Pengolahan Citra Pada Mobil Robot

Tabratas Tharom

tharom@yahoo.com

Copyright © Tabratas Tharom 2003

IlmuKomputer.Com

LAMPIRAN B

BEBERAPA FUNGSI PENTING DALAM PENGAMBILAN DAN PENAMPUNGAN GAMBAR

Pada lampiran B beberapa buah fungsi yang sangat berhubungan dengan pengoperasian *Rio* Card sebagai alternatif *frame grabber* yang digunakan untuk mengambil dan menampung gambar. Untuk memahami fungsi tersebut, Anda diharapkan sudah menguasai bahasa pemrograman C/C++ atau minimal mengetahuinya.

1. *Header* Rio yang dilambangkan dengan `<Rio.h>`

```
#if !defined(__RIODOS32_H)
#define __RIODOS32_H
#ifdef __cplusplus
extern "C" {
#endif

/* general functions */

int32 RioOpen(void);
void RioClose(void);
int32  RioGetOverlappedResult(OVERLAPPED  *Overlapped,  int32  *ReturnCode,  BOOL
bWait);
int32 RioSizeOfRiodl(void);
int32 RioCreateRiodl(PRIODL pRiodl);
int32 RioInitialize(int32 BoardId);
```

```
int32 RioSetLed(int32 BoardId, RIO_ON_OFF_MODE LedState);
int32 RioIndexSetLed(int32 BoardIndex, RIO_ON_OFF_MODE LedState);
int32 RioSetBoardId(int32 BoardIndex, int32 BoardId);
int32 RioGetBoardType(int32 BoardId, BOOL *IsBasicVersion);
int32 RioIndexGetBoardType(int32 BoardIndex, BOOL *IsBasicVersion);
int32 RioScatterLock(void *ImageDstPtr,long ImageSize,
                     HANDLE *ImageBufferHandle);
int32 RioScatterUnlock(HANDLE ImageBufferHandle);

/* inputmodule functions */

int32 RioSetInputModule(int32 BoardId, PRIO_MODULE Module);
int32 RioGetInputModule(int32 BoardId, uchar *NrModules, PRIO_MODULE Module);
int32 RioSelectCamera(int32 BoardId, RIO_INPUT_MODULE InputModule,
   int32 Camera);
int32 RioGetCamera(int32 BoardId,RIO_INPUT_MODULE InputModule,
    int32 *Camera);
int32 RioSetInputGain(int32 BoardId,RIO_INPUT_MODULE InputModule,
    int32 Input,uchar Value,RIO_ON_OFF_MODE AutoGainMode);
int32 RioGetInputGain(int32 BoardId, RIO_INPUT_MODULE InputModule,
    int32 Input,uchar *Value,RIO_ON_OFF_MODE *AutoGainMode);
int32 RioSetBrightness(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Value);
int32 RioGetBrightness(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar *Value);
int32 RioSetContrast(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Value);
int32 RioGetContrast(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar *Value);
int32 RioSetSaturation(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Value);
int32 RioGetSaturation(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar *Value);
int32 RioSetHue(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Value);
int32 RioGetHue(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar *Value);
int32   RioGetVideoStatus(int32   BoardId,   RIO_INPUT_MODULE   InputModule,uchar
*Value);

/* high quality b/w functions */

int32 RioSetHqBw(int32 BoardId,float Gain,float Offset);
int32 RioGetHqBw(int32 BoardId,float *Gain,float *Offset);
```

```
/* capture functions */

int32 RioCapture(int32 BoardId,RIO_INPUT_MODULE InputModule,
    BOOL Continuous,BOOL SquarePixels,BOOL TopDown, RECT *SrcRect,
    RECT *DestRect,int16 Pitch,HANDLE ImageBufferHandle,
    OVERLAPPED *pOverlapped);

int32 RioCaptureStop(int32 BoardId, RIO_INPUT_MODULE InputModule);

int32 RioCaptureCancel(int32 BoardId, RIO_INPUT_MODULE InputModule);

/* external interrupt functions */

int32 RioExternalIntTimeOut(int BoardId, RIO_GPIO_PIN GpioPin, long TimeOut);
int32 RioExternalInt(int BoardId,RIO_GPIO_PIN GpioPin,
                     OVERLAPPED *pOverlapped);
int32 RioExternalIntCancel(int BoardId, RIO_GPIO_PIN GpioPin);

/* direct hardware access functions */

int32 RioEepromRead(int32 BoardId,int32 Address,void *Data,int32 Size);
int32 RioEepromWrite(int32 BoardId,int32 Address,void *Data,
    int32 Size);
int32 RioI2cRead(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Reg,uchar *Value);
int32 RioI2cWrite(int32 BoardId,RIO_INPUT_MODULE InputModule,
    uchar Reg,uchar Value);

int32 RioDebiRead(int32 BoardId,int Address,int *Value);
int32 RioDebiWrite(int32 BoardId,int Address,int Value);
int32 RioGpioControl(int32 BoardId,RIO_GPIO_MODE Gpio0,
    RIO_GPIO_MODE Gpio1,RIO_GPIO_MODE Gpio2, RIO_GPIO_MODE Gpio3);
int32 RioGpioWrite(int32 BoardId,uchar Data);
int32 RioGpioRead(int32 BoardId,uchar *Ptr);

#ifdef __cplusplus
}
#endif
#endif
```

2. *Header* `RioCheck` yang dilambangkan dengan `<RioCheck.h>`

```
#if !defined (__RIOCHECK_H)
#define __RIOCHECK_H

#ifdef __cplusplus
extern "C" {
#endif
```

```
BOOL RioCheckGpioPin(RIO_GPIO_PIN GpioPin);
BOOL RioCheckInputModule(int BoardId, RIO_INPUT_MODULE InputModule);
BOOL RioCheckModuleMode(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleInputModule(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleScaler(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleVideoStandard(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleTvOrVtr(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleOutputFormat(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleFieldOrFrame(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModuleCvbsOrYc(int BoardId, PRIO_MODULE Module);
BOOL RioCheckModulePostProcess(int BoardId, PRIO_MODULE Module);
BOOL RioCheckOnOffMode(RIO_ON_OFF_MODE OnOffMode);
BOOL RioCheckGpioMode(RIO_GPIO_MODE GpioMode);
BOOL RioCheckEepromAddress(int Address);
BOOL RioCheckEepromAddressAndDataSize(int Address, int Size);
BOOL RioCheckCameraForModule(int BoardId, RIO_INPUT_MODULE InputModule, int
Camera);
BOOL RioCheckInputGain(uchar Gain);
BOOL RioCheckHqGain(float Gain);
BOOL RioCheckHqOffset(float Offset);
BOOL RioCheckI2cReg(uchar Reg);
BOOL RioCheckRect(RECT *Rect);
BOOL RioCheckPtr(void *Ptr);
BOOL RioCheckFlashLine(uint16 StartLine, uint16 EndLine, uint16 Length);
BOOL RioCheckNotZero(uint32 Val);
BOOL RioCheckTriggerPos(uint32 TriggerPosition, uint32 NrBuffers);
BOOL RioCheckNotNegative(int Val);

int RioCheckSetLed(RIO_ON_OFF_MODE LedState);
int RioCheckGpioControl(RIO_GPIO_MODE Gpio0,
                        RIO_GPIO_MODE Gpio1,
                        RIO_GPIO_MODE Gpio2,
                        RIO_GPIO_MODE Gpio3);
int RioCheckEeprom(int Address,
                   void *Data,
                   int Size);
int RioCheckI2cWrite(int BoardId,
                     RIO_INPUT_MODULE InputModule,
                     uchar Reg,
                     uchar Value);

int RioCheckI2cRead(int BoardId,
                    RIO_INPUT_MODULE InputModule,
                    uchar Reg,
                    uchar *Value);
int RioCheckBCSH(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckSetInputGain(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Input,
                         uchar Value,
                         RIO_ON_OFF_MODE AutoGainMode);
int RioCheckGetInputGain(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Input,
                         uchar *Value,
                         RIO_ON_OFF_MODE *AutoGainMode);
int RioCheckSetInputModule(int BoardId, PRIO_MODULE Module);
int RioCheckSelectCamera(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Camera);
int RioCheckGetCamera(int BoardId,
                      RIO_INPUT_MODULE InputModule,
                      int *Camera);
int RioCheckGetChromaKey(int BoardId,
                         char *VLowerLimit,
                         char *VUpperLimit,
                         char *ULowerLimit,
                         char *UUpperLimit,
                         BOOL *Enabled);
int RioCheckSetHqBw(int BoardId, float Gain, float Offset);
int RioCheckExternalIntTimeOut(int BoardId, RIO_GPIO_PIN GpioPin, long TimeOut);
int RioCheckExternalInt(int BoardId, RIO_GPIO_PIN GpioPin);
int RioCheckExternalIntCancel(int BoardId, RIO_GPIO_PIN GpioPin);
int RioCheckCaptureTimeOut(int BoardId, RIO_INPUT_MODULE InputModule, long
TimeOut);
int RioCheckCapture(int BoardId,
                    RIO_INPUT_MODULE InputModule,
                    BOOL Continuous,
                    BOOL SquarePixels,
                    BOOL TopDown,
                    RECT *SrcRect,
                    RECT *DestRect,
                    int Pitch,
                    HANDLE ImageBufferHandle,
                    OVERLAPPED *pOverlapped);
int RioCheckCaptureCancel(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckCaptureStop(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckPostProcess(int BoardId,
                        PRIO_MODULE Module,
                        BOOL TopDown,
                        RECT *DestRect,
                        int Pitch,
                        HANDLE ImageBufferHandle
                       );
int RioCheckStreamInit(int BoardId,
                       RIO_INPUT_MODULE InputModule,
                       int MicroSecPerCapture,
                       BOOL SquarePixels,
                       BOOL TopDown,
                       RECT *SrcRect,
                       RECT *DestRect,
                       int Pitch);
int RioCheckStreamClose(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckStreamStart(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckStreamStop(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckStreamAddEmptyBuffer(int BoardId,
                                 RIO_INPUT_MODULE InputModule,
                                 PRIO_VIDEO_HDR VideoHdr);
int RioCheckStreamGetFilledBuffer(int BoardId,
                                  RIO_INPUT_MODULE InputModule,
                                  PRIO_VIDEO_HDR *VideoHdr,
                                  OVERLAPPED *pOverlapped);
int RioCheckStreamGetStartTime(int BoardId, RIO_INPUT_MODULE InputModule, DWORD
*StartTime);
int RioCheckTriggerCaptureTimeOut(int BoardId, RIO_INPUT_MODULE InputModule,
long TimeOut);
int RioCheckTriggerCapture(int BoardId,
                           RIO_INPUT_MODULE InputModule,
                           BOOL SquarePixels,
                                                       BOOL TopDown,
                           PRIO_FLASH_PARAMS FlashParams,
                           PRIO_TBUFFER_PARAMS TBufferParams,
                                                  RECT *SrcRect,
                                     RECT *DestRect,
                                                       int Pitch,
                                                       OVERLAPPED *pOverlapped);
int RioCheckTriggerCaptureCancel(int BoardId, RIO_INPUT_MODULE InputModule);
int RioCheckCreateRiodl(PRIODL pRiodl);
int RioCheckGetOverlappedResult(OVERLAPPED *pOverlapped,
                                int *ReturnCode,
                                BOOL bWait);
```

```
#ifdef __cplusplus
}
#endif

#endif /* __RIOCHECK_H */
```

3. *Header* `SaaHandl` yang dilambangkan dengan `<SaaHandl.h>`

```
#if !defined (__SAAHANDL_H)
#define __SAAHANDL_H

#ifdef __cplusplus
extern "C" {
#endif


typedef void*   SAA_HANDLE;

#define SAA_INVALID_HANDLE        0


#ifdef __cplusplus
}
#endif

#endif /* __SAAHANDL_H */
```

4. *Header* `RioHandl` yang dilambangkan dengan `<RioHandl.h>`

```
#if !defined (__RIOHANDL_H)
#define __RIOHANDL_H

#ifdef __cplusplus
extern "C" {
#endif


#include <saahandl.h>

typedef SAA_HANDLE      RIO_HANDLE;


#ifdef __cplusplus
}
#endif

#endif /* __RIOHANDL_H */
```

