttot = ms5- ms1;
         if (ttot<0) ttot+=1000;

  print("tc_rq = %d, texp = %d,  ttr = %d, ttot = %d \n", tc_rq, texp, ttr, ttot);
```

```
////////////////////////////////////////////////////////
// 3. image aquisition with capture_request using events //
////////////////////////////////////////////////////////
  print("\nPress any key - \n"); getchar();
  wait(EXP_READY, 1); // in case the event has already occured and wait returns 2
         ms1 = getvar(MSEC);
  TrackNr[1] = capture_request(getvar(EXPCNT), getvar(GAIN),(int*)ScrGetCaptPage,0);
         ms2 = getvar(MSEC);
  do
    {
    x = wait(EXP_READY,100); // waits until next event "Exposure Ready" occurs, timeout 100 ms
    } while(x!=1); // wait returns 2 and does not wait for the next event, if the event has occured before wait was called
         ms3 = getvar(MSEC);
  do
    {
       y = wait(IMAGE_READY, 100); // waits until next event "Image Ready" occurs, timeout 100 ms
    } while(y!=1); // wait returns 2 and does not wait for the next event, if the event has occured before wait was called
         ms4 = getvar(MSEC);
         tc_rq = ms2 - ms1;
         if (tc_rq<0) tc_rq+=1000;
         texp = ms3 - ms2;
         if (texp<0) texp+=1000;
         ttr = ms4 -ms3;
         if (ttr<0) ttr+=1000;
         ttot = ms4- ms1;
         if (ttot<0) ttot+=1000;
  print("tc_rq = %d, texp = %d,  ttr = %d, ttot = %d \n\n", tc_rq, texp, ttr, ttot);
  print("tracking No 1 = %d , tracking No.2 = %d \n", TrackNr[0], TrackNr[1]);
  print("x= %d, y= %d \n", x, y);

 switch(x)
       {
       case 1:
              print("exposure ready event occured\n");
            break;
       case 2:
            print("exposure ready event has occured before wait was called occured\n");
            break;
       case -1:
            print("exposure ready event has timed out\n");
            break;
      default:
              print("Return value wait(EXP_READY,x) = %d \n", x);
       }

switch(y)
       {
       case 1:
              print("image ready event occured\n");
            break;
       case 2:
            print("image ready event has occured before wait was called\n");
            break;
       case -1:
            print("image ready event has timed out\n");
            break;
      default:
              print("Return value wait(IMAGE_READY,x) = %d \n", y);
       }
```

```
////////////////////////////////////////////////////////
// 4. image aquisition loop using getvar(IMGREADY)       //
////////////////////////////////////////////////////////

  print("\nPress any key to start image loop polling \n Press 'q' to stop \n"); getchar();

      do
        {
         if (kbhit())
                    {
                       key = rs232rcv();
                       }


           ImgNr = capture_request(getvar(EXPCNT), getvar(GAIN),(int*)ScrGetCaptPage,0);

            while(ImgNr != getvar(IMGREADY));
        }
        while(key != 'q');

     key = 'a';

////////////////////////////////////////////////////////////////////
// 5. image aquisition loop using events  wait(IMAGE_READY, 200)); //
////////////////////////////////////////////////////////////////////

  print("\nPress any key to start image loop using events \n Press 'q' to stop \n");
getchar();

      do
        {
         if (kbhit())
                    {
                       key = rs232rcv();
                       }
            capture_request(getvar(EXPCNT), getvar(GAIN),(int*)ScrGetCaptPage,0);
            while(wait(IMAGE_READY,100)!= 1);
        }
        while(key != 'q');
   vmode(vmLive);
}

/*****************************************************************/
```

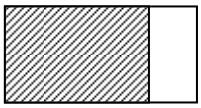
6.4 Allocation of additional image pages:

DRAMScreenMalloc()

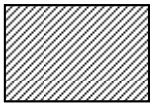
→ allocates memory for „SizeOfScreen“ = ScrGetPitch x ScrGetRows = 800x480Pixel with VGA cameras.

→ Only use this macro for cameras without integrated Video output!

Allocated memory for cameras with Video output (image height x display width):



Allocated memory for cameras without Video output (image size):

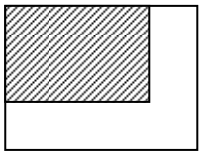


DRAMDisplayMalloc()

→ allocates sufficient memory for Display Screen = DispGetPitch x DispGetRows = 800x600 Pixel with VGA cameras.

→ Use this macro if display of the image page is required!

→ Do not use this macro when programming cameras without Video output, since no memory will be allocated (DispGetRows = 0) !



6.5 Memory Alignment

When allocating memory for additional image pages it is important to align the start addresses to a multiple of 1024:

```
    int OldScreen, NewScreen, NewScreenAlign;

     /* get addr of old screen */
     OldScreen=ScrGetPhysPage; /* OR ScrGetCaptPage OR ScrGetDispPage */

     /* allocate new screen page */
     NewScreen=(int)DRAMScreenMalloc();

     /* Image display addresses have to be in alignments of 1024 */
     NewScreenAlign=(NewScreen&0xFFFFFC00) + 1024;

     /* switch to new screen page */
     ScrSetPhysPage(NewScreenAlign); /* OR ScrSetCaptPage OR ScrSetDispPage */
...
/* At the end of the program */
     /* switch back to old screen page */
     ScrSetPhysPage(OldScreen);
```

```
/* release screen allocation */
DRAMPgFree(NewScreen);
```

Note that DRAMScreenMalloc() may allocate more memory than sufficient for the actual image size, since the display size is for instance 600x800 for a 480x640 image.

6.6 Assignment of Image Variable to a new Image Page

In order to do any processing an ROI (region of interest) has to be defined.

```
Image src;              // declaration of the image variable “src”


/* assignment of an image variable */
ImageAssign(&a, NewScreenAlign, DispGetColumns, DispGetRows, DispGetPitch);
```

The image variable “a” has been assigned to the image page “NewScreenAlign” defined in the section above.

6.7 Image Processing in Ping Pong Mode

6.7.1 Flicker

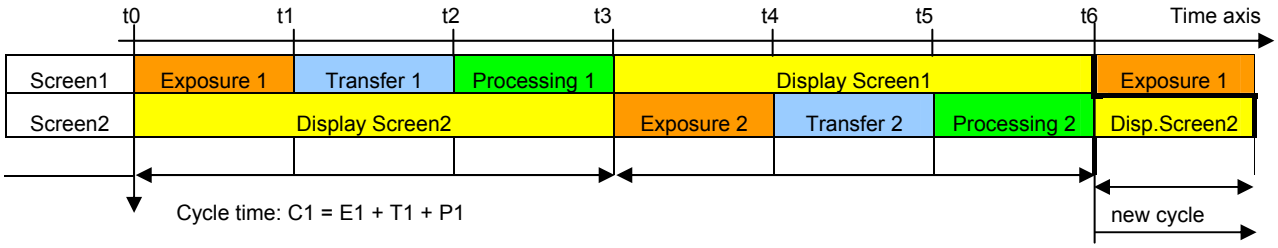
The following loop takes an image, stores it at the current capture page and then processes the image variable “src” in place.
If the same image page is displayed on the video output, both – the captured and the processed image will be displayed in random order. This results in a flickering image:

```
/*****************************************************************/

#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
#include <flib.h>

/*****************************************************************/

void main(void)
{

  int key = 0;
  image src;

  init_licence(“T1244353523”);  //Initialising VC SDK-Ti Licence

  /* Assiging image variable for processing Sobel filter */
  ImageAssign(&src, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);
                  //Declaring an image variable starting at (100;100), width = image width/2, height = image height /2

  print("press any key to start, press 'q' to stop progam\n");

      do
        {
         tpict();                      // takes 1 image and sets camera into still mode

         if(kbhit()) key = rs232rcv();// reads character in case of a keyboard input

        /* Sobel Filter*/
         sobel(&src, &src);    // processes the image variable “scr” inplace with a sobel filter

        }
        while(key != 'q');

}

/*****************************************************************/
```

6.7.2 Ping Pong

With help of a second image page (and overlay page) it is possible to display either the captured or the processed image (and overlay). For this the capture and the display page need to be swapped after every image acquisition – this is called “Ping Pong” mode.

Ping Pong Timing Diagram:



In order to be able to swap capture and display pages, at least 2 image pages (Screen1, Screen2) are required. The diagram above shows the processing order for both screens, as performed in the main loop of the demo program:

1. taking image on screen 1
2. processing screen 1
3. Switching Display page (image and overlay) to page 1
4. taking image on screen 2
5. processing screen 1
6. Switching Display page (image and overlay) to page 2

After step 6, the main program cycle starts from the beginning.

The program outputs the times t0, t2, t3, t5, and t6 (executing “pingpong.c” on a VC4038):

- t0= 0, t2 = 26, t3= 31. t5 = 57, t6 = 61
- These times remain constant (in contrast to parallel mode, see section 6.8)

→ Using pingpong mode with a VC4038 and 10ms exposure time, about 32 parts per second can be inspected.

Sourcecode: “pingpong.c”

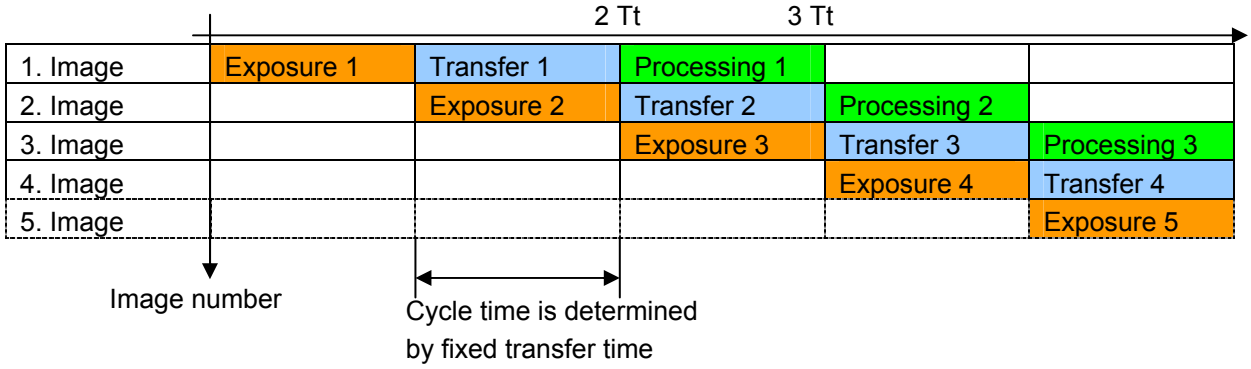
```
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
#include <flib.h>

void main(void)
{
  int Screen1, Overlay1, Screen2_not_aligned,  Screen2, Overlay2_not_aligned, Overlay2;
  int t0, t2, t3, t5, t6, key = 0;
  image CleanScreen, a, b, c, d;