5. *Header* `Rioll` yang dilambangkan dengan `<Rioll.h>`

```
#if !defined (__RIOLL_H)
#define __RIOLL_H

#ifdef __cplusplus
extern "C" {
#endif


/* irq event list */
#define RIO_EVENT_NONE              0x0000L
#define RIO_EVENT_CAP_HPS_DONE      0x0001L
#define RIO_EVENT_CAP_BRS_DONE      0x0002L
```

```
#define RIO_EVENT_CAP_HPS_BRS_DONE  0x0004L
#define RIO_EVENT_GPIO0             0x0008L
#define RIO_EVENT_GPIO1             0x0010L
#define RIO_EVENT_GPIO2             0x0020L
#define RIO_EVENT_GPIO3             0x0040L
#define RIO_EVENT_FID_IM_0          0x0080L
#define RIO_EVENT_FID_IM_1          0x0100L
#define RIO_EVENT_I2C               0x0200L
#define RIO_EVENT_I2C_ERROR         0x0400L


/* irq error list */
#define RIO_ERR_PPEF            0x00000001L
#define RIO_ERR_PABO            0x00000002L
#define RIO_ERR_PPED            0x00000004L
#define RIO_ERR_VFOU            0x00000008L
#define RIO_ERR_RPS0            0x00000010L
#define RIO_ERR_RPS1            0x00000020L
#define RIO_ERR_SPUR_IRQ        0x00000040L
#define RIO_ERR_IRQ_I2C         0x00000080L

#define RIO_GPIO0_OUT_HIGH      1
#define RIO_GPIO1_OUT_HIGH      2
#define RIO_GPIO2_OUT_HIGH      4
#define RIO_GPIO3_OUT_HIGH      8



typedef struct rio_irq_im_stat_t
{
    ell_bool    CaptureError;
    ell_bool    CaptureDone;
} RIO_IRQ_IM_STAT_T;

typedef struct rio_irq_event_t
{
    int32               IrqEvent;
    int32               IrqErrors;
    RIO_IRQ_IM_STAT_T   Module[RIO_NR_IM];
    ell_bool            Gpio[RIO_NR_GPIO];
    ell_bool            Fid[RIO_NR_IM];
} RIO_IRQ_EVENT_T;



size_t  RioLlHandleSize(void);

int32   RioLlOpen(PDEVICE_EXTENSION DevExt, LOGICAL_ADDR LinMemBase,
    ell_bool MmuEnable, ell_bool IrqEnable);
void    RioLlClose(PDEVICE_EXTENSION DevExt);
int32   RioLlInit(PDEVICE_EXTENSION DevExt);

ell_bool    RioLlIrq(PDEVICE_EXTENSION DevExt, RIO_IRQ_EVENT_T *IrqData);

int32   RioLlDebiWrite(PDEVICE_EXTENSION DevExt, uint16 DebiAddr, uchar Data);
int32   RioLlDebiRead(PDEVICE_EXTENSION DevExt, uint16 DebiAddr, uchar *Ptr);

int32   RioLlEepromWrite(PDEVICE_EXTENSION DevExt, uchar EepromAddr,
    const uchar *Ptr, uint16 n);
int32   RioLlEepromRead(PDEVICE_EXTENSION DevExt, uchar EepromAddr, uchar *Ptr,
    uint16 n);

int32   RioLlCapture(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule,
    ell_bool Continuous, ell_bool SquarePixels, ell_bool TopDown, RECT *SrcRect,
    RECT *DestRect, uint16 InPitch, uint32 VidBasePhysAddr, uint32
VidMmuBasePhysAddr,
    ell_bool CaptureNow, ell_bool DirectlyAfterFlashing);
```

```
void    RioLlCaptureCancel(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE
InputModule);
void    RioLlCaptureEnd(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule);

void RioLlFlashThresholdSetup(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE
InputModule,
    uint16 EndLine, uint16 Length);
int32 RioLlFlash(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule,
    RIO_GPIO_PIN FlashPin, uint16 StartLine, uint16 EndLine, uint16 Length,
    ell_bool FlashNextField);

int32   RioLlSetContrast(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar Value);
int32   RioLlGetContrast(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar *Value);

int32   RioLlSetBrightness(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar Value);
int32   RioLlGetBrightness(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar *Value);

int32   RioLlSetSaturation(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar Value);
int32   RioLlGetSaturation(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar *Value);

int32   RioLlSetHue(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar Value);
int32   RioLlGetHue(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar *Value);

int32 RioLlGetVideoStatus(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, uchar *Value);

int32   RioLlGetBoardType(PDEVICE_EXTENSION DevExt, ell_bool *IsBasicVersion);

int32   RioLlSetInputModule(PDEVICE_EXTENSION DevExt, PRIO_MODULE Module);
int32   RioLlGetInputModule(PDEVICE_EXTENSION DevExt, uchar *NrModules,
    PRIO_MODULE Module);

int32   RioLlSelectCamera(PDEVICE_EXTENSION DevExt,
   RIO_INPUT_MODULE InputModule, int32 Camera);
int32   RioLlGetCamera(PDEVICE_EXTENSION DevExt,
   RIO_INPUT_MODULE InputModule, int32 *Camera);

int32   RioLlSelectCameraFast(PDEVICE_EXTENSION DevExt,
   RIO_INPUT_MODULE InputModule, int32 Camera);

int32   RioLlSetInputGain(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, int32 Input, uchar Value,
    RIO_ON_OFF_MODE AutoGainMode);
int32   RioLlGetInputGain(PDEVICE_EXTENSION DevExt,
    RIO_INPUT_MODULE InputModule, int32 Input, uchar *Value,
    RIO_ON_OFF_MODE *AutoGainMode);

int32   RioLlSetChromaKey(PDEVICE_EXTENSION DevExt, char VLowerLimit, char
VUpperLimit,
    char ULowerLimit, char UUpperLimit, ell_bool Enable);
int32   RioLlGetChromaKey(PDEVICE_EXTENSION DevExt, char *VLowerLimit, char
*VUpperLimit,
    char *ULowerLimit, char *UUpperLimit, ell_bool *Enabled);

int32   RioLlInitHqBw(PDEVICE_EXTENSION DevExt);
int32   RioLlSetHqBw(PDEVICE_EXTENSION DevExt, int32 Gain, int32 Offset);
int32   RioLlGetHqBw(PDEVICE_EXTENSION DevExt, int32 *Gain, int32 *Offset);

int32   RioLlSetLed(PDEVICE_EXTENSION DevExt, RIO_ON_OFF_MODE LedState);
```

```
int32   RioLlI2cWrite(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule,
    uchar SubAddr, uchar Value);
int32   RioLlI2cRead(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule,
    uchar SubAddr, uchar *Value);

int32   RioLlGpioControl(PDEVICE_EXTENSION DevExt, RIO_GPIO_MODE Gpio0,
    RIO_GPIO_MODE Gpio1, RIO_GPIO_MODE Gpio2, RIO_GPIO_MODE Gpio3);
int32   RioLlGpioWrite(PDEVICE_EXTENSION DevExt, uchar Data);
int32   RioLlGpioRead(PDEVICE_EXTENSION DevExt, uchar *Ptr);

void    RioLlEnableExtInt(PDEVICE_EXTENSION DevExt, RIO_GPIO_PIN Pin);
void    RioLlDisableExtInt(PDEVICE_EXTENSION DevExt, RIO_GPIO_PIN Pin);

void    RioLlEnableFidInt(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE
InputModule);
void    RioLlDisableFidInt(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE
InputModule);


#ifdef __cplusplus
}
#endif

#endif /* __RIOLL_H */
```

6. *Header* `Riolldef` yang dilambangkan dengan `<Riolldef.h>`

```
#if !defined (__RIOLLDEF_H)
#define __RIOLLDEF_H
#ifdef __cplusplus
extern "C" {
#endif
/* SAA7110 */
#define RIO_NR_OCF              2
#define RIO_OCF_0               0
#define RIO_OCF_1               1

#define RIO_IDLE                2

#define RIO_OCF_I2C_ADDR0       0x9c /* write addr = even, read addr = odd */
#define RIO_OCF_I2C_ADDR1       0x9e

#define RIO_NR_7110_REGS        53 /* 26 + gap (1a-1f) [=6] + 21 */

#define SAA_7110IDEL            0x00
#define SAA_7110HSYB50          0x01
#define SAA_7110HSYS50          0x02
#define SAA_7110HCLB50          0x03
#define SAA_7110HCLS50          0x04
#define SAA_7110HSYAP50         0x05
#define SAA_7110LUMINANCE       0x06
#define SAA_7110HUE             0x07
#define SAA_7110CKTQUAM         0x08
#define SAA_7110CKTSECAM        0x09
#define SAA_7110PALSS           0x0a
#define SAA_7110SECAMSS         0x0b
#define SAA_7110GAINCHR         0x0c
#define SAA_7110MODE            0x0d
#define SAA_7110IOCLOCK         0x0e
#define SAA_7110CONTROL1        0x0f
#define SAA_7110CONTROL2        0x10
#define SAA_7110CHRGAINREF      0x11
#define SAA_7110CHRSATURATION   0x12
#define SAA_7110LUMCONTRAST     0x13
```

```
#define SAA_7110HSYB60          0x14
#define SAA_7110HSYS60          0x15
#define SAA_7110HCLB60          0x16
#define SAA_7110HCLS60          0x17
#define SAA_7110HSYAP60         0x18
#define SAA_7110LUMBRIGHTNESS   0X19

#define SAA_7110ANALOG1         0X20
#define SAA_7110ANALOG2         0X21
#define SAA_7110MIXER1          0X22
#define SAA_7110CLC21           0X23
#define SAA_7110CLC22           0X24
#define SAA_7110CLC31           0X25
#define SAA_7110CLC32           0X26
#define SAA_7110GAIN1           0X27
#define SAA_7110WHITEPEAK       0X28
#define SAA_7110SYNCBOTTOM      0X29
#define SAA_7110GAIN2           0X2a
#define SAA_7110GAIN3           0X2b
#define SAA_7110MIXER2          0X2c
#define SAA_7110GAININTEGRATION 0X2d
#define SAA_7110VBLKSET         0X2e
#define SAA_7110VBLKRESET       0X2f
#define SAA_7110ADCGAIN         0X30
#define SAA_7110MIXER3          0X31
#define SAA_7110WPINTEGRATION   0X32
#define SAA_7110MIXER4          0X33
#define SAA_7110GAINUL          0X34

#define SAA_7110_MAX_GAIN       63


#define RIO_EEPROM_I2C_ADDR     0xa0
#define RIO_EEPROM_SIZE         128


#define RIO_CVBS_INPUTS 6
#define RIO_YC_INPUTS   3

#define RIO_CAMERA_UNKNOWN  -1

#define RIO_NR_IM_HQ_FF 1


/* Rio Debi registers */
#define RIO_DEBI_SELECT     0
#define RIO_DEBI_PGA        1
#define RIO_DEBI_VOFF       0x100
#define RIO_DEBI_VRT        0x101
#define RIO_DEBI_VRB        0x102
#define RIO_DEBI_VTEST      0x103

/* RIO_DEBI_SELECT */
#define RIO_DEBI_SEL_Y_MASK     0xf3
#define RIO_DEBI_SEL_C_MASK     0xfc
#define RIO_DEBI_SEL_YC_MASK    0xf0
#define RIO_DEBI_SEL_Y0         0x00
#define RIO_DEBI_SEL_C1         0x00
#define RIO_DEBI_SEL_Y2         0x04
#define RIO_DEBI_SEL_C3         0x01
#define RIO_DEBI_SEL_Y4         0x08
#define RIO_DEBI_SEL_C5         0x02
#define RIO_DEBI_SEL_YC0        0x00
#define RIO_DEBI_SEL_YC1        0x05
#define RIO_DEBI_SEL_YC2        0x0a

/* old glue */
```

```
#define RIO_DEBI_SEL_HQBW_MASK  0xef
#define RIO_DEBI_SEL_COLOR      0x00
#define RIO_DEBI_SEL_HQBW       0x10
#define RIO_DEBI_SEL_HQEIA_MASK 0xdf
#define RIO_DEBI_SEL_HQCCIR     0x00
#define RIO_DEBI_SEL_HQEIA      0x20

/* new glue */
#define RIO_DEBI_SEL_A_VS_MASK  0xef
#define RIO_DEBI_SEL_A_VS_NORM  0x00
#define RIO_DEBI_SEL_A_VS_INP   0x10
#define RIO_DEBI_SEL_A_EIA_MASK 0xdf
#define RIO_DEBI_SEL_A_CCIR     0x00
#define RIO_DEBI_SEL_A_EIA      0x20
#define RIO_DEBI_SEL_B_EIA_MASK 0x7f
#define RIO_DEBI_SEL_B_CCIR     0x00
#define RIO_DEBI_SEL_B_EIA      0x80
#define RIO_DEBI_SEL_B_VS_MASK  0xbf
#define RIO_DEBI_SEL_B_VS_NORM  0x00
#define RIO_DEBI_SEL_B_VS_INP   0x40

/* RIO_DEBI_PGA */
#define RIO_DEBI_PGA_MASK       0xfc
#define RIO_DEBI_PGA_0_5        0
#define RIO_DEBI_PGA_1_0        1
#define RIO_DEBI_PGA_2_0        2
#define RIO_DEBI_PGA_4_0        3

#define RIO_DEBI_PGA_A_HQ_MASK  0xfb
#define RIO_DEBI_PGA_A_COLOR    0x00
#define RIO_DEBI_PGA_A_HQBW     0x04

/* default values */
#define RIO_DEBI_PGA_DEF_VAL    RIO_DEBI_PGA_1_0
/*#define RIO_DEBI_VOFF_DEF_VAL   154*/ /* 0.0 (+ 0.2) Volt for 1.0 gain */
/*#define RIO_DEBI_VRT_DEF_VAL    166*/ /* 2.6 Volt */
/*#define RIO_DEBI_VRB_DEF_VAL    153*/ /* 0.6 Volt */
/*#define RIO_DEBI_VRB_0_5        128*/ /* 0.5 Volt */
#define RIO_DEBI_VOFF_DEF_VAL   141 /* 0.0 (+ 0.2) Volt for 1.0 gain */
#define RIO_DEBI_VRT_DEF_VAL    164 /* 2.6 Volt */
#define RIO_DEBI_VRB_DEF_VAL    144 /* 0.6 Volt */
#define RIO_DEBI_VRB_0_5        125 /* 0.5 Volt */

#define RIO_DEBI_VRT_NUM         38
#define RIO_DEBI_VRT_NUM05       41
#define RIO_DEBI_VRT_DEN         32
#define RIO_DEBI_VRT_CONST_10   255
#define RIO_DEBI_VRT_CONST_26   661 /* ~2.6*255 */
#define RIO_DEBI_VRT_CONST_66  1674 /* ~(2.6 + 4.0)*255 */
#define RIO_DEBI_VOFF_CONST_G05 616 /* ~(0.4192 + 2.0)*255 */
#define RIO_DEBI_VOFF_CONST_G10 564 /* ~(0.2096 + 2.0)*255 */
#define RIO_DEBI_VOFF_CONST_G20 536 /* ~(0.1048 + 2.0)*255 */
#define RIO_DEBI_VOFF_CONST_G40 524 /* ~(0.0524 + 2.0)*255 */


#ifdef RIOLL_HQ_INT
#define RIO_HQ_GAIN_MIN          82
#define RIO_HQ_GAIN_MAX        2040
#define RIO_HQ_GAIN_4_0        1020
#define RIO_HQ_GAIN_2_0         510
#define RIO_HQ_GAIN_1_0         255
#define RIO_HQ_GAIN_0_5         128
#define RIO_HQ_OFFSET_MIN      -255
#define RIO_HQ_OFFSET_MAX       255
#else
#define RIO_HQ_GAIN_MIN         0.325F
#define RIO_HQ_GAIN_MAX         8.0F
```

```
#define RIO_HQ_OFFSET_MIN      -1.0F
#define RIO_HQ_OFFSET_MAX       1.0F
#endif


#define RIO_RPS_PAGE_SIZE           1024                /* uint32 */
#define RIO_RPS_PROGRAM_AREA_SIZE   3*RIO_RPS_PAGE_SIZE /* uint32 */


#ifdef __cplusplus
}
#endif

#endif /* __RIOLLDEF_H */
```

7. *Header* `Riollint` yang dilambangkan dengan `<Riollint.h>`

```
#if !defined (__RIOLLINT_H)
#define __RIOLLINT_H

#ifdef __cplusplus
extern "C" {
#endif


typedef struct _RIO_INPUT_MODULE_STATE_T
{
    RIO_MODULE  Module;
    int32       Camera;
} RIO_INPUT_MODULE_STATE_T;


typedef struct _RIO_HANDLE_DATA_T
{
    SAA_7146_HANDLE_DATA_T      Saa7146HandleData;
    ell_bool                    BasicVersion;
    ell_bool                    IrqEnable;
    uchar                       OcfShadow[RIO_NR_OCF][RIO_NR_7110_REGS];
    uchar                       HqTop;
    uchar                       HqOffset;
    RIO_INPUT_MODULE_STATE_T    InputModuleState[RIO_NR_OCF];
    uchar                       ScalerState[RIO_NR_SCALERS];
    ell_bool                    ScalerError[RIO_NR_SCALERS];
    ell_bool                    ChromaKeyEnabled;
    ell_bool                    FidPassUp[RIO_NR_OCF];
} RIO_HANDLE_DATA_T;


int32   RioLlRpsSetup(PDEVICE_EXTENSION DevExt);
void    RioLlRpsCaptureSetup(
    PDEVICE_EXTENSION   DevExt,
    uchar               Ocf,
    RIO_SCALER          Scaler,
    RIO_FIELD_OR_FRAME  FieldOrFrame,
    ell_bool            CaptureOne,
    ell_bool            Rgba,
    ell_bool            DirectlyAfterFlashing,
    uint16              FieldEndLine);
void    RioLlRpsFlashSetup(
    PDEVICE_EXTENSION   DevExt,
    uchar               Ocf,
    RIO_SCALER          Scaler,
    RIO_GPIO_PIN        FlashPin,
    uint16              StartLine,
    uint16              EndLine,
    uint16              Length,
    ell_bool           FlashNextField);

void    RioLlEnableCapture(RIO_HANDLE RioHandle, RIO_SCALER Scaler);


#ifdef __cplusplus
}
#endif

#endif /* __RIOLLINT_H */
```

8. *Header* `Riollocf` yang dilambangkan dengan `<Riollocf.h>`

```
#if !defined (__RIOLLOCF_H)
#define __RIOLLOCF_H

#ifdef __cplusplus
extern "C" {
#endif


int32   RioLlOcfWriteByteTable(RIO_HANDLE RioHandle, uchar Ocf, uchar *Table);
int32   RioLlOcfRegSet(RIO_HANDLE RioHandle, uchar Ocf, uchar SubAddr, uchar
Value);
int32   RioLlOcfRegGet(RIO_HANDLE RioHandle, uchar Ocf, uchar SubAddr, uchar
*Ptr);
int32   RioLlOcfStatus(RIO_HANDLE RioHandle, uchar Ocf, uchar *Ptr);
int32   RioLlOcfVersion(RIO_HANDLE RioHandle, uchar Ocf, uchar *Ptr);
int32   RioLlOcfReset(RIO_HANDLE RioHandle, uchar Ocf);
int32   RioLlOcfInit(RIO_HANDLE RioHandle);
int32   RioLlOcfSetMode(RIO_HANDLE RioHandle, PRIO_MODULE Module);


#ifdef __cplusplus
}
#endif

#endif /* __RIOLLOCF_H */
```

9. *Header* `Dos4gw` untuk membuat *prototype protected mode* pada dos yang dilambangkan dengan `<Dos4gw.h>`

```
#if !defined (__DOS4GW_H)
#define __DOS4GW_H
#ifdef __cplusplus
extern "C" {
#endif
int __cdecl _dx_map_phys(uint16 selector,uint32 phys_addr,
                          uint32 page_cnt,uint32 *offp);
#ifdef __cplusplus
}
#endif
#endif
```

10. *Header* `Riomem` untuk membuat *prototype* pengendali memori pada Rio yang dilambangkan dengan `<Riomem.h>`

```
#if !defined(__RIOMEM_H)
#define __RIOMEM_H
#ifdef __cplusplus
extern "C" {
#endif

int32 MemMapPhysMemory(uint32 PhysicalAddress,
uint32 NumberOfBytes,LOGICAL_ADDR *BaseAddress);
int32 MemUnMapPhysMemory(LOGICAL_ADDR BaseAddress,
uint32 NumberOfBytes);

#if defined(PHARLAP)
#ifndef RIO_BASE_ADDRESS
#define RIO_BASE_ADDRESS    0x1000
#endif

int32 MemScatterLock(uint32 LinAddr, uint32 MemSize,
                     uint32 *GlobalAlias, uint32 *PageBuf);
int32 MemScatterUnlock(uint32 GlobalAlias, uint32 MemSize);
#endif
#ifdef __cplusplus
}
#endif
#endif
```

11. *Header* `Riotype` untuk inisialisasi beberapa fungsi dasar pada penggunaan Rio *card* yang dilambangkan dengan `<Riotype.h>`

```
#if !defined(__RIOTYPE_H)
#define __RIOTYPE_H
#ifdef __cplusplus
extern "C" {
#endif

typedef uint32 DWORD;
typedef void* HANDLE;
typedef int     BOOL;
#if defined(PHARLAP)
typedef DWORD *LOGICAL_ADDR;
#endif
#if defined(DOS4GW)
typedef DWORD far *LOGICAL_ADDR;
#define FALSE   0
#define TRUE    !FALSE
#endif

// not Win32 compatible
typedef struct _OVERLAPPED
{
    uint32  ReturnCode;
    BOOL    EventDone;
} OVERLAPPED;
typedef struct tagRECT
{
    uint32    left;
    uint32    top;
    uint32    right;
    uint32    bottom;
} RECT;

/* PCI defines */
#define RIO_MEMBASE_SIZE  512
#define RIO_VENDOR_ID       0x1131
#define RIO_DEVICE_ID       0x7146
#define RIO_VENDOR_SUB_ID   0x454C
#define RIO_DEVICE_SUB_ID   0x0001
#ifdef __cplusplus
}
#endif
#endif
```

12. *Header* `Riodef` untuk membuat *prototype* fungsi pengenalan iluminasi dan modul warna yang dilambangkan dengan `<Riodef.h>`

```
#ifndef _RIODEF_H
#define _RIODEF_H
#ifdef __cplusplus
extern "C" {
#endif
#define NO_BOARD_ID   0

/* GetVideoStatus defines */
#define RIO_VS_CODE 0x01
#define RIO_VS_ALTD 0x02
#define RIO_VS_WIPA 0x04
#define RIO_VS_GLIM 0x10
#define RIO_VS_FIDT 0x20
#define RIO_VS_HLCK 0x40
```

```
#define RIO_VS_STTC 0x80
typedef enum _RIO_MODULE_MODE
{
    RIO_COLOR,
    RIO_BW,
    RIO_HQ,
    RIO_STEREO_LOCKED,
    RIO_S_FULL_FRAME,
    RIO_FULL_FRAME,
    RIO_RGBA
} RIO_MODULE_MODE;

typedef enum _RIO_ON_OFF_MODE
{
    RIO_ON, RIO_OFF
} RIO_ON_OFF_MODE;

typedef enum _RIO_FIELD_OR_FRAME
{
    RIO_FIELD, RIO_FRAME
} RIO_FIELD_OR_FRAME;

typedef enum _RIO_COLOR_OUTPUT_FORMAT
{
    RIO_YUV16,
    RIO_RGB8,
    RIO_ARGB15,
    RIO_RGAB15,
    RIO_RGB16,
    RIO_RGB24,
    RIO_ARGB32,
    RIO_RGB8_GC,
    RIO_ARGB15_GC,
    RIO_RGAB15_GC,
    RIO_RGB16_GC,
    RIO_RGB24_GC,
    RIO_ARGB32_GC
} RIO_COLOR_OUTPUT_FORMAT;
typedef enum _RIO_RGBA_OUTPUT_FORMAT
{
    RIO_RGBA_RGB24,
    RIO_RGBA_ARGB32,
    RIO_RGBA_RGB24P,
    RIO_RGBA_ARGB32P
} RIO_RGBA_OUTPUT_FORMAT;
```

```
typedef enum _RIO_BW_OUTPUT_FORMAT
{
    RIO_Y1, RIO_Y2, RIO_Y8
} RIO_BW_OUTPUT_FORMAT;
typedef enum _RIO_SCALER
{
    RIO_HPS, RIO_BRS, RIO_HPS_BRS, RIO_NR_SCALERS
} RIO_SCALER;
typedef enum _RIO_INPUT_MODULE
{
    RIO_IM_0, RIO_IM_1, RIO_NR_IM
} RIO_INPUT_MODULE;
typedef enum _RIO_CVBS_OR_YC
{
    RIO_CVBS, RIO_YC
} RIO_CVBS_OR_YC;
typedef enum _RIO_TV_OR_VTR
{
    RIO_TV, RIO_VTR
} RIO_TV_OR_VTR;
typedef enum _RIO_COLOR_VIDEO_STANDARD
{
    RIO_PAL, RIO_SECAM, RIO_NTSC
} RIO_COLOR_VIDEO_STANDARD;
typedef enum _RIO_BW_VIDEO_STANDARD
{
    RIO_CCIR, RIO_EIA
} RIO_BW_VIDEO_STANDARD;
typedef enum _RIO_GPIO_MODE
{
    RIO_GPIO_IGNORE,
    RIO_GPIO_INPUT,
    RIO_GPIO_OUTPUT,
    RIO_GPIO_IRQ_RISE,
    RIO_GPIO_IRQ_FALL,
    RIO_GPIO_IRQ_BOTH
} RIO_GPIO_MODE;
typedef enum _RIO_GPIO_PIN
{
    RIO_GPIO_0,
    RIO_GPIO_1,
    RIO_GPIO_2,
    RIO_GPIO_3,
    RIO_NR_GPIO
```

```
} RIO_GPIO_PIN;
typedef struct _RIODL
{
    int NrBoards;
    int BoardId[1];
} *PRIODL;
typedef struct _RIO_VIDEO_HDR
{
    HANDLE ImageBuffer;
    DWORD User;
    DWORD TimeCaptured;
    DWORD Reserved[4];
} RIO_VIDEO_HDR, *PRIO_VIDEO_HDR;
typedef struct _RIO_FLASH_PARAMS
{
    RIO_GPIO_PIN FlashPin;
    uint16 StartLine;
    uint16 EndLine;
    uint16 Length;
    uint16 VideoOutDelay;
} RIO_FLASH_PARAMS, *PRIO_FLASH_PARAMS;
typedef struct _RIO_TBUFFER_PARAMS
{
    RIO_GPIO_PIN TriggerPin;
    uint32 NrBuffers;
    HANDLE *ImageBuffer;
    uint32 TriggerPosition;
    uint32 *TriggerBufferNr;
    uint16 CapturePeriod;
} RIO_TBUFFER_PARAMS, *PRIO_TBUFFER_PARAMS;
typedef struct _RIO_MODULE_COLOR
{
    RIO_INPUT_MODULE InputModule;
    RIO_SCALER Scaler;
    RIO_COLOR_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_CVBS_OR_YC CvbsOrYc;
    RIO_FIELD_OR_FRAME FieldOrFrame;
    RIO_COLOR_OUTPUT_FORMAT OutputFormat;
} RIO_MODULE_COLOR, *PRIO_MODULE_COLOR;
typedef struct _RIO_MODULE_BW
{
    RIO_INPUT_MODULE InputModule;
    RIO_SCALER Scaler;
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_FIELD_OR_FRAME FieldOrFrame;
    RIO_BW_OUTPUT_FORMAT OutputFormat;
} RIO_MODULE_BW, *PRIO_MODULE_BW;
typedef struct _RIO_MODULE_HQ
{
    RIO_SCALER Scaler;
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_FIELD_OR_FRAME FieldOrFrame;
    RIO_BW_OUTPUT_FORMAT OutputFormat;
} RIO_MODULE_HQ, *PRIO_MODULE_HQ;
typedef struct _RIO_MODULE_STEREO_LOCKED
{
    RIO_INPUT_MODULE InputModule;
    RIO_SCALER Scaler;
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_FIELD_OR_FRAME FieldOrFrame;
    RIO_ON_OFF_MODE PostProcess;
} RIO_MODULE_STEREO_LOCKED, *PRIO_MODULE_STEREO_LOCKED;
typedef struct _RIO_MODULE_S_FULL_FRAME
{
    RIO_INPUT_MODULE InputModule;
    RIO_SCALER Scaler;
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_ON_OFF_MODE PostProcess;
} RIO_MODULE_S_FULL_FRAME, *PRIO_MODULE_S_FULL_FRAME;
typedef struct _RIO_MODULE_FULL_FRAME
{
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_BW_OUTPUT_FORMAT OutputFormat;
} RIO_MODULE_FULL_FRAME, *PRIO_MODULE_FULL_FRAME;
typedef struct _RIO_MODULE_RGBA
{
    RIO_BW_VIDEO_STANDARD VideoStandard;
    RIO_TV_OR_VTR TvOrVtr;
    RIO_FIELD_OR_FRAME FieldOrFrame;
    RIO_ON_OFF_MODE PostProcess;
    RIO_RGBA_OUTPUT_FORMAT OutputFormat;
} RIO_MODULE_RGBA, *PRIO_MODULE_RGBA;
typedef struct _RIO_MODULE
{
    RIO_MODULE_MODE Mode;
    union
    {
        RIO_MODULE_COLOR Color;
        RIO_MODULE_BW Bw;
        RIO_MODULE_HQ Hq;
        RIO_MODULE_STEREO_LOCKED Sl;
        RIO_MODULE_S_FULL_FRAME Sff;
        RIO_MODULE_FULL_FRAME Ff;
        RIO_MODULE_RGBA Rgba;
    } Def;
} RIO_MODULE, *PRIO_MODULE;
#ifdef __cplusplus
}
#endif
#endif /* _RIODEF_H */
```