        /* Initialising Licences */
        Init_licence(“T1234567890”); //initialising your VC SDK-Ti licence code
```

```
/* Memory Allocation for addtional image and overlay pages */
// obtaining start address of page 1 = default capture page (overlay page)
Screen1 = ScrGetCaptPage;
Overlay1 = OvlGetPhysPage;
// Memory Allocation for 2nd image and 2nd Overlay Page
Screen2_not_aligned = (int)DRAMDisplayMalloc();
Screen2 = (Screen2_not_aligned&0xFFFFFC00) + 1024;
Overlay2_not_aligned = (int)DRAMDisplayMalloc();
Overlay2 = (Overlay2_not_aligned&0xFFFFFC00) + 1024;

/*  Clean borders image and overlay */
ImageAssign(&CleanScreen, Screen1, DispGetColumns, DispGetRows, DispGetPitch);
set(&CleanScreen, 0);
ImageAssign(&CleanScreen, Screen2, DispGetColumns, DispGetRows, DispGetPitch);
set(&CleanScreen, 0);
ImageAssign(&CleanScreen, Overlay1, DispGetColumns, DispGetRows, DispGetPitch);
set(&CleanScreen, 0);
ImageAssign(&CleanScreen, Overlay2, DispGetColumns, DispGetRows, DispGetPitch);
set(&CleanScreen, 0);

/* Assigning image variables for source and destination for Sobel filter */
ScrSetLogPage(Screen1); // the following ScrByteAddr refers to the current logical page
ImageAssign(&a, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);
// image var "a" is assigned to Screen1
ScrSetLogPage(Screen2);
ImageAssign(&b, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);

/* Assigning image variables for the overlay */
ImageAssign(&c, Overlay1, DispGetColumns, DispGetRows, DispGetPitch);
ImageAssign(&d, Overlay2, DispGetColumns, DispGetRows, DispGetPitch);

/* Selecting Overlay Colors and switch Overlay on (refer to section 5.1) */
set_overlay_bit(7,255,0,0); //Bit level 7 set to red,
set_overlay_bit(6,0,255,0);

/* Setting the camera into the correct start mode for capture request and overlay */
tpict(); //vmode(vmOvlLive) is not sufficient, since it waits for image capture!
set_ovlmask(255);
```

```
        /* Main Program Loop */
   do
     {
        if (kbhit()) key = rs232rcv();
                      setvar(MSEC,0);
                      t0 = getvar(MSEC);
        // 1. taking image on screen 1
        ScrSetCaptPage(Screen1);
        tpict();
                      t2 = getvar(MSEC);
        // 2. processing screen 1
        sobel(&a,&a);   // process Screen1
        markerd(&c,128); // draw overlay1
        // 3. Switching Display page (image and overlay) to page 1
        ScrSetDispPage(Screen1);
        OvlSetPhysPage(Overlay1);
        display_update(2);
                      t3 = getvar(MSEC);
        // 4. taking image on screen 2
        ScrSetCaptPage(Screen2);
        tpict();
                      t5 = getvar(MSEC);
        // 5. processing screen 1
        sobel(&b,&b);   // process Screen2
        markerd(&d,64); // draw overlay2
        // 6. Switching Display page (image and overlay) to page 2
        ScrSetDispPage(Screen2);
        OvlSetPhysPage(Overlay2);
        display_update(2);
                      t6 = getvar(MSEC);
        print("t0= %d, t2 = %d, t3= %d. t5 = %d, t6 = %d\n",t0, t2, t3, t5, t6);
        }
          while(key != 'q');

        /* Program Closure */
        ScrSetPhysPage(Screen1);
        Display_update(2); required in order to switch display page back to default phys. page
        ScrSetLogPage(Screen1);
        OvlSetPhysPage(Overlay1);
        OvlSetLogPage(Screen1);
        vmode(0); //switching camera back to live mode

        // Release allocated memory in reverse order!
        DRAMPgFree(Overlay2_not_aligned);
        DRAMPgFree(Screen2_not_aligned);
}
/*****************************************************************/
```

The function “display_update(2)” is required for VC4XXX cameras to ensure the display update is done after the image processing is completed.

6.8 Parallel Image Acquisition



The transfer time is the time required for reading the image data from the sensor, digitizing the analogue signal and saving the image to memory. This process determines the maximum camera frame rate (images per second).

Only VC smart cameras to perform all 3 steps – Exposure, Transfer and Processing in parallel. This means if exposure and processing time is not larger than the transfer time, the maximum frame rate is still maintained – even with parallel processing!

Parallel image acquisition with video output can be seen as an extension of Ping Pong mode.

The following is required for parallel image acquisition:

1. The allocation of at least 3 image pages is required (at least 2 pages without video display) – otherwise the next incoming image would instantly overwrite the previous image!
2. It is necessary to switch the pages - i.e. every cycle the “capture”, “logical” (working) page and display page needs to change.
3. The use of the “capture_request()” function is required (see section 6.2 and the “Video Control functions” section in the VCCRT documentation).
4. In order to avoid signal delays, at least 2 capture requests need to be called prior taking the first image!

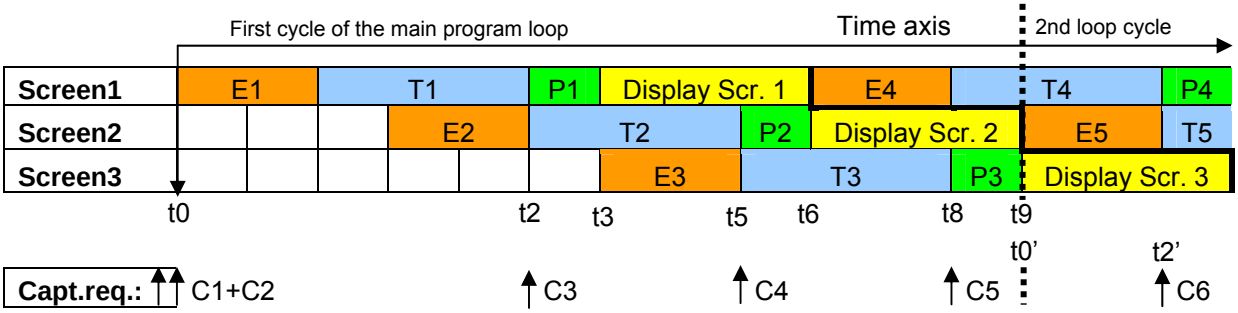
Comparison of demo Program “tp_par.c” and “parallel.c”:

- The demo program tp_par.c performs parallel processing in a loop fashion, which makes it more complex, but more flexible.
- The number of image screens can be selected and the calculation time simulated.

While “parallel.c” is easier to understand, use the “tp_par.c” program for inserting a an image processing task into a flexible parallel processing structure!

6.8.1 Demo Program “parallel.c”:

Timing Example for parallel image capture (execution times program “parallel.c with VC4038):



Notes:

- One diagram column corresponds roughly to 5 ms.
- The exposure/ transfer-/ and processing times correspond to the actual execution times of the “parallel” demo program, using a VC4038 with a 10ms shutter.
- C1 is the time the capture request call for the first image is made and so on.
- t0 to t9 are the actual times printed out by the demo program.
- t0’ and t2’ are the times of the second cycle of the main loop – since E4 and T4 starts prior to the end of P3, the total cycle time of the second and all following cycles “t9’” is one transfer time less than the first cycle time “t9”.

Executing the “parallel.c” demo program on a VC4038 results in the following times:

First cycle:	t0= 0, t2 = 25, t3= 30. t5 = 41, t6 = 46, t8 = 57, t9 = 62
Second and following cycles:	t0= 0, t2 = 9, t3= 14. t5 = 25, t6 = 31, t8 = 41, t9 = 46

with:

- Exposure time: 10ms (standard shutter)
- Transfer time: 16 ms (fix, dependent on camera model, here: VC4038)
- Processing time: 4ms (sobel processing of ROI 320 x 240, here with 400MHz Processor)

The second and all following loops are faster, since the exposure and part of the image transfer takes already place during the previous loop (see E4 and T4 in the diagram).
Since in this example both - shutter and processing times – are smaller than the transfer time, the cycle time is determined by the transfer time:

→ In this example one image processing is completed every 15 ms, which corresponds to the transfer time of the VC4038.

→ Using parallel mode with a VC4038 and 10ms exposure time, about 66 parts per second can be inspected (compare with section 6.7.2).

The following rules apply for parallel processing:

- Transfer times cannot overlap.
- Transferring an image needs to start immediately after finishing its exposure.

→ This means for this example: The exposure of the next image is delayed automatically by system, so the image transfer can start straight after exposure.

Note:

- The timing shown may not work for every possible combination of transfer, shutter and processing time! The order of events or the timing of the capture_request calls may need adjustment depending on the actual process times.
- Parallel or background image acquisition should not be confused with multitasking. Once the capture_request has been called, the FPGA or CPLD takes care of the image capture. Handling of multiple user tasks can be done in *addition* to background image capture.

Source code “parallel.c”

```
/*****************************************************************/
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
#include <flib.h>
/*****************************************************************/

void main(void)
{
  int Screen1, Screen2, Screen3, Overlay1, Overlay2, Overlay3, Screen2_not_aligned;
  int Screen3_not_aligned, Overlay2_not_aligned, Overlay3_not_aligned;
  int t0, t2, t3, t5, t6, t8, t9, key = 0;
  int tr1, tr2, tr3;

  image CleanScreen, a, b, c, d, e, f;

  /* Initialising Licences   */
      "init_licence("T...");


  /* Memory Allocation for addtional image and overlay pages */
      // obtaining start address of page 1 = default capture page
      Screen1 = getvar(CAPT_START);
      Overlay1 = OvlGetPhysPage;

      // Allocate screen page 2+3
      Screen2_not_aligned = (int)DRAMDisplayMalloc();
      Screen2 = (Screen2_not_aligned&0xFFFFFC00) + 1024;
      print("Screen2 =%x\n", Screen2);
          Screen3_not_aligned = (int)DRAMDisplayMalloc();
      Screen3 = (Screen3_not_aligned&0xFFFFFC00) + 1024;
      print("Screen3 =%x\n", Screen3);