13. *Header* `Rioerror` untuk inisialisasi kode error pada pemrograman Rio yang dilambangkan dengan `<Rioerror.h>`

```
#ifndef _RIOERROR_H
#define _RIOERROR_H
#define RIO_OK                      0
#define RIO_ERROR                   1
#define RIO_PENDING                 2

/* ll error codes */
#define RIO_ERR_PARAM               0x101L
#define RIO_ERR_SEM_CREATE          0x102L
#define RIO_ERR_RING_ADJUST         0x103L
#define RIO_ERR_I2C_SHORTTIMEOUT    0x201L
#define RIO_ERR_I2C_BUSYTIMEOUT     0x202L
#define RIO_ERR_I2C                 0x203L
#define RIO_ERR_I2C_DTERR           0x204L
#define RIO_ERR_I2C_APERR           0x205L
#define RIO_ERR_I2C_AL              0x206L
#define RIO_ERR_I2C_DRERR           0x207L
#define RIO_ERR_I2C_SPERR           0x208L
#define RIO_ERR_I2C_RING_ADD        0x209L
#define RIO_ERR_I2C_RING_REMOVE     0x20AL
#define RIO_ERR_I2C_SPUR_IRQ        0x20BL
#define RIO_ERR_DEBI_SWTIMEOUT      0x301L
#define RIO_ERR_DEBI_READ           0x302L
#define RIO_ERR_DEBI_WRITE          0x303L

/* drv error codes */
```

```
#define RIO_ERR_PARAMS                      0x800
#define RIO_ERR_INIT                        0x801
#define RIO_OCF_REG_OUT_OF_RANGE            0x802
#define RIO_SCALER_BUSY                     0x803

/* w95drv/ntdrv errors */
#define RIO_CAPTURE_ERROR                   0x1000
#define RIO_INSUFFICIENT_MEMORY             0x1001
#define RIO_LOCK_FAILED                     0x1002
#define RIO_UNLOCK_FAILED                   0x1003
#define RIO_DEBI_READ_ERROR                 0x1004
#define RIO_DEBI_WRITE_ERROR                0x1005
#define RIO_BOARD_ID_READ_ERROR             0x1006
#define RIO_BOARD_ID_WRITE_ERROR            0x1007
#define RIO_VIDEO_OVERFLOW                  0x1008
#define RIO_UNABLE_TO_POST_PROCESS          0x1009
#define RIO_VIDEO_OVERFLOW_NO_POST_PROCESS  0x100A
#define RIO_NO_FILLED_BUFFERS               0x100B
#define RIO_ALREADY_GETTING_BUFFER          0x100C
#define RIO_ALREADY_WAITING_FOR_EXT_INT     0x100D
#define RIO_ALREADY_STARTED_CAPTURE         0x100E

/* dll parameter errors */
#define RIO_INVALID_BOARD_ID                    0x2000
#define RIO_INVALID_GPIO_PIN                    0x2001
#define RIO_INVALID_INPUT_MODULE                0x2002
#define RIO_INVALID_LED_STATE                   0x2003
#define RIO_INVALID_GPIO_MODE                   0x2004
#define RIO_INVALID_EEPROM_ADDRESS              0x2005
#define RIO_INVALID_EEPROM_DATASIZE_FOR_ADDRESS 0x2006
#define RIO_INVALID_AUTO_GAIN_MODE              0x2007
#define RIO_INVALID_INPUT_GAIN                  0x2008
#define RIO_INVALID_INPUT_FOR_INPUT_MODULE      0x2009
#define RIO_INVALID_CAMERA_FOR_INPUT_MODULE     0x200A
#define RIO_INVALID_HQ_GAIN                     0x200B
#define RIO_INVALID_HQ_OFFSET                   0x200C
#define RIO_INVALID_I2C_REG                     0x200D
#define RIO_INVALID_SRC_RECT                    0x200E
#define RIO_INVALID_DEST_RECT                   0x200F
#define RIO_INVALID_FLASH_PIN                   0x2010
#define RIO_INVALID_FLASH_LINE                  0x2011
#define RIO_INVALID_VIDEO_OUT_DELAY             0x2012
#define RIO_INVALID_TBUFFER_PARAMS              0x2013
#define RIO_INVALID_TRIGGER_PIN                 0x2014
#define RIO_INVALID_NR_BUFFERS                  0x2015
```

```
#define RIO_INVALID_IMAGE_BUFFER                0x2016
#define RIO_INVALID_TRIGGER_POSITION            0x2017
#define RIO_INVALID_TRIGGER_BUFFER_NR           0x2018
#define RIO_INVALID_CAPTURE_PERIOD              0x2019
#define RIO_INVALID_MODULE_MODE                 0x201A
#define RIO_INVALID_SCALER                      0x201B
#define RIO_INVALID_VIDEO_STANDARD              0x201C
#define RIO_INVALID_TV_OR_VTR                   0x201D
#define RIO_INVALID_OUTPUT_FORMAT               0x201E
#define RIO_INVALID_FIELD_OR_FRAME              0x201F
#define RIO_INVALID_CVBS_OR_YC                  0x2010
#define RIO_INVALID_POST_PROCESS                0x2021
#define RIO_INVALID_MICROSEC_PER_CAPTURE        0x2022
#define RIO_INVALID_OVERLAPPED                  0x2023
#define RIO_INVALID_EVENT                       0x2024
#define RIO_INVALID_POINTER                     0x2025
#define RIO_INVALID_HANDLE                      0x2026

/* dll errors */
#define RIO_DRIVER_NOT_LOADED                   0x3000
#define RIO_UNABLE_TO_CREATE_FILE               0x3001
#define RIO_UNABLE_TO_MAP_FILE                  0x3002
#define RIO_WAIT_FAILED                         0x3003
#define RIO_TIMEOUT                             0x3004
#define RIO_DEVICE_IO_CONTROL                   0x3005
#define RIO_MEM_ALLOC                           0x3006
#endif /* _RIOERROR_H */
```

14. *Header* `Edef` untuk inisialisasi fungsi umum yang dilambangkan dengan `<Edef.h>`

```
#if !defined (__EDEF_H)
#define __EDEF_H
#ifdef __cplusplus
extern "C" {
#endif

#ifdef FALSE
#define ELL_FALSE     FALSE
#else
#define ELL_FALSE     0L
#endif
#ifdef TRUE
#define ELL_TRUE      TRUE
#else
#define ELL_TRUE      (!ELL_FALSE)
#endif

/*typedef int32       ELL_RETVAL;*/
#define ELL_OK      0
#define ELL_ERROR   (!ELL_OK)
#define ELL_SUCCESS      0
#define ELL_FAILURE      (!ELL_SUCCESS)


#ifdef __cplusplus
}
#endif
#endif /* __EDEF_H */
```

15. *Header* `Etype` untuk mendefinisikan penggunaan Watcom dan arsitektur perangkat yang dilambangkan dengan `<Etype.h>`

```
#if !defined(__ETYPE_H)
#define __ETYPE_H

#if defined(__WATCOMC__) && defined(__386__)
#define __WATCOMC386__
#endif
#if !defined(__386__) && defined(_ARCHITECTURE_)
#if (_ARCHITECTURE_==386) || (_ARCHITECTURE_==486)
#define __386__
#endif
#endif

#if !defined (__FAR__)
#if defined (__386__)
#define __FAR__
#else
#define __FAR__ far
#endif
#endif

typedef unsigned char    uchar;
#if defined (__386__) || defined(_MSC_VER)
  typedef unsigned short uint16;
  typedef unsigned int   uint32;
  typedef short          int16;
  typedef int            int32;

#ifdef BOOL
 typedef BOOL           ell_bool;
```

```
#else
 typedef int32          ell_bool;
#endif
#if defined (_LONG64_)
#if _LONG64_
typedef unsigned         int64;
typedef unsigned long    uint64;
#endif
#endif

#else
  typedef unsigned int   uint16;
  typedef unsigned long  uint32;
  typedef int            int16;
  typedef long           int32;
#endif
#if !defined(_ARCHITECTURE_) && !defined(selector)
#define selector uint16
#endif
#endif
```

16. File `Dos4gw.C` yang menyertakan semua *header* yang ada di atas

```
#include <dos.h>
#include <etype.h>
#include <dos4gw.h>

typedef struct
{
 unsigned int limit15_0  :16;
 unsigned int base15_0   :16;
 unsigned int base23_16  : 8;
 unsigned int type       : 4;
 unsigned int dt         : 1;
 unsigned int dpl        : 2;
 unsigned int p          : 1;
 unsigned int limit19_16 : 4;
 unsigned int avl        : 1;
 unsigned int r0         : 1;
 unsigned int r1         : 1;
 unsigned int g          : 1;
 unsigned int base31_24  : 8;
} cd_dest;

uint16 __cdecl _dpmi_phys_addr_mapping(
               uint32 phys_addr,
                uint32 size,
                uint32 *lin_addr
                )
{
 union REGS regs;
 regs.w.ax = 0x0800;
 regs.w.cx = phys_addr;
 regs.w.bx = phys_addr >> 16;
 regs.w.di = size;
 regs.w.si = size >> 16;
 int386(0x31, &regs, &regs);
 if (regs.w.cflag & 1)
 return(regs.w.ax);
 *lin_addr = (regs.w.bx << 16) | regs.w.cx;
 return(0);
}

uint16 __cdecl _dpmi_set_segment_base(
                uint16 selector,
                uint32 base)
{
 union REGS regs;
 regs.w.ax = 0x0007;
 regs.w.bx = selector;
 regs.w.dx = base;
 regs.w.cx = base >> 16;
 int386(0x31, &regs, &regs);

 if (regs.w.cflag & 1)
  return(regs.w.ax);
 return(0);
}
uint16 __cdecl _dpmi_set_segment_limit(
                uint16 selector,
                uint32 limit)
{
 union REGS regs;
 regs.w.ax = 0x0008;
 regs.w.bx = selector;
 regs.w.dx = limit;
 regs.w.cx = limit >> 16;
 int386(0x31, &regs, &regs);
 if (regs.w.cflag & 1)
  return(regs.w.ax);
 return(0);
```

```
}
uint16 __cdecl _dpmi_get_descriptor(uint16 selector, uchar *descp)
{
 union REGS regs;
 struct SREGS sregs;
 regs.w.ax  = 0x000B;
 regs.w.bx  = selector;
 regs.x.edi = FP_OFF(descp);
 sregs.es   = FP_SEG(descp);
 int386(0x31, &regs, &regs);
 if (regs.w.cflag & 1)
 return(regs.w.ax);
 return(0);
}

int __cdecl _dx_ldt_rd(uint16 selector, uchar *descp)
{
 return(_dpmi_get_descriptor(selector, descp));
}
int __cdecl _dx_map_phys(uint16 selector,   uint32 phys_addr,
                           uint32 page_cnt, uint32 *offp)
{
 uint32 seg_size;
 uint32 lin_addr;
 cd_dest desc;
 int status;

 /* check segment size */
 _dpmi_get_descriptor(selector, (uchar *) &desc);
 seg_size = ((desc.limit19_16 << 16) | desc.limit15_0) + 1;
 if (desc.g)
  seg_size *= 0x1000;
 if (seg_size <= 16)
 {
  status = _dpmi_phys_addr_mapping(phys_addr, page_cnt * 0x1000,
                                     &lin_addr);
  if (status != 0)
   return(130);
  status = _dpmi_set_segment_base(selector, lin_addr);
  if (status != 0)
  return(130);
  status = _dpmi_set_segment_limit(selector, page_cnt * 0x1000 -  1);
  if (status != 0)
  return(130);
  *offp = 0;
 }
 else
  return(130);
 return(0);
}
```

17. Program `Rioboard.c` yang berfungsi mendukung penggunaan *multiple board* pada Rio.

```
#include <stdio.h>
#include <dos.h>

#include <etype.h>
#include <edef.h>

#include <saalnk.h>

#include <riotype.h>

#if defined(__WATCOMC__)
#include <conio.h>
#endif

#include <pciconf.h>

#include <riodef.h>
#include <ellring.h>
#include <saalnk.h>
#include <saa7146.h>
#include <saahandl.h>
#include <saacommn.h>
#include <riohandl.h>

#if defined(PHARLAP)
#include <rioirq.h>
#endif
#include <dglobal.h>

#include <rioboard.h>

#include <rio.h>

#include <riodma.h>
#include <riomem.h>

#include <rioll.h>

#include <rioerror.h>

#if defined(DOS4GW)
#include <dos4gw.h>
#endif


#define EEPROM_ADDR_BOARD_ID_AND_SIG    100

#define RIO_BOARD_ID_AND_SIG_SIZE   3

#define RIO_BOARD_ID_SIG1_RPOS      0
#define RIO_BOARD_ID_RPOS           1
#define RIO_BOARD_ID_SIG2_RPOS      2

#define RIO_BOARD_ID_SIG1           0xC3
```

```
#define RIO_BOARD_ID_SIG2           0x3C


#define RIO_EEPROM_ADDR_GAINDIFF_AND_SIG    103

#define RIO_GAIN_DIFF_AND_SIG_SIZE  3

#define RIO_GAIN_DIFF_SIG1_RPOS     0
#define RIO_GAIN_DIFF_RPOS          1
#define RIO_GAIN_DIFF_SIG2_RPOS     2

#define RIO_GAIN_DIFF_SIG1          0xC3
#define RIO_GAIN_DIFF_SIG2          0x3C


#define EEPROM_ADDR_SUB_SYSTEM      0

#define RIO_SUB_SYSTEM_SIZE         4

#define RIO_SUB_SYSTEM_ID_HIGH_RPOS     0
#define RIO_SUB_SYSTEM_ID_LOW_RPOS      1
#define RIO_SUB_SYSTEM_VID_HIGH_RPOS    2
#define RIO_SUB_SYSTEM_VID_LOW_RPOS     3

// Rio = 'EL' 0x0001
#define RIO_SUB_SYSTEM_VID_HIGH     0x45
#define RIO_SUB_SYSTEM_VID_LOW      0x4c
#define RIO_SUB_SYSTEM_ID_HIGH      0x00
#define RIO_SUB_SYSTEM_ID_LOW       0x01


#define DelayOneUsec() inp(0x80)


// configuration of one board
typedef struct
{
    BOOL BoardOk;
    int32 BoardId;
    long CaptureTimeOut[RIO_NR_IM];
    long ExtIntTimeOut[RIO_NR_GPIO];
    RIO_IRQ_EVENT_T IrqData;
    PDEVICE_EXTENSION DevExt;
    void* NextBoardOnSameIrq;
    void* PrevBoardOnSameIrq;
} BOARD_CONF, *PBOARD_CONF;


//static  char    Version[] = "rioboard.c 1.4 19980925";


int32 NrRioOpen = 0;     // nr of calls made to RioOpen
int32 NrBoards  = 0;     // nr of RIOs found in the system
int32 NrOkBoards = 0;    // nr of working RIOs
PBOARD_CONF BoardConf;



void GpioIsr(PDEVICE_EXTENSION DevExt, RIO_IRQ_EVENT_T *IrqData, RIO_GPIO_PIN
Pin)
{
    PEXT_INT_T ExtInt;

    RioLlDisableExtInt(DevExt, Pin);

    ExtInt = &DevExt->ExtInt[Pin];
    if ( ExtInt->Overlapped != NULL )
    {
        ExtInt->Overlapped->ReturnCode = RIO_OK;
        ExtInt->Overlapped->EventDone = TRUE;

        ExtInt->Overlapped = NULL;
    }
}


void CaptureDoneIsr(PDEVICE_EXTENSION DevExt, RIO_INPUT_MODULE InputModule)
{
    PMODULE_T Module;
    OVERLAPPED *CaptureOverlapped;

    Module = &(DevExt->Module[InputModule]);

    CaptureOverlapped = Module->CaptureOverlapped;
    Module->CaptureOverlapped = NULL;

    if (CaptureOverlapped != NULL)
    {
        if (Module->CaptureError == FALSE)
            CaptureOverlapped->ReturnCode = RIO_OK;
        else
            CaptureOverlapped->ReturnCode = RIO_VIDEO_OVERFLOW;

        CaptureOverlapped->EventDone = TRUE;
    }
}



#if defined(PHARLAP)

void RioIrqHandler(PBOARD_CONF BoardConf)
{
    PDEVICE_EXTENSION DevExt;
    RIO_IRQ_EVENT_T* IrqData;
    ell_bool IrqWasPending;
    uchar Isr;
    int i;

    DevExt = BoardConf->DevExt;

    if (IrqIsSpurious(DevExt->IrqRegister) != ELL_FALSE) /* slave */
    {
        /* interrupt 7 and 15 can be spurious */
        outp(SLAVE_8259L, ISRREG);  /* select ISR register   */
        EllDelay(1);
        Isr = inp(SLAVE_8259L);     /* read ISR reg via OCW3 */
        EllDelay(1);
        if (Isr == 0) /* no other interrupts */
        {
            outp(MASTER_8259L, NONSEOI); /*0010 0xxx : non-specific EOI */
            EllDelay(1);
        }
    }
    else
    {
        /* not spurious */
        while (BoardConf != NULL)
        {
            DevExt = BoardConf->DevExt;
            IrqData = &BoardConf->IrqData;

            IrqWasPending = RioLlIrq(DevExt, IrqData);
            if (IrqWasPending != ELL_FALSE)
            {
                if (IrqData->IrqEvent & (RIO_EVENT_I2C | RIO_EVENT_I2C_ERROR))
                {
                   if (IrqData->IrqEvent & RIO_EVENT_I2C)
                   {
                      FlCoI2cTransferContinue(DevExt->BoardHandle);
                   } else
                   {
                      FlCoI2cTransferError(DevExt->BoardHandle);
                   }
                }

                for (i = 0; i < RIO_NR_IM; i++)
                {
                    if (IrqData->Module[i].CaptureDone != ELL_FALSE)
                    {
                        if (IrqData->Module[i].CaptureError == ELL_FALSE)
                            DevExt->Module[i].CaptureError = FALSE;
                        else
                            DevExt->Module[i].CaptureError = TRUE;

                        CaptureDoneIsr(DevExt, i);
                    }
                }

                for (i = 0; i < RIO_NR_GPIO; i++)
                {
                    if (IrqData->Gpio[i] != ELL_FALSE)
                    {
                        GpioIsr(DevExt, IrqData, i);
                    }
                }
            }

            BoardConf = (PBOARD_CONF) BoardConf->NextBoardOnSameIrq;
        }

        IrqSendEoi(DevExt->IrqRegister);
    }
}


// irq support for RIO_MAXIMUM_DEVICES rio boards
IRQHANDLER_ATTRIB RioIrqHandlerStub0()
{
    RioIrqHandler(&BoardConf[0]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub1()
{
    RioIrqHandler(&BoardConf[1]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub2()
{
    RioIrqHandler(&BoardConf[2]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub3()
{
    RioIrqHandler(&BoardConf[3]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub4()
{
    RioIrqHandler(&BoardConf[4]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub5()
{
    RioIrqHandler(&BoardConf[5]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub6()
{
    RioIrqHandler(&BoardConf[6]);
}

IRQHANDLER_ATTRIB RioIrqHandlerStub7()
{
    RioIrqHandler(&BoardConf[7]);
}

#endif // PHARLAP


#if defined(DOS4GW)

void SearchBoardsForOverlapped(OVERLAPPED* Overlapped, PBOARD_CONF *pBoardConf)
{
    PDEVICE_EXTENSION DevExt;
    int i;
    int j;

    for (j = 0; j < NrBoards; j++)
    {
        *pBoardConf = &BoardConf[j];
        DevExt = (*pBoardConf)->DevExt;

        for (i = 0; i < RIO_NR_IM; i++)
        {
            if (DevExt->Module[i].CaptureOverlapped == Overlapped)
            {
                break;
            }
        }
        if (i < RIO_NR_IM)
            break;

        for (i = 0; i < RIO_NR_GPIO; i++)
        {
            if (DevExt->ExtInt[i].Overlapped == Overlapped)
            {
                break;
            }
        }
        if (i < RIO_NR_GPIO)
            break;
    }
}



void RioIrqHandler(PBOARD_CONF BoardConf)
{
    PDEVICE_EXTENSION DevExt;
    RIO_IRQ_EVENT_T* IrqData;
    ell_bool IrqWasPending;
    int i;

    DevExt = BoardConf->DevExt;
    IrqData = &BoardConf->IrqData;
    IrqWasPending = RioLlIrq(DevExt, IrqData);

    if (IrqWasPending != ELL_FALSE)
    {
        for (i = 0; i < RIO_NR_IM; i++)
        {
            if (IrqData->Module[i].CaptureDone != ELL_FALSE)
            {
                if (IrqData->Module[i].CaptureError == ELL_FALSE)
                    DevExt->Module[i].CaptureError = FALSE;
                else
                    DevExt->Module[i].CaptureError = TRUE;

                CaptureDoneIsr(DevExt, i);
            }
        }

        for (i = 0; i < RIO_NR_GPIO; i++)
        {
            if (IrqData->Gpio[i] != ELL_FALSE)
            {
                GpioIsr(DevExt, IrqData, i);
            }
        }
    }
}

#endif


int32 RioOpenBoards(void)
{
    int32 RetVal = RIO_OK;
    int32 i;
    int32 j;
    LOGICAL_ADDR LinMemBase;

    if (NrRioOpen == 0)
    {
        NrBoards = GetPciDeviceCount(RIO_VENDOR_ID, RIO_DEVICE_ID);
        if (NrBoards == 0 || NrBoards > RIO_MAXIMUM_DEVICES)
        {
            return (RIO_DRIVER_NOT_LOADED);
        }
        NrOkBoards = 0;

        BoardConf = (PBOARD_CONF) EllCalloc(NrBoards * sizeof(BOARD_CONF));
        if (BoardConf == NULL)
        {
            return (RIO_MEM_ALLOC);
        }

        for (i = 0; i < NrBoards; i++)
        {
            conf_space_hdrt PciData;

            PDEVICE_EXTENSION DevExt = (PDEVICE_EXTENSION) EllCalloc(
                sizeof(DEVICE_EXTENSION) + RioLlHandleSize());
            if (DevExt == NULL)
            {
                return (RIO_MEM_ALLOC);
            }
            BoardConf[i].DevExt = DevExt;
            BoardConf[i].NextBoardOnSameIrq = NULL;
            BoardConf[i].PrevBoardOnSameIrq = NULL;
            for (j = 0; j < RIO_NR_IM; j++)
            {
                BoardConf[i].CaptureTimeOut[j] = RIO_DEFAULT_CAPTURE_TIMEOUT;
                DevExt->Module[j].CaptureOverlapped = NULL;
            }
            for (j = 0; j < RIO_NR_GPIO; j++)
            {
                BoardConf[i].ExtIntTimeOut[j] = RIO_DEFAULT_EXTINT_TIMEOUT;
                DevExt->ExtInt[j].Overlapped = NULL;
            }

            // Allocate two pages, due to rps bug in V1c should restrict address
to below 16MB
            DevExt->RPSBufferAddress = EllCalloc(sizeof(uint32) *
                RIO_RPS_PROGRAM_AREA_SIZE);

            if (((uint32) DevExt->RPSBufferAddress) & 0xfff)
            {
                DevExt->RPSBuffer[0].VirtualAddress = (uint32 *)
                    (((uint32)DevExt->RPSBufferAddress) & 0xfffff000);
                DevExt->RPSBuffer[0].VirtualAddress += RIO_RPS_PAGE_SIZE;
            } else
            {
                DevExt->RPSBuffer[0].VirtualAddress = (uint32 *) DevExt-
>RPSBufferAddress;
            }
            DevExt->RPSBuffer[1].VirtualAddress =
                DevExt->RPSBuffer[0].VirtualAddress + RIO_RPS_PAGE_SIZE;
            EllLinAddrToPhys((uint32 *) &DevExt->RPSBuffer[0].VirtualAddress,
                &DevExt->RPSBuffer[0].LogicalAddress);
            EllLinAddrToPhys((uint32 *) &DevExt->RPSBuffer[1].VirtualAddress,
                &DevExt->RPSBuffer[1].LogicalAddress);

            if (DevExt->RPSBuffer[0].LogicalAddress > 0xffffff ||
                DevExt->RPSBuffer[1].LogicalAddress > 0xffffff)
            {
                int32 BufNum = 0;
                uint32** UnusableBufs = NULL;
                uint32 UnusableBufSize = 1024;
                uint32 *VirtualAddress;
                PHYSICAL_ADDRESS LogicalAddress;

                EllFree(DevExt->RPSBufferAddress);

                // search for right type
                UnusableBufs = EllCalloc(sizeof(uint32*) * UnusableBufSize);
                if (UnusableBufs == NULL)
                {
                    return (RIO_MEM_ALLOC);
                }

                while (1)
                {
                    UnusableBufs[BufNum] = EllCalloc(sizeof(uint32) *
RIO_RPS_PROGRAM_AREA_SIZE);
                    if (UnusableBufs[BufNum] == NULL)
                    {
                        while (--BufNum >= 0)
                        {
                            EllFree(UnusableBufs[BufNum]);
                        }
                        return (RIO_UNABLE_TO_MAP_FILE);
                    }

                    if (((uint32) UnusableBufs[BufNum]) & 0xfff)
                    {
                        VirtualAddress = (uint32 *)
```

```
                            (((uint32)UnusableBufs[BufNum]) & 0xfffff000);
                        VirtualAddress += RIO_RPS_PAGE_SIZE;
                    } else
                    {
                        VirtualAddress = (uint32 *) UnusableBufs[BufNum];
                    }
                    EllLinAddrToPhys((uint32 *) &VirtualAddress,
&LogicalAddress);

                    if (LogicalAddress <= 0xffffff)
                    {
                        VirtualAddress = VirtualAddress + RIO_RPS_PAGE_SIZE;
                        EllLinAddrToPhys((uint32 *) &VirtualAddress,
&LogicalAddress);

                        if (LogicalAddress <= 0xffffff)
                        {
                            // okay
                            DevExt->RPSBuffer[0].VirtualAddress = VirtualAddress