          // Allocate overlay page 2+3
      Overlay2_not_aligned = (int)DRAMDisplayMalloc();
      Overlay2 = (Overlay2_not_aligned&0xFFFFFC00) + 1024;

      Overlay3_not_aligned = (int)DRAMDisplayMalloc();
      Overlay3 = (Overlay2_not_aligned&0xFFFFFC00) + 1024;
```

```
      /* Preparation of screen pages, image variables and other settings */

 /*  Clean borders image*/
     ImageAssign(&CleanScreen, Screen1, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);
     ImageAssign(&CleanScreen, Screen2, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);
     ImageAssign(&CleanScreen, Screen3, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);

/*  Clean borders overlay*/
     ImageAssign(&CleanScreen, Overlay1, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);
     ImageAssign(&CleanScreen, Overlay2, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);
     ImageAssign(&CleanScreen, Overlay3, DispGetColumns, DispGetRows, DispGetPitch);
     set(&CleanScreen, 0);

 /* Assiging image variables for source and destination for Sobel filter */
     ScrSetLogPage(Screen1); // the following ScrByteAddr refers to the current logical page
     ImageAssign(&a, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);
                                              //image var "a" is assigned to Screen1
     ScrSetLogPage(Screen2);
     ImageAssign(&b, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);
     ScrSetLogPage(Screen3);
     ImageAssign(&c, ScrByteAddr(100,100), DispGetColumns/2, DispGetRows/2, DispGetPitch);

  /* Assiging image variables for the overlay */

     ImageAssign(&d, Overlay1, DispGetColumns, DispGetRows, DispGetPitch);
     ImageAssign(&e, Overlay2, DispGetColumns, DispGetRows, DispGetPitch);
     ImageAssign(&f, Overlay3, DispGetColumns, DispGetRows, DispGetPitch);

  /* setting overlay colors*/
       set_overlay_bit(7,255,0,0);
       set_overlay_bit(6,0,255,0);
       set_overlay_bit(5,0,0,255);

  /* setting the camera to the correct start mode */

       set_ovlmask(255); // turning the overlay on, see section 5.1.

       tpict(); // tpict() is required in order to set the camera to still mode,
              vmode(vmstill) is not sufficient, since it waits until the next image has been
              completed until the camera actually turns into still mode!
              This should be avoided when using capture_requests!
```

```
   /* Main Program Loop with initial capture_requests */

   tr1 = capture_request(getvar(EXPCNT), getvar(GAIN), (int *)Screen1, 0);   // first
                                                                              capture request, C1
   tr2 = capture_request(getvar(EXPCNT), getvar(GAIN), (int *)Screen2, 0);   // second
                                                                              capture request, C2


do
   {
          if (kbhit()) key = rs232rcv();
                 setvar(MSEC,0);
                 t0 = getvar(MSEC);
   while(tr1 > getvar(IMGREADY)); // wait that first image has been written to memory
                 t2 = getvar(MSEC);

   tr3 = capture_request(getvar(EXPCNT), getvar(GAIN), (int *)Screen3, 0);
          sobel(&a,&a);   // process Screen1
          markerd(&d,128); // draw overlay1
          ScrSetDispPage(Screen1);
          OvlSetPhysPage(Overlay1);
          display_update(2);

                 t3 = getvar(MSEC);

   while(tr2 > getvar(IMGREADY)); // wait that the second image has been written to memory
                 t5 = getvar(MSEC);
   tr1 = capture_request(getvar(EXPCNT), getvar(GAIN), (int *)Screen1, 0);

          sobel(&b,&b);   // process Screen2
          markerd(&e,64); // draw overlay2
          ScrSetDispPage(Screen2);
          OvlSetPhysPage(Overlay2);
          display_update(2);
                 t6 = getvar(MSEC);

   while(tr3 > getvar(IMGREADY)); // wait that second image has been written to memory
                 t8 = getvar(MSEC);
   tr2 = capture_request(getvar(EXPCNT), getvar(GAIN), (int *)Screen2, 0);

          sobel(&c,&c);   // process Screen3
          markerd(&f,32); // draw overlay3
          ScrSetDispPage(Screen3);
          OvlSetPhysPage(Overlay3);
          display_update(2);
                 t9 = getvar(MSEC);
   print("t0= %d,  t2 = %d, t3= %d. t5 = %d, t6 = %d, t8 = %d,  t9 = %d\n",t0, t2, t3, t5,
   t6, t8, t9);

   }
   while(key != 'q');
```

```
        /* Program Closure */

        // switching all page settings back to default mode
        OvlSetPhysPage(Overlay1);
        OvlSetLogPage(Screen1);
        ScrSetDispPage(Screen1);
        display_update(2); // if not inserted here, the camera does not switch back to the
                           default display page!
        ScrSetCaptPage(Screen1);
        ScrSetLogPage(Screen1);

        // Release momory allocation in reverse order!
        DRAMPgFree(Overlay3_not_aligned);
        DRAMPgFree(Overlay2_not_aligned);
        DRAMPgFree(Screen3_not_aligned);
        DRAMPgFree(Screen2_not_aligned);

        vmode(vmLive); //switching camera back to live mode

}

/*********************************************************************/
```

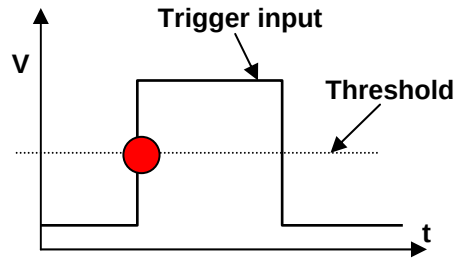
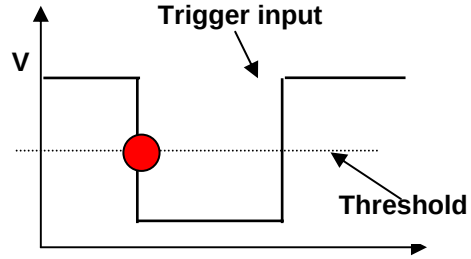
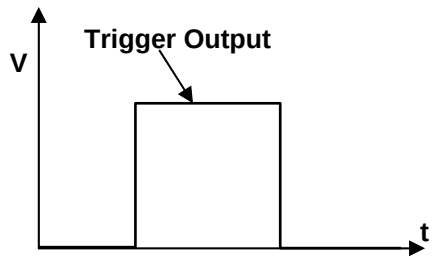
6.9 Hardware controlled image Acquisition

Program description: **trig_in.c**
This program use the trigger output and trigger input.
Please refer to the hardware documentation of the VC20xx,
chapter "Trigger Input and Trigger Output"
A buzzer is connected between +5V (brown cable) and
Trig. Out (grey cable)

Program Codes	**Comments**

/***/
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
/***/
void main(void)
 	
{
int TrigInp, Run = 1;
 	
TRIGINP_POS();
 	
while(!kbhit())
{
TrigInp = (int)(*((volatile int *)VIRTX_STAT) & 0x2000);
 	
if (TrigInp)
{
print ("Trigger Input ON\n");
}
else
{
print ("Trigger Input OFF\n");
}
 	
time_delay(20); // ms
}
 	
getchar();
 	
pstr("Waiting for Tigger (rising edge at Trig.Inp). Press any Key for quit\n");
 	
vmode(vmStill); /* set allowed vmode for tenable() */
TRIGOUT_EXP();
	/* Display status of Trigger Input */
	Status register for trigger input- note that there is a different register used for VC4018, VCSBC4018 and VC4002L cameras.
	
	Delay time of 20ms
	
	set correct vmode for tenable()
	set trigger output to exposure controlled mode;
	trigger out signal positive during

TRIGOUT_POS();
while(Run)
{
 	
tenable(); /* now wait for external trigger */
 	
while(!trdy())
{
if (kbhit())
{
print("kill capture\n");
getchar();
while(cancel_capture_rq());
Run = 0;
}
 	
}
pstr("*");
}
vmode(vmLive);
}
/***/
	exposure;
	Interrupt controlled image acquisition:
	
	Starts interrupt driven image acquisition for one image.
	
	Waits until the complete image has been written to DRAM memory.
	
	Cancel last capture request that would otherwise prevent live mode.
	
	Print string over the serial line
	
	Switch display to live mode

Signal	Trigger Mode
	TRIGINP_POS() Rising Edge Trigger Signal
	TRIGINP_NEG() Falling Edge Trigger Signal
	Trigger Output Signal in Exposure Controlled Mode: TRIGOUT_EXP(); AND Trigger output set positive: TRIGOUT_POS();

6.10 Input Lookup table

Program Description:

See the **lut_inp.c** demo program.

This file programs the input LUT in three differnt ways:
(new 10 bit input LUT for SPARTAN II devices)

1) inverse
2) logarithmic
3) linear

Input 10 Bit (LUT 1024 -15 values) – (the AD converter skips the first 15 values of the 1024 input values)
Output 8 Bit

Demo Program input_lut.c	Comments
/***/ #include <vcrt.h> #include <vclib.h> #include <macros.h> #include <sysvar.h> #include <math.h> /***/ void program_ilut(unsigned int *table); /***/ #define OFS 15 /***/	Offset for omitting the first 15 values

void main()
{
int i;
int table[1024];
 	
vmode(vmLive);
shutter(10000);
 	
pstr("Input LUT inverse (Press any key)\n");
for(i=0; i<1024; i++)
{
	Programming the lookup table in inverse order (0 becomes highest grey value 255).

```
   table[i] = (256 *  (1023-i) ) / (1024 - OFS);

   if (table[i]>255) table[i]=255;
   if (table[i]<  0) table[i]=0;
   }
  program_ilut((unsigned int *)table);
  rs232rcv();

  pstr("Input LUT logarithmic (Press any key)\n");
  for(i=  0; i< OFS; i++)
   table[i]=0;
  for(i=OFS; i<1024; i++)
   {
   table[i]  = (int)(85.0*log10((float)(i-OFS+1))+0.5) ;
   }
  program_ilut((unsigned int *)table);
  rs232rcv();


  pstr("Input LUT linear\n");
  for(i=0; i<1024; i++)
   {
   table[i] = (256 * (i - OFS)) / (1024 - OFS);

   if (table[i]>255) table[i]=255;
   if (table[i]<  0) table[i]=0;
   }
  program_ilut((unsigned int *)table);

}

/*********************************************************************/

void program_ilut(unsigned int *table)
{
int i,x ;
volatile unsigned int *ilut  = (volatile unsigned int
*)VIRTX_ILUT;

for(i=0; i<1024 ;i+=4)
  {
  x  =  table[i]   & 0xff;
  x |= (table[i+1] & 0xff) << 8;
  x |= (table[i+2] & 0xff) << 16;
  x |= (table[i+3] & 0xff) << 24;
  *ilut++ = x;
  }
}
/*********************************************************************/
```

Programming the lookup table in logarithmic order

85*log1024 = 255

Input lookup table pointer

7 Grey Value Image Processing

7.1 Edge Detection horizontal or vertical

- required for almost any measurement task.

Examples:

- Object tracking and position / rotation compensation

Process:

- Defining a search direction and area.
- Defining number of search lines / edge positions
- Finding an edge for each search line with pixel or sub pixel accuracy (most common algorithm):
  - Reading the grey values along the search line
  - Calculating the first or second derivative of the grey values
- Edge positions are at the maximum slope of the grey value change or at the 0 values of the second derivatives.
- Calculate best fit line through edge points if required.

Realisation with VC Functions:

- Calculating a search line equation and defining the image variable
- Calculating multiple search lines if required.
- Reading the grey values at the corresponding pixel positions and saving them in a “pixel list”.
- Calculating the derivatives / maxima, etc using the standard functions
- Using standard best fit line functions to calculate edge line.

Demonstration Program:

- Simple Edge Detection example using a horizontal “search line” of variable thickness to find a vertical edge (or vertical line and horizontal edge).
- A thick line serves as a low pass – filtering out small edge defects. Disadvantage: edge positions are inaccurate for non- vertical edges.
- A thin search line may produce inaccurate edge positions for rugged edges, but can also be used for diagonal edges.
- The demo program uses the “projv” function that calculates the vertical projection of an image variable. Accordingly the “projh” can be used for detecting horizontal edges).

```
/*****************************************************************************************************/

Software for smart cameras from Vision Components
-------------------------------------------------

Program: Klaus Schneider, VC

--------------------
/*****************************************************************************************************/
```

Edge Detection, horizontal or vertical

simple edge detection using the projh() or projv() function

```
/*****************************************************************************************************/
#include <math.h>

#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>

#include "flib.h"


/*****************************************************************************************************/
I16  VerticalEdgeDetection      (image *EdgeArea, image *Brightness, image *Difference);
void DisplayGraph               (image *Graph, U32 *values, I16 nr);
void IniDisplay                 (void);

/*****************************************************************************************************/
```

void main(void)
{
I16 response;
I16 x, y, dx, dy, pitch;
image EdgeAreaHor, Brightness, Difference, Area;
 	
// ini variables
 	
// ini working page
pitch = ScrGetPitch;
 	
// Clear Display
IniDisplay();
 	
// ini images
dx = ScrGetColumns;
dy = 8;
x = 0;
y = (DispGetRows/3-dy)/2;
ImageAssign(&EdgeAreaHor, ScrByteAddr(x, y), dx, dy, pitch);
ImageAssign(&Area, OvlByteAddr(x, y), dx, dy, pitch);
frameo(&Area);
 	
dx = ScrGetColumns;
dy = DispGetRows/3;
x = 0;
y = dy;
ImageAssign(&Brightness, ScrByteAddr(x, y), dx, dy, pitch);
ImageAssign(&Area, OvlByteAddr(x, y), dx, dy, pitch);
frameo(&Area);
 	
dx = ScrGetColumns;
	The main program declares 3 image variables for splitting the video output in three horizontal areas. The upper are is showing the edge image, the middle section the grey value change along the seach line, and the lower third shows the first derivative of the grey value change.
	
	Clearing the display (see below) and the overlay page – switching the camera to live mode.
	
	Assignment of image variable for the “sear line” – the line is in fact an image variable with pixel height dy.
	The variable “Area” is used to draw a boundary box for the previously assigned image variable in the overlay.
	
	Assignment of image variable “Brightness” for displaying the grey value changes along the search line and image variable. The variable “Area” is used to draw a boundary box for the previously assigned image variable in the overlay.

```
  dy = DispGetRows/3;
  x  =    0;
  y  = 2*dy;
  ImageAssign(&Difference, ScrByteAddr(x, y), dx, dy, pitch);
  ImageAssign(&Area, OvlByteAddr(x, y), dx, dy, pitch);
  frameo(&Area);

  do
  {
    tpict();


    ////////////////////////////////////////////////////////////////////////////////
    // Find Edge
    ////////////////////////////////////////////////////////////////////////////////

    VerticalEdgeDetection(&EdgeAreaHor, &Brightness, &Difference);

    ////////////////////////////////////////////////////////////////////////////////
    // Wait For Response
    ////////////////////////////////////////////////////////////////////////////////

    print("'q'=quite\n");

    response = rs232rcv();

  }
  while(response != 'q');

}

/*****************************************************************************************************/
```

Assignment of th image variable “Difference”, displaying the first derivative of the grey value projection.
The variable “Area” is used to draw a boundary box for the previously assigned image variable in the overlay.

Calling the edge detection function shown below.

Condition for quitting the program

End of main()

I16 **VerticalEdgeDetection** (image *EdgeArea, image *Brightness, image *Difference)
{
I16 dx, x, max, pos;
U32 *pProj;
 	
dx = EdgeArea->dx;
pProj = (U32 *)sysmalloc(dx, MDATA);
	Memory allocation for the result array of the "projv" function.

// projection of the edge
projv(EdgeArea, pProj);
	Vertical projection of grey values of the "EdgeArea"

DisplayGraph (Brightness, pProj, dx);
for (x=0; x<dx-1; x++)
{
pProj[x] = abs(pProj[x+1] - pProj[x]);
}
	Calling the "Display Graph" (see below) function to display the grey value projection.

DisplayGraph (Difference, pProj, dx-1);
 	
max = pProj[0];
pos = 0;
for (x=1; x<dx-1; x++)
	Calculating and displaying of the first derivative of the grey value projection. Here the difference between each neighbouring pixel is calculated simply by subtracting the pixel values.

{
if (pProj[x] > max)
{
max = pProj[x];
pos = x;
}
}
	Calculating the maximum value of the first derivative (pProj[] Array).

Sysfree(pProj);	**Releasing memory for grey value data**
return(pos); }	Returning the position of the maximum grey value change to the main program.

```
/*****************************************************************************************************/

void IniDisplay (void)
{
  image Temp;

  ScrSetLogPage(ScrGetPhysPage);
  OvlSetLogPage(OvlGetPhysPage);

  // Clear Overlay
  ImageAssign(&Temp, OvlByteAddr(0, 0), DispGetColumns, DispGetRows, ScrGetPitch);
  set(&Temp, 0);

  // Clear Image
  ImageAssign(&Temp, ScrByteAddr(0, 0), DispGetColumns, DispGetRows, ScrGetPitch);
  set(&Temp, 0);

  set_ovlmask(255);

  vmode(vmOvlLive);
}
```

Function for clearing the display and the overlay pages,

```
/*****************************************************************************************************/
void DisplayGraph (image *Graph, U32 *values, I16 nr)
{
  I32 i, x, y, dy, xold, yold;
  I16 max=0, scale=1, factor=1;


  dy    = Graph->dy;
  x     = 0;
  y     = 0;

  for (i=0; i<nr; i++) if (values[i] > max) {max=values[i];}

  if (max > dy) scale  = (max-1)/dy + 1;
  else          factor = dy/max;

  set(Graph, 0);

  for (i=0; i<nr; i++)
  {
    xold = x;
          yold = y;

    x = i;
    y = dy - ((factor * values[i])/scale);

    lined(Graph, xold, yold, x, y, 255);
  }
}

/*****************************************************************************************************/
```

Function for displaying the graphs

Scaling the graph (y-axis) in a way that the maximum Y – value corresponds to the maximum Y- axis level.

7.2 Reading Pixel values with a “Pixel List” or “Direct Addressing”

Address lists (pixel lists)

For one-dimensional image structures (lines, edges, circles), it is often recommendable to use so-called pixel lists. Such a list contains both the (x,y) coordinates of the pixels and the video memory addresses of the pixels. The latter can serve to save processor time. The (x,y) coordinates or addresses can also be stored together with the gray scales of the corresponding pixels.

Direct addressing of pixel
VC Cameras with TI DSP allow direct accessing of pixel values. This increases the read out speed dramatically.

Removing the Advanced Compiler option “-mi100000” increases the execution speed of this program about 10 fold. This setting (under

Program Description: **readpixl.c**

The program is reading Pixels from the DRAM (complete image)
and adds up the grey values of all pixels in four different ways:

2) pixel list - writing pixel values into a list
4) direct addr - reading pixel values directly with pointer

The program calculates and displays the processing time for each method used.

Live Mode is always updating the DRAM – this means the image content is overwritten. For this reason the image capture functions switches to still mode after each image acquisition.

```
/*********************************************************************/
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
/*********************************************************************/

int Brightness2(image *area);
int Brightness4(image *area);

/*********************************************************************/

void main(void)

 {
 int time_ms, brightness;
 image area;


 /* working page = physical page */
 ScrSetLogPage(ScrGetPhysPage);
 ImageAssign(&area,ScrByteAddr(0,0), ScrGetColumns, ScrGetRows,
ScrGetPitch);

 vmode(vmLive);
 print("Get image and calculate the sum of it. Press any key\n");

 rs232rcv();          /* wait for any key */

 tpict();
```

```
/*********************************************************************/
Demo-Software for smart cameras from Vision Components
-----------------------------------------------------

Program: Klaus Schneider,  VC

Program Description:
--------------------

The program is reading Pixels from the DRAM (complete image)
and calculates the brightness of the pixels in 2 different ways:

1) pixel list

2) direct addr

It shows the time for each way.