- RIO_RPS_PAGE_SIZE;
                            DevExt->RPSBuffer[1].VirtualAddress =
VirtualAddress;
                            EllLinAddrToPhys((uint32 *) &DevExt-
>RPSBuffer[0].VirtualAddress,
                                &DevExt->RPSBuffer[0].LogicalAddress);
                            EllLinAddrToPhys((uint32 *) &DevExt-
>RPSBuffer[1].VirtualAddress,
                                &DevExt->RPSBuffer[1].LogicalAddress);

                            while (--BufNum >= 0)
                            {
                                EllFree(UnusableBufs[BufNum]);
                            }

                            break;
                        }
                    }

                    BufNum++;
                    if (BufNum == UnusableBufSize)
                    {
                        uint32** TempUnusableBufs = UnusableBufs;

                        UnusableBufSize += 1024;
                        UnusableBufs = EllCalloc(sizeof(uint32*) *
UnusableBufSize);
                        if (UnusableBufs == NULL)
                        {
                            while (--BufNum >= 0)
                            {
                                EllFree(TempUnusableBufs[BufNum]);
                            }
                            return (RIO_UNABLE_TO_MAP_FILE);
                        }

                        EllMemCpy(UnusableBufs, TempUnusableBufs,
sizeof(uint32*) * BufNum);
                        EllFree(TempUnusableBufs);
                    }
                }
            }

            if (DevExt->RPSBuffer[0].LogicalAddress == NULL ||
                DevExt->RPSBuffer[1].LogicalAddress == NULL)
            {
                return (RIO_UNABLE_TO_MAP_FILE);
            }
            if (GetPciDeviceData(i, RIO_VENDOR_ID, RIO_DEVICE_ID, &PciData) ==
ELL_FALSE)
            {
                // shouldn't happen
                return (RIO_DEVICE_IO_CONTROL);
            }

            DevExt->PhysMemBase = PciData.base_addr[0];
            DevExt->IrqRegister = PciData.interrupt_line;

            DevExt->BoardHandle = (RIO_HANDLE) (DevExt + 1);

#if defined(PHARLAP)
            LinMemBase = (LOGICAL_ADDR) RIO_BASE_ADDRESS + i*0x1000;
#endif
            RetVal = MemMapPhysMemory(DevExt->PhysMemBase, RIO_MEMBASE_SIZE,
&LinMemBase);
            if (RetVal != RIO_OK)
            {
                return (RetVal);
            }

            DevExt->LinMemBase = LinMemBase;

#if defined(PHARLAP)
// irq support for RIO_MAXIMUM_DEVICES rio boards
            for (j = i - 1; j >= 0; j--)
            {
                if (BoardConf[j].DevExt->IrqRegister == DevExt->IrqRegister &&
                    BoardConf[j].BoardOk != FALSE)
                {
                    BoardConf[j].NextBoardOnSameIrq = (void*) &BoardConf[i];
                    BoardConf[i].PrevBoardOnSameIrq = (void*) &BoardConf[j];
                    break;
                }
            }

            if (j < 0) /* new irq */
            {
                switch (i)
                {
                    case 0:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub0, &DevExt-
>OldIrqHandler);
                        break;
                    case 1:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub1, &DevExt-
>OldIrqHandler);
                        break;
                    case 2:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub2, &DevExt-
>OldIrqHandler);
                        break;
                    case 3:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub3, &DevExt-
>OldIrqHandler);
                        break;
                    case 4:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub4, &DevExt-
>OldIrqHandler);
                        break;
                    case 5:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub5, &DevExt-
>OldIrqHandler);
                        break;
                    case 6:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub6, &DevExt-
>OldIrqHandler);
                        break;
                    case 7:
                        IrqInstallHandler(DevExt->IrqRegister,
                            (FARPTR) RioIrqHandlerStub7, &DevExt-
>OldIrqHandler);
                        break;
                    default:
                        return (RIO_UNABLE_TO_CREATE_FILE);
                }
            }

            /* mmu enabled, irq's enabled */
            RetVal = RioLlOpen(DevExt, LinMemBase, ELL_TRUE, ELL_TRUE);

#elif defined(DOS4GW)
            /* mmu not enabled, irq's not enabled */
            RetVal = RioLlOpen(DevExt, LinMemBase, ELL_FALSE, ELL_FALSE);
#endif

            if (RetVal == RIO_OK)
            {
                RetVal = GetBoardId(DevExt, &(BoardConf[i].BoardId));
            }

            if (RetVal == RIO_OK)
            {
                RetVal = GetGainDiff(DevExt, &DevExt->GainDiff);
            }

            if (RetVal != RIO_OK)
            {
                BoardConf[i].BoardOk = FALSE;
#if defined(PHARLAP)
                if (BoardConf[i].PrevBoardOnSameIrq == NULL)
                {
                    IrqRemoveHandler(DevExt->IrqRegister, DevExt-
>OldIrqHandler);
                } else
                {
                    ((PBOARD_CONF) BoardConf[i].PrevBoardOnSameIrq)-
>NextBoardOnSameIrq = NULL;
                    BoardConf[i].PrevBoardOnSameIrq = NULL;
                }
#endif
                EllFree(DevExt->RPSBufferAddress);
                EllFree(DevExt);
            } else
            {
                BoardConf[i].BoardOk = TRUE;
                NrOkBoards++;
            }
        }

        if (NrOkBoards == 0)
        {
            EllFree(BoardConf);
            return (RIO_DRIVER_NOT_LOADED);
        }
```

```
#if defined(PHARLAP)
        (void) DMAInitPool(32);
#endif
    }

   NrRioOpen++;

   return (RetVal);
}


void RioCloseBoards(void)
{
   int32 i;
   int32 RetVal;

   NrRioOpen--;

   if (NrRioOpen == 0)
   {
        for (i = NrBoards - 1; i >= 0; i--)
        {
            if (BoardConf[i].BoardOk != FALSE)
            {
                PDEVICE_EXTENSION DevExt = (PDEVICE_EXTENSION)
BoardConf[i].DevExt;

                RioLlClose(DevExt);

#if defined(PHARLAP)
                if (BoardConf[i].PrevBoardOnSameIrq == NULL)
                {
                    IrqRemoveHandler(DevExt->IrqRegister, DevExt-
>OldIrqHandler);
                } else
                {
                    ((PBOARD_CONF) BoardConf[i].PrevBoardOnSameIrq)-
>NextBoardOnSameIrq = NULL;
                    BoardConf[i].PrevBoardOnSameIrq = NULL;
                }
#endif
                RetVal = MemUnMapPhysMemory(DevExt->LinMemBase,
RIO_MEMBASE_SIZE);

                EllFree(DevExt->RPSBufferAddress);
                EllFree(DevExt);
            }
        }
        EllFree(BoardConf);
   }
}


int32 SetBoardId(PDEVICE_EXTENSION DevExt, int32 BoardId)
{
    uchar BoardIdAndSig[RIO_BOARD_ID_AND_SIG_SIZE];
    uchar SubSystemId[RIO_SUB_SYSTEM_SIZE];

    // write SubSystem ID's
    SubSystemId[RIO_SUB_SYSTEM_ID_HIGH_RPOS ] = RIO_SUB_SYSTEM_ID_HIGH ;
    SubSystemId[RIO_SUB_SYSTEM_ID_LOW_RPOS  ] = RIO_SUB_SYSTEM_ID_LOW  ;
    SubSystemId[RIO_SUB_SYSTEM_VID_HIGH_RPOS] = RIO_SUB_SYSTEM_VID_HIGH;
    SubSystemId[RIO_SUB_SYSTEM_VID_LOW_RPOS ] = RIO_SUB_SYSTEM_VID_LOW ;

    if (RioLlEepromWrite(DevExt,
                         EEPROM_ADDR_SUB_SYSTEM,
                         SubSystemId,
                         RIO_SUB_SYSTEM_SIZE) != RIO_OK)
    {
        return (RIO_BOARD_ID_WRITE_ERROR);
    }

    BoardIdAndSig[RIO_BOARD_ID_SIG1_RPOS] = RIO_BOARD_ID_SIG1;
    BoardIdAndSig[RIO_BOARD_ID_RPOS     ] = (uchar) BoardId;
    BoardIdAndSig[RIO_BOARD_ID_SIG2_RPOS] = RIO_BOARD_ID_SIG2;

    if (RioLlEepromWrite(DevExt,
                         EEPROM_ADDR_BOARD_ID_AND_SIG,
                         BoardIdAndSig,
                         RIO_BOARD_ID_AND_SIG_SIZE) != RIO_OK)
    {
        return (RIO_BOARD_ID_WRITE_ERROR);
    }

    return (RIO_OK);
}


int32 GetBoardId(PDEVICE_EXTENSION DevExt, int32 *BoardId)
{
    uchar BoardIdAndSig[RIO_BOARD_ID_AND_SIG_SIZE];

    // default no id
    *BoardId = NO_BOARD_ID;

    if (RioLlEepromRead(DevExt,
                        EEPROM_ADDR_BOARD_ID_AND_SIG,
                        BoardIdAndSig,
                        RIO_BOARD_ID_AND_SIG_SIZE) != RIO_OK)

    {
        return (RIO_BOARD_ID_READ_ERROR);
    }

    if ((BoardIdAndSig[RIO_BOARD_ID_SIG1_RPOS] == RIO_BOARD_ID_SIG1) &&
        (BoardIdAndSig[RIO_BOARD_ID_SIG2_RPOS] == RIO_BOARD_ID_SIG2))
    {
        *BoardId = BoardIdAndSig[RIO_BOARD_ID_RPOS];
    }

    return (RIO_OK);
}


int32 GetGainDiff(PDEVICE_EXTENSION DevExt, char *GainDiff)
{
    uchar GainDiffAndSig[RIO_GAIN_DIFF_AND_SIG_SIZE];
    int32 r;

    // default zero GainDiff
    *GainDiff = 0;

    r = RioLlEepromRead(DevExt,

         RIO_EEPROM_ADDR_GAINDIFF_AND_SIG,
                                               GainDiffAndSig,
                                               RIO_GAIN_DIFF_AND_SIG_SIZE);
    if (r != RIO_OK)
                 return(r);

    if ((GainDiffAndSig[RIO_GAIN_DIFF_SIG1_RPOS] == RIO_GAIN_DIFF_SIG1) &&
        (GainDiffAndSig[RIO_GAIN_DIFF_SIG2_RPOS] == RIO_GAIN_DIFF_SIG2))
    {
        *GainDiff = GainDiffAndSig[RIO_GAIN_DIFF_RPOS];
    }

    return(RIO_OK);
}


int32 RioBoardIndexToDevExt(int32 BoardIndex, PDEVICE_EXTENSION *DevExt)
{
    if (BoardIndex < 0 || BoardIndex >= NrBoards)
        return (RIO_INVALID_BOARD_ID);

    *DevExt = BoardConf[BoardIndex].DevExt;

    return (RIO_OK);
}


int32 RioBoardIdToDevExt(int32 BoardId, PDEVICE_EXTENSION *DevExt)
{
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        if (BoardConf[i].BoardId == BoardId)
            break;
    }
    if (i == NrBoards)
    {
        return (RIO_INVALID_BOARD_ID);
    }

    *DevExt = BoardConf[i].DevExt;

    return (RIO_OK);
}


int32 RioGetCaptureTimeOut(int32 BoardId, RIO_INPUT_MODULE InputModule, long*
TimeOut)
{
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        if (BoardConf[i].BoardId == BoardId)
            break;
    }
    if (i == NrBoards)
    {
        return (RIO_INVALID_BOARD_ID);
    }

    *TimeOut = BoardConf[i].CaptureTimeOut[InputModule];

    return (RIO_OK);
}


int32 RioSetCaptureTimeOut(int32 BoardId, RIO_INPUT_MODULE InputModule, long
TimeOut)
{
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        if (BoardConf[i].BoardId == BoardId)
```

```
            break;
    }
    if (i == NrBoards)
    {
        return (RIO_INVALID_BOARD_ID);
    }

    BoardConf[i].CaptureTimeOut[InputModule] = TimeOut;

    return (RIO_OK);
}


int32 RioGetExtIntTimeOut(int32 BoardId, RIO_GPIO_PIN GpioPin, long* TimeOut)
{
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        if (BoardConf[i].BoardId == BoardId)
            break;
    }
    if (i == NrBoards)
    {
        return (RIO_INVALID_BOARD_ID);
    }

    *TimeOut = BoardConf[i].ExtIntTimeOut[GpioPin];

    return (RIO_OK);
}


int32 RioSetExtIntTimeOut(int32 BoardId, RIO_GPIO_PIN GpioPin, long TimeOut)
{
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        if (BoardConf[i].BoardId == BoardId)
            break;
    }
    if (i == NrBoards)
    {
        return (RIO_INVALID_BOARD_ID);
    }

    BoardConf[i].ExtIntTimeOut[GpioPin] = TimeOut;

    return (RIO_OK);
}


int32 RioGetOverlappedResultWithTimeOut(OVERLAPPED* Overlapped, long
MilliSecTimeOut)
{
    uint32 i;
    uint32 TimeOut = (uint32) 1000*MilliSecTimeOut + 1;
#if defined(DOS4GW)
    PBOARD_CONF pBoardConf;

    SearchBoardsForOverlapped(Overlapped, &pBoardConf);
#endif

    for (i = 0; i < TimeOut; i++)
    {
#if defined(DOS4GW)
```

```
        RioIrqHandler(pBoardConf);
#endif
        if (Overlapped->EventDone == TRUE)
        {
            break;
        }
        DelayOneUsec();
    }
    if (i == TimeOut)
    {
        return (RIO_TIMEOUT);
    }
    else
    {
        return (Overlapped->ReturnCode);
    }
}


int32 SizeOfRiodl(void)
{
    return (sizeof(int) + NrBoards * sizeof(int));
}


void CreateRiodl(PRIODL pRiodl)
{
    PDEVICE_EXTENSION DevExt;
    int i;

    for (i = 0; i < NrBoards; i++)
    {
        DevExt = BoardConf[i].DevExt;
         GetBoardId(DevExt, &(pRiodl->BoardId[i]));
    }

    pRiodl->NrBoards = NrBoards;
}
```

18. Program `RioCheck` yang dilambangkan dengan `RioCheck.c`

```
#include <stdio.h>
#include <stdlib.h>

#if !(defined(__WATCOMC__) && defined(__DOS__))
#include <windows.h>
#endif

#include <etype.h>
#include <edef.h>

#if defined(__WATCOMC__) && defined(__DOS__)
#include <riotype.h>
#if defined(PHARLAP)
#include <pharlap.h>
#endif
#endif

#include <riodef.h>
#include <rioerror.h>

#include <riolldef.h>

#include <riocheck.h>
```

```
#define RIO_PRIVATE
#include <rio.h>


/* check functions return TRUE if an erroneous parameter is detected */


BOOL RioCheckGpioPin(RIO_GPIO_PIN GpioPin)
{
    switch (GpioPin)
    {
        case RIO_GPIO_0:
        case RIO_GPIO_1:
        case RIO_GPIO_2:
        case RIO_GPIO_3:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckInputModule(int BoardId, RIO_INPUT_MODULE InputModule)
{
    BOOL IsBasicVersion;

    if (RioGetBoardType(BoardId, &IsBasicVersion) != RIO_OK)
    {
        return (TRUE);
    }

    switch (InputModule)
    {
        case RIO_IM_0:
            break;
        case RIO_IM_1:
            if (IsBasicVersion != FALSE)
                return (TRUE);
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckModuleMode(int BoardId, PRIO_MODULE Module)
{
    BOOL IsBasicVersion;

    if (RioGetBoardType(BoardId, &IsBasicVersion) != RIO_OK)
    {
        return (TRUE);
    }

    switch (Module->Mode)
    {
        case RIO_COLOR:
        case RIO_BW:
        case RIO_STEREO_LOCKED:
        case RIO_S_FULL_FRAME:
            break;
        case RIO_HQ:
        case RIO_FULL_FRAME:
        case RIO_RGBA:
            if (IsBasicVersion != FALSE)
                return (TRUE);
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckModuleInputModule(int BoardId, PRIO_MODULE Module)
{
    BOOL IsBasicVersion;

    if (RioGetBoardType(BoardId, &IsBasicVersion) != RIO_OK)
    {
        return (TRUE);
    }

    switch (Module->Mode)
    {
        case RIO_COLOR:
            if (Module->Def.Color.InputModule == RIO_IM_1 && IsBasicVersion !=
FALSE)
                return (TRUE);
            break;

        case RIO_BW:
            if (Module->Def.Bw.InputModule == RIO_IM_1 && IsBasicVersion !=
FALSE)
                return (TRUE);
            break;

        case RIO_STEREO_LOCKED:
            if (Module->Def.Sl.InputModule == RIO_IM_1 && IsBasicVersion !=
FALSE)
                return (TRUE);
            break;

        case RIO_S_FULL_FRAME:
            if (Module->Def.Sff.InputModule == RIO_IM_1 && IsBasicVersion !=
FALSE)
                return (TRUE);
            break;

        case RIO_HQ:
        case RIO_FULL_FRAME:
        case RIO_RGBA:
            if (IsBasicVersion != FALSE)
                return (TRUE);
            break;
    }
    return (FALSE);
}


BOOL RioCheckModuleScaler(int BoardId, PRIO_MODULE Module)
{
    RIO_SCALER Scaler;

    switch (Module->Mode)
    {
        case RIO_COLOR:
            Scaler = Module->Def.Color.Scaler;
            break;

        case RIO_BW:
            Scaler = Module->Def.Bw.Scaler;
            break;

        case RIO_HQ:
            Scaler = Module->Def.Hq.Scaler;
            break;

        case RIO_STEREO_LOCKED:
            Scaler = Module->Def.Sl.Scaler;
            break;

        case RIO_S_FULL_FRAME:
            Scaler = Module->Def.Sff.Scaler;
            break;

        case RIO_FULL_FRAME:
        case RIO_RGBA:
            Scaler = RIO_HPS; /* dummy value */
            break;
    }
    switch (Scaler)
    {
        case RIO_HPS:
        case RIO_BRS:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckModuleVideoStandard(int BoardId, PRIO_MODULE Module)
{
    BOOL IsBasicVersion;
    RIO_COLOR_VIDEO_STANDARD ColorVideoStandard;
    RIO_BW_VIDEO_STANDARD BwVideoStandard;

    if (RioGetBoardType(BoardId, &IsBasicVersion) != RIO_OK)
    {
        return (TRUE);
    }

    switch (Module->Mode)
    {
        case RIO_COLOR:
            ColorVideoStandard = Module->Def.Color.VideoStandard;
            break;

        case RIO_BW:
            BwVideoStandard = Module->Def.Bw.VideoStandard;
            break;

        case RIO_HQ:
            BwVideoStandard = Module->Def.Hq.VideoStandard;
            break;

        case RIO_STEREO_LOCKED:
            BwVideoStandard = Module->Def.Sl.VideoStandard;
            break;

        case RIO_S_FULL_FRAME:
            BwVideoStandard = Module->Def.Sff.VideoStandard;
            break;

        case RIO_FULL_FRAME:
            BwVideoStandard = Module->Def.Ff.VideoStandard;
            break;

        case RIO_RGBA:
            BwVideoStandard = Module->Def.Rgba.VideoStandard;
            break;
    }
    if (Module->Mode == RIO_COLOR)
    {
        switch (ColorVideoStandard)
        {
            case RIO_PAL:
            case RIO_SECAM:
            case RIO_NTSC:
                break;
            default:
                return (TRUE);
        }
    } else
    {
        switch (BwVideoStandard)
        {
            case RIO_CCIR:
            case RIO_EIA:
                break;
            default:
                return (TRUE);
        }
    }
    return (FALSE);
}


BOOL RioCheckModuleTvOrVtr(int BoardId, PRIO_MODULE Module)
{
    RIO_TV_OR_VTR TvOrVtr;

    switch (Module->Mode)
    {
        case RIO_COLOR:
            TvOrVtr = Module->Def.Color.TvOrVtr;
            break;

        case RIO_BW:
            TvOrVtr = Module->Def.Bw.TvOrVtr;
            break;

        case RIO_HQ:
            TvOrVtr = Module->Def.Hq.TvOrVtr;
            break;

        case RIO_STEREO_LOCKED:
            TvOrVtr = Module->Def.Sl.TvOrVtr;
            break;

        case RIO_S_FULL_FRAME:
            TvOrVtr = Module->Def.Sff.TvOrVtr;
            break;

        case RIO_FULL_FRAME:
            TvOrVtr = Module->Def.Ff.TvOrVtr;
            break;

        case RIO_RGBA:
            TvOrVtr = Module->Def.Rgba.TvOrVtr;
            break;
    }
    switch (TvOrVtr)
    {
        case RIO_TV:
        case RIO_VTR:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckModuleOutputFormat(int BoardId, PRIO_MODULE Module)
{
    RIO_COLOR_OUTPUT_FORMAT ColorOutputFormat;
    RIO_BW_OUTPUT_FORMAT BwOutputFormat;
    RIO_RGBA_OUTPUT_FORMAT RgbaOutputFormat;

    switch (Module->Mode)
    {
        case RIO_COLOR:
            ColorOutputFormat = Module->Def.Color.OutputFormat;
            break;

        case RIO_BW:
            BwOutputFormat = Module->Def.Bw.OutputFormat;
            break;

        case RIO_HQ:
            BwOutputFormat = Module->Def.Hq.OutputFormat;
            break;

        case RIO_STEREO_LOCKED:
        case RIO_S_FULL_FRAME:
            BwOutputFormat = RIO_Y8;
            break;

        case RIO_FULL_FRAME:
            BwOutputFormat = Module->Def.Ff.OutputFormat;
            break;

        case RIO_RGBA:
            RgbaOutputFormat = Module->Def.Rgba.OutputFormat;
            break;
    }
    if (Module->Mode == RIO_COLOR)
    {
        switch (ColorOutputFormat)
        {
            case RIO_YUV16:
            case RIO_RGB8:
            case RIO_ARGB15:
            case RIO_RGAB15:
            case RIO_RGB16:
            case RIO_RGB24:
            case RIO_ARGB32:
            case RIO_RGB8_GC:
            case RIO_ARGB15_GC:
            case RIO_RGAB15_GC:
            case RIO_RGB16_GC:
            case RIO_RGB24_GC:
            case RIO_ARGB32_GC:
                break;
            default:
                return (TRUE);
        }
    } else if (Module->Mode == RIO_RGBA)
    {
        switch (RgbaOutputFormat)
        {
            case RIO_RGBA_RGB24:
            case RIO_RGBA_ARGB32:
            case RIO_RGBA_RGB24P:
            case RIO_RGBA_ARGB32P:
                break;
            default:
                return (TRUE);
        }
    } else
    {
        switch (BwOutputFormat)
        {
            case RIO_Y1:
            case RIO_Y2:
            case RIO_Y8:
                break;
            default:
                return (TRUE);
        }
    }
    return (FALSE);
}


BOOL RioCheckModuleFieldOrFrame(int BoardId, PRIO_MODULE Module)
{
    RIO_FIELD_OR_FRAME FieldOrFrame;

    switch (Module->Mode)
    {
        case RIO_COLOR:
            FieldOrFrame = Module->Def.Color.FieldOrFrame;
            break;

        case RIO_BW:
            FieldOrFrame = Module->Def.Bw.FieldOrFrame;
            break;

        case RIO_HQ:
            FieldOrFrame = Module->Def.Hq.FieldOrFrame;
            break;

        case RIO_STEREO_LOCKED:
            FieldOrFrame = Module->Def.Sl.FieldOrFrame;
            break;

        case RIO_S_FULL_FRAME:
        case RIO_FULL_FRAME:
            FieldOrFrame = RIO_FRAME;
            break;

        case RIO_RGBA:
            FieldOrFrame = Module->Def.Rgba.FieldOrFrame;
            break;
    }
    switch (FieldOrFrame)
    {
        case RIO_FIELD:
        case RIO_FRAME:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}
```

```
BOOL RioCheckModuleCvbsOrYc(int BoardId, PRIO_MODULE Module)
{
    if (Module->Mode == RIO_COLOR)
    {
        switch (Module->Def.Color.CvbsOrYc)
        {
            case RIO_CVBS:
            case RIO_YC:
                break;
            default:
                return (TRUE);
        }
    }
    return (FALSE);
}


BOOL RioCheckModulePostProcess(int BoardId, PRIO_MODULE Module)
{
    RIO_ON_OFF_MODE PostProcess;

    switch (Module->Mode)
    {
        case RIO_COLOR:
        case RIO_BW:
        case RIO_HQ:
        case RIO_FULL_FRAME:
            PostProcess = RIO_OFF;
            break;

        case RIO_STEREO_LOCKED:
            PostProcess = Module->Def.Sl.PostProcess;
            break;

        case RIO_S_FULL_FRAME:
            PostProcess = Module->Def.Sff.PostProcess;
            break;

        case RIO_RGBA:
            PostProcess = Module->Def.Rgba.PostProcess;
            break;
    }
    return (RioCheckOnOffMode(PostProcess));
}


BOOL RioCheckOnOffMode(RIO_ON_OFF_MODE OnOffMode)
{
    switch (OnOffMode)
    {
        case RIO_ON:
        case RIO_OFF:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckGpioMode(RIO_GPIO_MODE GpioMode)
{
    switch (GpioMode)
    {
        case RIO_GPIO_IGNORE:
```

```
        case RIO_GPIO_INPUT:
        case RIO_GPIO_OUTPUT:
        case RIO_GPIO_IRQ_RISE:
        case RIO_GPIO_IRQ_FALL:
        case RIO_GPIO_IRQ_BOTH:
            break;
        default:
            return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckEepromAddress(int Address)
{
    if (Address >= RIO_EEPROM_SIZE || Address < 0)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckEepromAddressAndDataSize(int Address, int Size)
{
    if ((Address + Size) > RIO_EEPROM_SIZE || Size < 0)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckCameraForModule(int BoardId, RIO_INPUT_MODULE InputModule, int
Camera)
{
    uchar NrModules;
    RIO_MODULE Module[RIO_NR_IM];