- Setting the working page to the physical page;
- Assigning the entire image to the image variable area;




- wait for keyboard input

- image aquisition
```

/* **Possibility 2: Read with function pixel list**(); */
/* ---------- */
time_ms = getvar(MSEC);
brightness=**Brightness2(&area);**
time_ms = getvar(MSEC)-time_ms;
if (time_ms<0) time_ms+=1000;
print("pixel list: Pixels=%ld Sum=%ld Time=%d ms\n",(long)(area.dx*area.dy),(long)brightness,time_ms);
 	
/* **Possibility 4: Read with function rpix();** */
/* ---------- */
time_ms = getvar(MSEC);
brightness=**Brightness4(&area)**;
time_ms = getvar(MSEC)-time_ms;
if (time_ms<0) time_ms+=1000;
print("direct addr.: Pixels=%ld Sum=%ld Time=%d ms\n",(long)(area.dx*area.dy),(long)brightness,time_ms);
}
	Measurement of process duration and display of results for all 4 methods.
	Function call for different pixel reading methods.

int Brightness2(image *area)
 	
{
int i, j, dx, dy, sum=0;
int *v_list, *a_list, *xy, *ptemp;
 	
dx =area->dx;
dy =area->dy;
 	
v_list = (int *)vcmalloc(dx);
a_list = (int *)vcmalloc(2*dx);
	Allocate memory for storing grey values
	Allocate memories for Adresses (Long value)

```
/* add lines together */
for(j=0;j<dy;j++)
 {
 xy= a_list;    /* restore pointer */

 /* Generate Address list for one line */
 for (i=0; i<dx; i++)
  {
  *xy++ = i;
  *xy++ = 0;
  }

 xy= a_list;    /* restore pointer */

 ad_calc(dx, a_list, (long *)a_list, (long)j*area->pitch+area->st, area->pitch);
 rp_list(dx, (long *)a_list, v_list);

 ptemp=v_list;

 for (i=0; i<dx; i++)
  {
  sum+=0xff&(*ptemp++);
  }
 }


vcfree(a_list);        /* Don't forget */
vcfree(v_list);        /* Don't forget */

return(sum);
}
```

Calculate coordinates for one line

Address calculation
Read pixel list, write grey values into v_list

Summarise the grey values

Free allocated memory
Free allocated memory

/***/
int Brightness4(image *area)
 	
{
int i, j, dx, dy, sum=0;
long addr, pitch;
unsigned char *ppix;
 	
addr=area->st;
dx=area->dx;
dy=area->dy;
pitch=area->pitch;
 	
for(j=0;j<dy;j++)
{
ppix=(unsigned char *)(addr);
 	
for (i=0; i<dx; i++)
{
sum+=*ppix++;
}
addr+=pitch;
}
 	
return(sum);
}
 	
/***/
	Read out and summarising of all grey values using direct memory access.
	
	Summarising grey values.

7.3 Edge Detection along a Search Line

- required for any measurement task.

Application examples:

- Measurement of distances (between two parallel edges, an edge and a point, etc).
- Measurement of roundness, etc.

Function principle of the "edgediag.c" demo program:

- This demo program is similar to the "edge.c" demo program shown in section 7.1, however it is not restricted to horizontal or vertical edges since it calculates the grey values along an arbitrary search line.
- The search line is defined in 2 steps:
  - Using the "linexy()" function, the coordinates of a parallel line starting at (0;0) are calculated from the start end point specified.
  - The line start point is then moved from the image origin to its correct position using the offset vector calcutated from the line starting point.
- Subsequently the memory address of the search line coordinates are calculated using the "FL_adcalc_U8()" function. This function was formerly called "ad_calc()" – refer to the "readpixl.c" demo program shown in section 7.2. All functions starting with "FL…" have been further optimised and moved from the VCLIB to the "Fastlib" – part of the VC SDK-Ti Setup, containing all fast vector functions.
- The corresponding grey values are read from these memory addresses using the "FL_rplist_U8()" function – formerly "rp_list()".
- The search line is drawn into image memory using the "FL_wpset_U8()" or "FL_wpxor_U8()" functions.
- The grey value graph is displayed along the x-coordinates of the search line. This is done as in the "edge.c" program using the "lined()" function. In contrast to the "edge.c" demo it is not necessary here to scale the grey values – the absolute grey value is displayed.

```
/*****************************************************************************************************/

Software for smart cameras from Vision Components
-------------------------------------------------

Program: Peter Neuhaus, VC

/*****************************************************************************************************/
```

Edge Detection, diagonal

arbitrary direction along a search line defined by start and end coordinates

```
/*****************************************************************/
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
#include <flib.h>
#include <licence.h>
/*****************************************************************/
void main(void)
{
        I32  x, y, x1=1, y1=1, x2=600, y2=340, dx, dy, Color = 128,Pitch, MaxLength ;
        U8 response;

        U8 *pTemp, *LineVal;
        U8 **LineAddr;
        I32 *LineXY;

        image Source, Graph, Area;
        /*****************************************************************/
        void DisplayGraph           (image Graph, U8 *values, I32 nr);
        void IniDisplay             (void);
        /*****************************************************************/
```

Include flib required

Defining the start and end coordinates of the search line

“display graph” similar to section 7.1

“IniDisplay” identical to section 7.1

Init_licence("TXXXXXXXX");	Initialisation of VC SDK-Ti Licence Code required (refer to "Getting Started VC …")

// initialise graph area
IniDisplay();	

dx = ScrGetColumns;
dy = 256;
x = 0;
y = DispGetRows -256;	

ImageAssign(&Graph, ScrByteAddr(x, y), dx, dy,ScrGetPitch); ImageAssign(&Area, OvlByteAddr(x, y), dx, dy, ScrGetPitch); frameo(&Area);	Assignment of image variables
ScrSetLogPage(ScrGetPhysPage); ImageAssign(&Source,ScrByteAddr(0,0),ScrGetColumns,ScrGetRows,ScrGetPitch);	
// // ini values // Pitch = Source.pitch;	
MaxLength = max(Source.dx, Source.dy) + 1;	

LineXY = (I32 *)sysmalloc(2 * MaxLength, MDATA); //memory allocation pixel for coordinates
LineAddr = (U8 **)sysmalloc(MaxLength, MDATA); //memory alocation for address list
LineVal = (U8 *)sysmalloc(MaxLength, MDATA); //memory allocation for grey values
	Memory allocation for pixel coordinate list
	Memory allocation for memory address list
	Memory allocation for grey value list

//
// calculate intersection points and draw corner line0
//
dx = x2 - x1;
dy = y2 - y1;
	Calculating dx and dy for linexy()

```
MaxLength = linexy(dx, dy, LineXY);
pTemp = (U8 *)Source.st + x1 + y1 * Pitch;
FL_adcalc_U8(MaxLength, LineXY, LineAddr, pTemp, Pitch);

////////////////////////////////////////////////////////////////////////////
// take image and read and display grey values along line
////////////////////////////////////////////////////////////////////////////
do
 {
        tpict();

         // read line
        FL_rplist_U8(MaxLength, LineAddr, LineVal);

        // Display grey values on video output
        DisplayGraph(Graph, LineVal, dx-1);

        // draw line into image
        if (Color >= 0)
                FL_wpset_U8 (MaxLength, LineAddr, Color);
                else
                FL_wpxor_U8 (MaxLength, LineAddr, -Color);

        ////////////////////////////////////////////////////////////////////////////
        // Wait For Response
        ////////////////////////////////////////////////////////////////////////////

  print("'q'=quite\n");
  response = rs232rcv();

 }
 while(response != 'q');
```

Moving the line to the correct starting point

Calculation of memory address list

Image acquisition

Reading grey values from the memory addresses and storing them in “lineValue”

Display grey values along the x coordinates of the search line

```
        vmode(0);

        sysfree(LineVal);
        sysfree(LineAddr);
        sysfree(LineXY);
}

/*****************************************************************************************************************/

void IniDisplay (void)
{
        image Temp;

        ScrSetLogPage(ScrGetPhysPage);
        OvlSetLogPage(OvlGetPhysPage);

        // Clear Overlay
        ImageAssign(&Temp, OvlByteAddr(0, 0), DispGetColumns, DispGetRows, ScrGetPitch);
        set(&Temp, 0);

        // Clear Image
        ImageAssign(&Temp, ScrByteAddr(0, 0), DispGetColumns, DispGetRows, ScrGetPitch);
        set(&Temp, 0);

        set_ovlmask(255);

        vmode(vmOvlLive);
}
```

End of main program:

Turing the camera back into live mode and **releasing memory in reverse order**.

IniDisplay function as in section 7.1

Assignement of Image Variable with display size on the **overlay page**.
Seting the image Variable to grey value “0”.

Assignement of Image Variable with display size on **the image page.**
Seting the image Variable to grey value “0”.

```
/*****************************************************************************************************************/

void DisplayGraph(image Graph, U8 *values, I32 nr)
{
  I32 i, x, y, dy, xold, yold;


  dy    = Graph.dy;
  x     = 0;
  y     = 0;


  set(&Graph, 0);

            for (i=0; i<nr; i++)
            {
              xold = x;
                    yold = y;

              x = i;
              y = dy - values[i];

              lined(&Graph, xold, yold, x, y, 255);
            }
}

/*****************************************************************************************************************/
```

DisplayGraph function similar as in section 7.1
However – no scaling is required since grey values range from 0 to 255 only.

8 Binary Image Processing

8.1 Blob Detection using Run Length Code

- Uses same principle as fax
- Usually only used for binary images (also possible with grey scale of 16 steps).

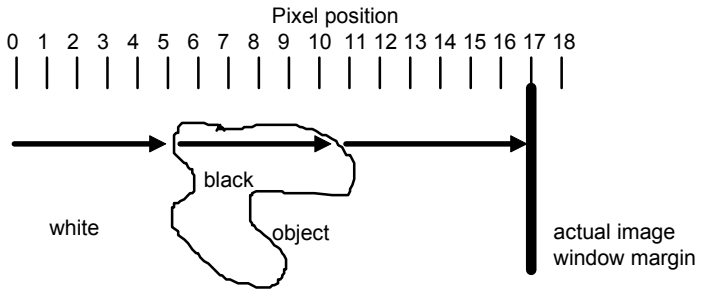
Advantages of image processing of RLC:

- Reduction of information
- Faster processing

Limits:

- Compression Rate / Execution speed depends on amount of image detail (worst case for RLC: chessboard pattern, 1 pixel per module)

Conversion of a binary image into RLC:



Entry	RLC	Remark
1	0	SLC address LSW (0 for unlabelled RLC)
2	0	SLC address MSW (0 for unlabelled RLC)
3	18	dx = end-of-line mark
4	25	dy = number of image/image window lines
5	-1	line begins white (0 if line starts black)
6	5	first change from white to black at position 5
7	11	change from black to white at position 11
8	18	end-of-line mark, because image window margin reached
9	-1	RLC entries for next line

Labelled run length code (SLC)

address	code (U16)	comment
0xA0000000 (RLC)	**0x0018**	SLC address (LSW)
0xA0000002	0xA000	SLC address (MSW)
0xA0000004	18	dx
0xA0000006	25	dy
0xA0000008	-1	line begins white
0xA000000A	5	change from white to black
0xA000000C	11	change from black to white
0xA000000E	18	end_of_line
0xA0000010	-1	Next line begins white
0xA0000012	6	
0xA0000014	12	
0xA0000016	18	end_of_line
0xA0000018 (SLC)	2	number of objects
0xA000001A	0	object 0 (white)
0xA000001C	1	object 1 (black)
0xA000001E	0	
0xA0000020	0 ----	dummy
0xA0000022	0	object 0 (white)
0xA0000024	1	object 1 (black)
0xA0000026	...	

After Labelling (example “sgmt”):

1	**0**	**0**	**0**	**2**
0	**1**	**0**	**2**	**0**
0	**0**	**1?**	**0**	**0**

After Feature Extraction (example “rl_ftr2”):

1	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**
0	**0**	**1**	**0**	**0**

This function also delivers for each blob:

- Blob area
- Centroid coordinates
- Coordinates of bounding box corners, etc

Run Length Code Creation using VCLIB Functions:

```
 U16* rlc; // relevant variable declaration

        /* Declaration of the image variable “a” */
…..ImageAssign(&a, ScrGetCaptPage, DispGetColumns, DispGetRows, DispGetPitch);

        /* Declaration of memory for stroring the Run Length Code */
    rlc = (U16 *)vcmalloc(0x10000); //allocation of 65536 32 bit words (equals 262144 bytes)

        /* binarization of the image with threshold 128 and converting it into RLC */
    rlcmk(&a, 128, rlc, 0x40000); // RLC size in 8 bytes, takes about 4ms for 480x640 with the VC4038
```

Displaying RLC on the video output:

```
    //if(rlcmk(&a, 128, rlc, 0x40000) != NULL)  This line can be used to verify the RLC could be created
    rlcout(&a, rlc, 0, 255); // Display function, takes about 9 ms for 480x640 with the VC4038
    ScrSetDispPage(Screen2); //

    vcfree((int *)rlc);
```

Program example: Blob Detection using Run Length Code

```
/*********************************************************************/

Demo-Software for smart cameras from Vision Components
------------------------------------------------------
Program: Klaus Schneider,  VC
```

Program Description:

The program is searching for black areas in a white background. Black areas are darker than the treshold value. The program gives the size, the position of every black area and shows the position on the monitor.

!!! NEW: PROGRAM HAS BEEN UPDATED TO VCLIB 3.0 !!!

```
/*********************************************************************/

#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>

/*********************************************************************/

#define  EVEN(x)  ((x)&0xFFFE)

/*********************************************************************/

void blobtest     (int x, int y, int dx, int dy, int color);
int  find_objects (image *area, ftr *f, int maxobjects, int th, int display);
void cross_image  (int line, int column, int size, int color);

/*********************************************************************/

int ms;

/*********************************************************************/
```

Process:

1. Taking a grey scale image
2. Binarization into a black and white image with a given threshold
3. Converting the binary image into RLC
4. Labelling the RLC (i.e. identifying connected pixel as blobs)
5. Feature extraction of al blobs and display of results.

Rounds off start address, dx and dy to the nearest even value (this was necessary for image variables used with the VCLIB 2.0)

- Finds blobs and displays results
- RLC Processing and Feature Extraction
- Marks each blob's centroid with a cross

void main(void) { int x, y, dx, dy; char resp;	**Start of Main Program**

dx = EVEN(640);
dy = EVEN(480);
x = EVEN((ScrGetColumns-dx)/2);
y = EVEN((ScrGetRows -dy)/2);
	Defining the position and size of the search window for blob detection (here the camera resolution is given – the search area can be narrowed down in the program “blobtest”.

/* Working Page */
ScrSetLogPage((int)ScrGetPhysPage);
vmode(vmLive);
print("\nblob analyse ('w'=white 'b'=black objects)\n"); resp=rs232rcv();
tpict();
	Setting the Working Page to the Physical Page (= capture = Display Page)
	Turing the camera into Live Mode.
	Printing the user prompt to the terminal program via RS232 or Telnet.
	Acquisition of one image

do
{
tpict();	

if (resp=='w') **blobtest**(x, y, dx, dy, -1); if (resp=='b') **blobtest**(x, y, dx, dy, 0);	Image acquisition in do / while loop.

print("\nblob analyse (%dx%d): t=%dms",dx,dy,ms);
print("\n\nnew run? ('w'=white 'b'=black 'q'=quit)");
resp=rs232rcv();
}
	The blob evaluation depends on the blob specified blob color (white = -1, black = 0).
	
	Printing the results via RS232 or telnet).

while(resp !='q');	Receiving 1 character form the serial / telnet port.

/* live picture at the end */
vmode(vmLive);
}
	End condition for program “q”.

```
/********************** SUBROUTINE ******************************/
#define MAXOBJECTS  100

void blobtest (int x, int y, int dx, int dy, int color)
  {
  int i, j = 0, nobj, threshold = 100 , display=1;
  image area;
  ftr f[MAXOBJECTS];

  /* Search area is the whole image */
  ImageAssign(&area,ScrByteAddr(x,y),dx,dy,ScrGetPitch);

  /* feature extraction */
  nobj = find_objects(&area,f,MAXOBJECTS,threshold,display);
  print("\n\nFound %d white and black objects (-1 means more than %d
objects)",nobj,MAXOBJECTS);

  if (color) print("\nwhite objects:");
  else       print("\nblack objects:");

  for(i=0;i<nobj;i++)
    {
    /* looking for color objects (black=0, white=-1) */
    if (f[i].color==color)
      {
      print("\nArea %d: Size=%d Center of Gravity = %d,%d",
            ++j,f[i].area,f[i].x_center+x,f[i].y_center+y);
cross_image(EVEN((int)f[i].y_center+y),EVEN((int)f[i].x_center+x),10,127);
      }
    }
  }
/*****************************************************************/
```

Turing camera back into live mode when the program is terminated.

— Finds blobs and displays results

“f” is the pointer to the feature list “ftr” (struct array). The dimension of the struct array should correspond the the maximum number of objects / blobs. For details refer to the “rl_ftr2()” section of the VCLIB documentation.

Defining the search area based on the size values specified in the main program.

Calling the subroutine “find_objects” that does the actual binarization, conversion into RLC, labelling of objects and feature extraction.

Selecting only black or only white blobs to be displayed as result, depending on the user selection.

Printing the results via RS232 or Telnet

Marking the centroid of each blob with a cross

```
int find_objects (image *a, ftr *f, int maxobjects, int th, int display)
 {
 long maxlng= 0x020000L;
 U16 *rlc;
 U16 *slc;
 int  objects;
 /* allocate DRAM Memory */
 rlc = (U16 *) sysmalloc((maxlng* sizeof(U16)+1) / sizeof(U32), MDATA);
 if (rlc == NULL) {pstr("DRAM Memory overrun\n"); return(-1);}

 /* create RLC */
 ms = getvar(MSEC);
 slc=rlcmk(a, th, rlc, maxlng);
 if (slc == NULL ) {pstr("RLC overrun\n"); sysfree(rlc); return(-1);}

 /* object labelling */
 if(sgmt(rlc,slc)==0L) {pstr("object number overrun\n"); sysfree(rlc); return(-1);}

 /* feature extraction */
 objects=rl_ftr2(rlc, f, maxobjects);

 ms = getvar(MSEC)-ms;
 if(ms<0) ms+=1000;

 /* display RLC */
 if (display) rlcout(a,rlc,0,255);

 /* free allocation */
 sysfree(rlc);
 return(objects);
 }
/*****************************************************************/
```

— **RLC Processing and Feature Extraction**

Initialising RLC pointers (now U16, previously long, refer to the “Changes..” section at the beginning of the VCLIB3.0 documentation).

Allocating memory as “MDATA” (global variables and heap). Sysmalloc usually allocates memory in 4 byte words – for this reason this memory is divided by 2.

Reading the millisecond timer for measurement of the processing time.

- Binarizing the image and conversion into RLC in one function. The rlcmk returns the next free memory address.

Refer to the corresponding sections in the VCLIB documentation.

Segmentation (labelling) of the RLC.

Feature extraction form labelled RLC. Check for connectedness, centroid-, area- and bounding box calculations.

Time measurement at the end of the RLC operations.

Display of RLC as binary image at the camera video output.

Freeing of memory, allocated for RLC
Returning the results of the blob detection to the calling program “blobtest”.

void **cross_image** (int line, int column, int size, int color)
 	
{
int i;
 	
for (i=-size;i<=size;i++)
{
wpix(color,ScrByteAddr(column ,line+i));
wpix(color,ScrByteAddr(column+i,line));
}
}
 	
/***/
	Drawing cross markers at each blob centroid position.

References:

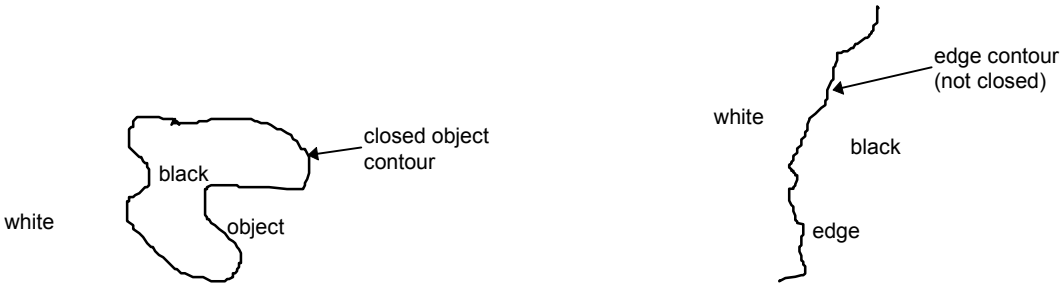
- **Further explanations on RLC:**
  VCLIB3.0: “Run Length Code” section.

- **Further RLC functions for processing unlabelled RLC:**
  VCLIB3.0-> Library Functions-> section: “Programs for processing binary images in unlabelled RLC” (memory management functions, RLC display functions, linking 2 RLC’s with multiple operations, erosion/ dilation of RLC, median filter, storing and retrieving in Flash, feature determination)
- **Further RLC functions for processing labelled RLC (SLC):**
  VCLIB3.0-> Library Functions-> section: “Programs for processing binary images in labelled RLC” (cutting / copying objects from RLC, area calculations, feature calculations for labelled RLC).

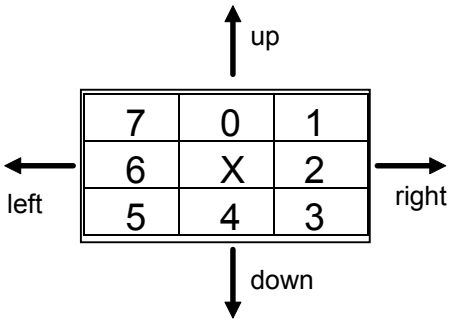
8.2 Working with the Contour function

Contour code (CC)

The contour code (CC) is a method for storing one-dimensional contour data of (closed) object contours or edge data (not closed). Instead of storing the x and y coordinates of all contour pixels, the contour code stores a differential 3 bit information, indicating the direction of movement from one pixel to the next in the contour list. With this data structure only the x and y coordinates of the starting point must be given in order to reconstruct the contour.



different types of contours



0	up
1	up right
2	right
3	down right
4	down
5	down left
6	left
7	up left

contour code values (0 - 7) and the four major directions

example of the CC data format:

byte offset:	**CC**	**Remark**
0	00000004	length (4 contour pixels)
4	00000020	CC status (32: space exhausted)
8	00000018	starting pixel x coordinate
12	00000025	starting pixel y coordinate
16	00	CC: up
17	07	CC: up_left
18	00	CC: up
19	01	CC: up_right

Program Description: cont.c Version VCLIB 2.0

A rectangular ROI is marked in the overly after starting the image.
Put a black object inside this rectangle (white background)
and press a key. You get the contour, the start point and the
contour length (clockwise and counter-clockwise) as a
result. You can see the contour on the monitor. Is the
object is closed inside the rectangle, the contour length
of both direction has the same length.

Example:

Contour Code for BLACK objects (Quite='q')

Press key to start
Start Point x0=165 y0=84
Result of the clockwise contour8() = 01 length=355
Result of the counter-clockwise contour8() = 01 length=355

Press key to start
No black object found

Program Codes	**Comments**

*/
/***/
 	
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
 	
/***/
 	
#define cdisp_dx(a, src, col) cdisp(a, src, col, (void (*)())WPIX);
WPIX(int x, int *adr) {wpix(x,adr);}
 	
/***/
 	
void main(void)
 	
{
int posx= 100, posy=100, dx=300, dy=300; /* search area */
long length=5000; /* maximum of the contour length */
int threshold=125; /* threshold */
 	
int lx, x0=0, y0=0;
int res1=0,res2=0, *rlc, *input, i;
long dest1,dest2,desttemp, addr;
char c;
image Area, Ovl;
	The cdisp function is redefined as cdisp_dx function (identical functionality).

```
  /* Working on Image 1 */
  ScrSetLogPage((int)ScrGetPhysPage);
  OvlSetLogPage((int)OvlGetPhysPage);

  /* clear overlay */
  OvlClearAll;

  /* Overlay on */
  set_ovlmask(255);

  /* define image variable (image) */
  ImageAssign(&Area,ScrByteAddr(posx,posy), dx, dy,
ScrGetPitch);

  /* define image variable (overlay) */
  ImageAssign(&Ovl,OvlBitAddr(posx,posy), dx, dy, OvlGetPitch);

  /* Draw a rectangle */
  frameo(&Ovl);
  vmode(vmOvlLive);

  /* follow contour */
  dest1=DRAMWordMalloc((long)length);                  Memory allocation for clockwise
  dest2=DRAMWordMalloc((long)length);                  and anticlockwise contour data.

  /* allocate memory for one line */
  rlc   = vcmalloc(ScrGetColumns);
  input = vcmalloc(ScrGetColumns);

  print("Contour Code for BLACK objects (Quite='q')\n");


  do
    {
    print("\nPress key to start\n"); c=rs232rcv();
    tpict();

vmode(vmOvlFreeze) ) */                                for higher processor power (better
    vmode(vmOvlLive);                                  for progressive camera is

    if (res1>0) cdisp_dx(&Ovl,dest1,0);                delete old drawings, if exist
    if (res2>0) cdisp_dx(&Ovl,dest2,0);

    /* Get a possible start point: */
    /* - searching direction in this program is from left to right (=
direction 2, please */
    /*   have a look at coutour code (VCLIB, page 8) */
    /* - start points are always black */
    /* - start points in this program have a white left neighbore
```

```
(coming from direction 2) */
   /* - start points shouldn't be on the border of the search area; */
   /*   there must be the possibility to look for all neighbors of the
pixel */

   addr = Area.st/2L+1L; /* not on the border of the searching area
*/
   lx   = dx - 4;
   for (i=2;i<dy;i++)
    {
    addr += ScrGetPitch/2;
    blrdb(lx/2, input, addr);
    rlcmkf(lx, threshold, input, rlc);
    /* find the first change of color */
    if(*(rlc+1)<lx)
     {
     /* select a black point as a start point */
     if (*rlc==-1) x0=*(rlc+1)+2;
     else          x0=*(rlc+1)+1;
     y0=i-1;
     print("Start Point x0=%d y0=%d\n",x0,y0);
     break;
     }
    }

   if (i==dy)
    {
    print("No black object found\n");
    }
   else
    {
    /* clockwise: the variable 'dir' from the function contour8 has to
be the direction from where you come */
    /* possible is ~0,~1,~2,~3,~4,~5,~6,~7 (please have a look at
coutour code (VCLIB, page 8) */
    desttemp = dest1;
    res1 = contour8(&Area,x0,y0,~2,threshold,length,&desttemp);
    print("Result of the        clockwise contour8() = %2d
length=%ld\n",res1,rd32(dest1));
    if (res1>0) cdisp_dx(&Ovl,dest1,255);

    /* counter-clockwise: the variable 'dir' (function contour8) has
to be the direction from where you come from */
    /* possible is 0,1,2,3,4,5,6,7 (please have a look at coutour
code (VCLIB, page 8) */
    desttemp = dest2;
    res2 = contour8(&Area,x0,y0,2,threshold,length,&desttemp);
    print("Result of the counter-clockwise contour8() = %2d
length=%ld\n",res2,rd32(dest2));
    if (res2>0) cdisp_dx(&Ovl,dest2,255);
```

Searching for a start point with the function rlcf()

find contour code in both directon (clockwise and counter-clockwise)

```
    }
   }
  while(c!='q');


  /* have to be in the other way around than allocation (FILO) */
  DRAMWordFree(dest2);
  DRAMWordFree(dest1);

  /* free memory */
  vcfree(input);
  vcfree(rlc);
  }

/*****************************************************************/
```

9 IO Handling

9.1 Working with the digital IOs

Program description: **port.c**

This program display all Inputs (IN0-IN3) and toggles the outputs (OUT0-OUT3).

Program Code	**Comments**
/***/	
#include <vcrt.h>	
#include <vclib.h>	
#include <macros.h>	
#include <sysvar.h>	
/***/	
void main(void)	
{	
int o1=0, o2=0, o3=0, o0=0;	
char response;	
print("Key 0-3: Toggel output OUT0-OUT3\n");	
print("Press any key for displaying Inputs / Outputs\n");	
print("Quit Program: Press 'q'\n\n");	
do	
{	
/* display results */	
print("\n");	
print("OUT0=%d ",o0);	
print("OUT1=%d ",o1);	
print("OUT2=%d ",o2);	
print("OUT3=%d ",o3);	
print(" ");	
print("INP0=%d ",inPLC()&0x01);	InPLC = Status Input 0: 1111&0001
print("INP1=%d ",inPLC()&0x02);	InPLC = Status Input 1: 1111&0010
print("INP2=%d ",inPLC()&0x04);	InPLC = Status Input 2: 1111&0100
print("INP3=%d ",inPLC()&0x08);	InPLC = Status Input 3: 1111&1000

```
   /* wait for key input */
   response=rs232rcv();

   switch (response)
    {
    case '0' :   if(o0) {o0=0; resPLC0();} else {o0=1; setPLC0();}
break;
    case '1' :   if(o1) {o1=0; resPLC1();} else {o1=1; setPLC1();}
break;
    case '2' :   if(o2) {o2=0; resPLC2();} else {o2=1; setPLC2();}
break;
    case '3' :   if(o3) {o3=0; resPLC3();} else {o3=1; setPLC3();}
break;
    }
   }
  while( response != 'q');

  /* live picture at the end */
  vmode(vmLive);
  }

/*********************************************************************/
```

Toggles the PLC outputs according to keyboard inputs “0”, “1”, “2”, “3”

Quit Program with “Q”

9.2 File IO- Camera ID check

Refer to “New Demo Files\ Camera ID\ id.c

This file reads the ID nr. and the camera type. This is useful to ensure that a software can only be used with one camera.

Program Codes	**Comments**

/***/
 	
#include <vcrt.h>
#include <vclib.h>
#include <macros.h>
#include <sysvar.h>
 	
/***/
 	
#define ID_FILE "fd:/#ID.001"
 	
/***/
 	
int GetCameraModel (char *CameraModel);
int GetCameraID (int *CameraID);
int ReadFileLine (char *FileName, int line, int maxlength, char *text);
 	
/***/
 	
void main()
{
char type[10];
int CamId;
 	
GetCameraModel(type);
print("\nCamera typ = %s", type);
 	
GetCameraID(&CamId);
print("\nCamera ID = %d", CamId);
}
	This program reads out the Ascii file “#ID” in the protected flash sector (check with “type #ID”):
	
	$type #ID
	type ascii file
	model: VC2038E
	S/N: 5900193
	DC: 09/01/04 12:42:41
	MAC: 00-06-1F-5A-07-A1
	
	$
	
	Output via RS232/ Telnet:
	
	$id2
	
	Camera typ = VC2038E
	Camera ID = 5900193
	$
	
	Read camera Model, i.e. VC2038E
	
	Read camera serial number (in “ID” file in protected sector on Eprom).

```
/*********************************************************************/
int GetCameraModel (char *CameraModel)
{
  int  res, error=0, line;
  char text[80], *ptext;

  line=0;
  res = ReadFileLine (ID_FILE, line, 80, text);

  if (res>=0)
  {
    ptext = text;

    while(*ptext != 'V') ptext++;
    while(*ptext) *CameraModel++ = *ptext++;
    *CameraModel=0;
  }
  else
  {
    error=-1;
  }

  return(error);
}
/*********************************************************************/
int GetCameraID (int *CameraID)
{
  int  res, error=0, nr=0, line;
  char text[80], *ptext;

  line=1;
  res = ReadFileLine (ID_FILE, line, 80, text);

  if (res>=0)
  {
    ptext = text;

    while(*ptext)
    {
      if (isdigit(*ptext)) nr = nr * 10 + (*ptext & 0x0F);
      ptext++;
    }
    *CameraID = nr;
  }
  else
  {
    error=-1;
  }
  return(error);
}
```

Reading out first line of file “e.g. “model: VC2065”

Jumping to second line of file
Reading out the serial number:
“S/N: 5900193”

```
/*********************************************************************/
int ReadFileLine (char *FileName, int line, int maxlength, char
*text)
{
  int  i, res, error=0;
  FILE *fp;

  /* read datas from file */
  fp = io_fopen(FileName, "r");

  if(fp)
  {
    // skip lines
    for (i=0; i<line; i++)
    {
      do
      {
        res=io_fgetc(fp);
        if (res == '\n') break;
      }
      while(res >= 0);
    }

    // read line
    for (i=0; i<maxlength-1; i++)
    {
      res=io_fgetc(fp);

      if (res <     0) {error = -2; break;}
      if (res == '\n') break;

      *text++=(char)res;
    }

    // end of string
    *text=0;

    io_fclose(fp);
  }
  else
  {
    // File doesn't exist
    error = -1;
  }

  return(error);
}
/*********************************************************************/
```

9.3 Serial RS232 Interface VC4XXX Smart Cameras

```
 void sertest(void)
{
FILE *tty;
//unsigned xbaud=115200;
unsigned xbaud=9600;
char c=0;

print("\nSerial Interface Test press <ESC> to abort\n\n");

INTERFACE_MODE(SERIAL);

tty = fopen("kbd:", (void *)0);       /* open serial device */


print("Set baudrate=%d\n",xbaud);
io_ioctl(tty,IO_IOCTL_SERIAL_SET_BAUD,&xbaud);
xbaud=0;
io_ioctl(tty,IO_IOCTL_SERIAL_SET_FLAGS,&xbaud);

print("\nWriting 'abcdefg' to serial device !\n");
write(tty,"abcdefg",7);

print("\nAny typed char on serial device will be echoed!\n");
print("\nPress ESC on serial terminal to exit!\n");

while(c != 0x1b && rbempty()==-1)
{
  if(io_fstatus(tty)) // test if there are some chars in in buffer
  {
        c=io_fgetc(tty);
        print("c=0x%x\n",c);
  if(c!=0x1b)
        {
          {
                io_fputc(c, tty);
          }
        }
  }
}

fclose(tty);   /* close serial device */

INTERFACE_MODE(ENCODER); // Encoder mode (RS232 disabled)
```

10 Creative Use of the Overlay and the Output Lookup Table

Usually captured images are displayed the same way they are stored on the SDRAM.
However by programming the out put lookup table certain grey values of the camera image are displayed with a certain colour. Programming the Output Lookup table has an effect on the video display only. The captured image itself is not modified.

The following are examples for the use of the output lookup table.

10.1 Example 2: Displaying text and translucent overlays

Step	**Function** of "tutorial \ overlay.c"	Comment
1	void Overlay (void) { image OvlCol1, OvlCol2, TraCol1, TraCol2;	
2	OvlSetLogPage((int)OvlGetPhysPage);	Setting the Working Page
3	OvlClearAll;	Clear Overlay
4	ImageAssign(&OvlCol1, OvlBitAddr(1*OvlGetColumns/5, 1*OvlGetRows/5), OvlGetColumns/5, OvlGetRows/5, OvlGetPitch); ImageAssign(&OvlCol2, OvlBitAddr(3*OvlGetColumns/5, 1*OvlGetRows/5), OvlGetColumns/5, OvlGetRows/5, OvlGetPitch); ImageAssign(&TraCol1, OvlBitAddr(1*OvlGetColumns/5, 3*OvlGetRows/5), OvlGetColumns/5, OvlGetRows/5, OvlGetPitch); ImageAssign(&TraCol2, OvlBitAddr(3*OvlGetColumns/5, 3*OvlGetRows/5), OvlGetColumns/5, OvlGetRows/5, OvlGetPitch);	Define image variables for Overlay – colored squares in display
5	set_overlay_bit(2, 255, 0, 0); /* red */ set_overlay_bit(3, 0, 255, 0); /* green */ set_translucent(1, 0, 0, 255); /* blue translucent */ set_translucent(2, 255, 255, 0); /* yellow translucent */	

6	set(&OvlCol1, 4);
	set(&OvlCol2, 8);
	set(&TraCol1, 1);
	8set(&TraCol2, 2);
		Fill the 4 squares defined in the previous step 4 with color

7	OvlCol1.st-=(12L*(long)OvlGetPitch); OvlCol2.st-=(12L*(long)OvlGetPitch); TraCol1.st-=(12L*(long)OvlGetPitch); TraCol2.st-=(12L*(long)OvlGetPitch);	

8	chprint1("Red",&OvlCol1,1,1,4);
	chprint1("Green",&OvlCol2,1,1,8);
	chprint1("Blue",&TraCol1,1,1,1);
	chprint1("Yellow",&TraCol2,1,1,2);	

9	set_ovlmask(255); }	

Explanation of the steps in above program:

Step 6: Filling the image variables OvlCol1 to TraCol2, defined in step 4 with the solid and translucent colours defined in step 5.
Contrary to the covering colors the overlay colors are assigned directly with their bit level.

Compare these two command lines:

set(&OvlCol2, 8); /*Sets square in “greay value” 8, which was assigned to “green” in step 5*/
set(&TraCol1, 1); /*sets square directly to the bit plane 1, which was assigned to blue translucent in the previous step. Also compare to step 9 of the previous example.

10.2 Example: Displaying inverse grey levels:

Function	Comments
void Inverse (void) { int i; volatile unsigned int *lut;	Declaration of variables: Integer are 4 byte words with TI. RGB values are stored the following way: unused | Red | Green | Blue
lut = (volatile unsigned int *)VIRTX_CLUT;	Operating System pointer to the output lookup table

for(i=255; i>=0 ;i--)
{
*lut++ =
(i) + /* RED */
(i << 8) + /* GREEN */
(i <<16) ; /* BLUE */
}
}
	Inverting the individual image grey levels resulting in a negative image.
	Start: replacing grey level 255 with 0
	Finish: replacing grey level 0 with 255
	Bit shift <<8 to place the green value
	Bit Shift <<16 to place the blue value
	For grey values: R = G = B

Refer to the demo program “**lutout.c**” for further details.

10.3 Example: Displaying False Color:

False color can be used to highlight the contrast or special features of an image. A well known example is the display of temperatures in weather maps or infra red imaging.

See the **lut_out.c** demo program.

Function	Comments
void FalseColor (void) { int i; volatile unsigned int *lut;	
lut = (volatile unsigned int *)VIRTX_CLUT;	Operating System pointer to the output lookup table

/* grey values from 0-15 become color 1 */
for(i=0; i<16 ;i++)
{
*lut++ =
(210) + /* RED */
(255 << 8) + /* GREEN */
(0 <<16) ; /* BLUE */
}
	Image level values are substituted with a certain color. In this example the grey values from 0-255 are replaced with 16 different colors.

As above	Grey level substitution repeated for color 2 to 15

/* grey values from 240-255 become color 16 */
for(; i<256 ;i++)
{
*lut++ =
(255) + /* RED */
(255 << 8) + /* GREEN */
(0 <<16) ; /* BLUE */
}
 	
}	

11 Utility Programs

11.1 Working with System Variables

Refer to the “sysvar.c” demofile under Demofiles\System Variables

Program Description:

How to use the 50 private system variales. The system varibles

are available until power reset.

This program initialisese 50 “private systemvariables” that can be used for instance by the programmer to pass on data between different programs running in parallel (multitasking).

Read out of system variables (VC2038 E):

Multimedia Card available
PLC power ok
DRAM Size=16 MByte
ScrGetColumns=640
ScrGetRows=480
ScrGetPitch=768
DispGetColumns=752
DispGetRows=582
EXUNIT=50

Program Codes	Comments
/***/ #include <vcrt.h> #include <vclib.h> #include <macros.h> #include <sysvar.h> #include <float.h> /***/ void main(void) { int i, index;	

```
  /* 50 free system variables for customer purpose */
  index=getvar(PRIVATE);

  /* initialze system variables */
  for (i=0; i<50; i++) setvar(index+i,i);

  /* read out system variables */
  for (i=0; i<50; i++) print("System Variable Nr. %d = %d
\n",i,getvar(index+i));




  if(getvar(LOWBAT))   pstr("RTC low battery\n");

  if(getvar(MMC))      pstr("Multimedia Card available\n");
  else                 pstr("Multimedia Card is not available\n");

  if(getvar(POWFAIL))  pstr("PLC power down\n");
  else                 pstr("PLC power ok\n");

  print("DRAM Size=%d MByte\n", getvar(DRAMSIZE)/1024/1024);

  print("ScrGetColumns=%d\n", ScrGetColumns);
  print("ScrGetRows=%d\n", ScrGetRows);
  print("ScrGetPitch=%d\n", ScrGetPitch);

  print("DispGetColumns=%d\n", DispGetColumns);
  print("DispGetRows=%d\n", DispGetRows);

  print("EXUNIT=%d\n", getvar(EXUNIT));
  }

/*********************************************************************/
```

Appendix A: Recent Changes of this Manual

Section 6.7.2: Modified pingpong example contains overlay pages:
The demo program switches capture and display pages in order to display only processed images. This new program also updates the overlay. The order of processing has been optimised.

Smart Cameras made in Germany

Visit the Vision Components site **www.vision-components.com** for further information and documentation and software downloads:

Web Site Menu Links	Content
Contact	Distributor list / Enquiry forms
Home	Latest News from VC
Our Company	VC Company Information
News and Events	Trade Show dates VC Publications Sign in for free VC Seminars
VC Network	Description of Partner Companies Application Overview 3rd Party Hard- and SW Products for VC Smart Cameras
Products	Product Overview:
VC Smart Camera Overview	VC44XX High End Camera Series VC40XX Standard Camera Series VC4016 / 18 Entry Level Cameras VC4002L Line Scan Camera VCSBC Single Board Cameras VC20XX Smart Cameras VCSBC Board Cameras VCM + Viscube Camera Sensors
VC Smart Camera Software	
VC Software Development Kit Ti:	VCRT Operating System VCLIB Image Processing Library
VC Special Libraries:	M200 Data Matrix Code Reader VCOCR Text Recognition Library Color Lib
Support:	
Support News (User Registration required)	Tech News – new SW and Documentation
Knowledge Base / FAQ (User Registration required)	Searchable FAQ Database with programming Examples and Demo Code
Download Area	Download of:
Public Download Area (free Access) Registered User Area (User Registration required) Customer Download Area (User- and SW License Registration required)	- Product Brochures - Camera Manuals - Programming Manuals - Training Manuals and Demo Code - Software Updates - Demo Code
RMA Number Form	Form for Allocation of Repair Numbers.

Vision Components GmbH
Ottostr. 2
76275 Ettlingen
Telefon +49 (0)7243 2167-0
Fax +49 (0)7243 2167-11