    if (RioGetInputModule(BoardId, &NrModules, Module) != RIO_OK)
    {
        return (TRUE);
    }

    if ((Module[InputModule].Mode == RIO_BW) ||
        ((Module[InputModule].Mode == RIO_COLOR) &&
         (Module[InputModule].Def.Color.CvbsOrYc == RIO_CVBS)))
    {
        if ((Camera < 0) || (Camera >= RIO_CVBS_INPUTS))
        {
            return (TRUE);
        }
    } else if (Module[InputModule].Mode == RIO_RGBA)
    {
        if (Camera != 0)
        {
            return (TRUE);
        }
    } else
    {
        if ((Camera < 0) || (Camera >= RIO_YC_INPUTS))
        {
            return (TRUE);
        }
    }
    return (FALSE);
}
```

```
BOOL RioCheckInputGain(uchar Gain)
{
    if (Gain > SAA_7110_MAX_GAIN)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckHqGain(float Gain)
{
    if (Gain < RIO_HQ_GAIN_MIN || Gain > RIO_HQ_GAIN_MAX)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckHqOffset(float Offset)
{
    if (Offset < RIO_HQ_OFFSET_MIN || Offset > RIO_HQ_OFFSET_MAX)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckI2cReg(uchar Reg)
{
    if (Reg >= RIO_NR_7110_REGS)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckRect(RECT *Rect)
{
    if (Rect == NULL || Rect->left >= Rect->right || Rect->bottom <= Rect->top)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckPtr(void *Ptr)
{
    if (Ptr == NULL)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckFlashLine(uint16 StartLine, uint16 EndLine, uint16 Length)
{
    if (EndLine < StartLine || Length > (EndLine - StartLine + 1))
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckNotZero(uint32 Val)
{
    if (Val == 0)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckTriggerPos(uint32 TriggerPosition, uint32 NrBuffers)
{
    if (TriggerPosition >= NrBuffers)
    {
        return (TRUE);
    }
    return (FALSE);
}


BOOL RioCheckNotNegative(int Val)
{
    if (Val < 0)
    {
        return (TRUE);
    }
    return (FALSE);
}


/*-------------------------------------------------------------------------*/


int RioCheckSetLed(RIO_ON_OFF_MODE LedState)
{
    if (RioCheckOnOffMode(LedState))
    {
        return(RIO_INVALID_LED_STATE);
    }
    return(RIO_OK);
}


int RioCheckGpioControl(RIO_GPIO_MODE Gpio0,
                        RIO_GPIO_MODE Gpio1,
                        RIO_GPIO_MODE Gpio2,
                        RIO_GPIO_MODE Gpio3)
{
    if (RioCheckGpioMode(Gpio0))
    {
        return(RIO_INVALID_GPIO_MODE);
    }
    if (RioCheckGpioMode(Gpio1))
    {
        return(RIO_INVALID_GPIO_MODE);
    }
    if (RioCheckGpioMode(Gpio2))
    {
        return(RIO_INVALID_GPIO_MODE);
    }
    if (RioCheckGpioMode(Gpio3))
    {
        return(RIO_INVALID_GPIO_MODE);
    }
    return(RIO_OK);
}


int RioCheckEeprom(int Address,
                   void *Data,
                   int Size)
{
    if (RioCheckEepromAddress(Address))
    {
        return(RIO_INVALID_EEPROM_ADDRESS);
    }
    if (RioCheckEepromAddressAndDataSize(Address, Size))
    {
        return(RIO_INVALID_EEPROM_DATASIZE_FOR_ADDRESS);
    }
    return(RIO_OK);
}


int RioCheckI2cRead(int BoardId,
                    RIO_INPUT_MODULE InputModule,
                    uchar Reg,
                    uchar *Value)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckI2cReg(Reg))
    {
        return(RIO_INVALID_I2C_REG);
    }
    return(RIO_OK);
}


int RioCheckI2cWrite(int BoardId,
                     RIO_INPUT_MODULE InputModule,
                     uchar Reg,
                     uchar Value)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckI2cReg(Reg))
    {
        return(RIO_INVALID_I2C_REG);
    }
    return(RIO_OK);
}


int RioCheckBCSH(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckSetInputGain(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Input,
                         uchar Value,
                         RIO_ON_OFF_MODE AutoGainMode)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckCameraForModule(BoardId, InputModule, Input))
    {
        return(RIO_INVALID_INPUT_FOR_INPUT_MODULE);
    }
    if (RioCheckInputGain(Value))
    {
        return(RIO_INVALID_INPUT_GAIN);
    }
    if (RioCheckOnOffMode(AutoGainMode))
    {
        return(RIO_INVALID_AUTO_GAIN_MODE);
    }
    return(RIO_OK);
}


int RioCheckGetInputGain(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Input,
                         uchar *Value,
                         RIO_ON_OFF_MODE *AutoGainMode)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckCameraForModule(BoardId, InputModule, Input))
    {
        return(RIO_INVALID_INPUT_FOR_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckSetInputModule(int BoardId, PRIO_MODULE Module)
{
    if (RioCheckPtr(Module))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckModuleMode(BoardId, Module))
    {
                 return(RIO_INVALID_MODULE_MODE);
    }
    if (RioCheckModuleInputModule(BoardId, Module))
    {
                 return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckModuleScaler(BoardId, Module))
    {
                 return(RIO_INVALID_SCALER);
    }
    if (RioCheckModuleVideoStandard(BoardId, Module))
    {
                 return(RIO_INVALID_VIDEO_STANDARD);
    }
    if (RioCheckModuleTvOrVtr(BoardId, Module))
    {
```

```
                 return(RIO_INVALID_TV_OR_VTR);
    }
    if (RioCheckModuleOutputFormat(BoardId, Module))
    {
                 return(RIO_INVALID_OUTPUT_FORMAT);
    }
    if (RioCheckModuleFieldOrFrame(BoardId, Module))
    {
                 return(RIO_INVALID_FIELD_OR_FRAME);
    }
    if (RioCheckModuleCvbsOrYc(BoardId, Module))
    {
                 return(RIO_INVALID_CVBS_OR_YC);
    }
    if (RioCheckModulePostProcess(BoardId, Module))
    {
                 return(RIO_INVALID_POST_PROCESS);
    }
    return(RIO_OK);
}


int RioCheckSelectCamera(int BoardId,
                         RIO_INPUT_MODULE InputModule,
                         int Camera)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckCameraForModule(BoardId, InputModule, Camera))
    {
        return(RIO_INVALID_CAMERA_FOR_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckGetCamera(int BoardId,
                      RIO_INPUT_MODULE InputModule,
                      int *Camera)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckGetChromaKey(int BoardId,
                         char *VLowerLimit,
                         char *VUpperLimit,
                         char *ULowerLimit,
                         char *UUpperLimit,
                         BOOL *Enabled)
{
    if (RioCheckPtr((void *) VLowerLimit))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckPtr((void *) VUpperLimit))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckPtr((void *) ULowerLimit))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckPtr((void *) UUpperLimit))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckPtr((void *) Enabled))
    {
        return(RIO_INVALID_POINTER);
    }
    return(RIO_OK);
}


int RioCheckSetHqBw(int BoardId, float Gain, float Offset)
{
    if (RioCheckHqGain(Gain))
    {
        return(RIO_INVALID_HQ_GAIN);
    }
    if (RioCheckHqOffset(Offset))
    {
        return(RIO_INVALID_HQ_OFFSET);
    }
    return(RIO_OK);
}


int RioCheckExternalIntTimeOut(int BoardId, RIO_GPIO_PIN GpioPin, long TimeOut)
{
    if (RioCheckGpioPin(GpioPin))
    {
        return(RIO_INVALID_GPIO_PIN);
    }
    return(RIO_OK);
}


int RioCheckExternalInt(int BoardId, RIO_GPIO_PIN GpioPin)
{
    if (RioCheckGpioPin(GpioPin))
    {
        return(RIO_INVALID_GPIO_PIN);
    }
    return(RIO_OK);
}


int RioCheckExternalIntCancel(int BoardId, RIO_GPIO_PIN GpioPin)
{
    if (RioCheckGpioPin(GpioPin))
    {
        return(RIO_INVALID_GPIO_PIN);
    }
    return(RIO_OK);
}


int RioCheckCaptureTimeOut(int BoardId, RIO_INPUT_MODULE InputModule, long
TimeOut)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}
```

```
int RioCheckCapture(int BoardId,
                    RIO_INPUT_MODULE InputModule,
                    BOOL Continuous,
                    BOOL SquarePixels,
                    BOOL TopDown,
                    RECT *SrcRect,
                    RECT *DestRect,
                    int Pitch,
                    HANDLE ImageBufferHandle,
                    OVERLAPPED *pOverlapped)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckRect(SrcRect))
    {
        return(RIO_INVALID_SRC_RECT);
    }
    if (RioCheckRect(DestRect))
    {
        return(RIO_INVALID_DEST_RECT);
    }
    if (RioCheckPtr((void *) ImageBufferHandle))
    {
        return(RIO_INVALID_HANDLE);
    }
    if (RioCheckPtr((void *) pOverlapped) && (Continuous != FALSE))
    {
        return(RIO_INVALID_OVERLAPPED);
    }
    return(RIO_OK);
}


int RioCheckCaptureCancel(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckCaptureStop(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckPostProcess(int BoardId,
                                           PRIO_MODULE Module,
                                           BOOL TopDown,
                                           RECT *DestRect,
                                           int Pitch,
                                           HANDLE ImageBufferHandle
                                          )
{
    if (RioCheckPtr(Module))
    {
        return(RIO_INVALID_POINTER);
    }
    if (RioCheckModuleMode(BoardId, Module))
    {
                 return(RIO_INVALID_MODULE_MODE);
    }
    if (RioCheckModuleOutputFormat(BoardId, Module))
    {
                 return(RIO_INVALID_OUTPUT_FORMAT);
    }
    if (RioCheckRect(DestRect))
    {
        return(RIO_INVALID_DEST_RECT);
    }
    if (RioCheckPtr((void *) ImageBufferHandle))
    {
        return(RIO_INVALID_HANDLE);
    }
    return(RIO_OK);
}


int RioCheckStreamInit(int BoardId,
                       RIO_INPUT_MODULE InputModule,
                       int MicroSecPerCapture,
                       BOOL SquarePixels,
                       BOOL TopDown,
                       RECT *SrcRect,
                       RECT *DestRect,
                       int Pitch)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckNotNegative(MicroSecPerCapture))
    {
        return(RIO_INVALID_MICROSEC_PER_CAPTURE);
    }
    if (RioCheckRect(SrcRect))
    {
        return(RIO_INVALID_SRC_RECT);
    }
    if (RioCheckRect(DestRect))
    {
        return(RIO_INVALID_DEST_RECT);
    }
    return(RIO_OK);
}


int RioCheckStreamClose(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckStreamStart(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckStreamStop(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckStreamAddEmptyBuffer(int BoardId,
                                 RIO_INPUT_MODULE InputModule,
                                 PRIO_VIDEO_HDR VideoHdr)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckPtr(VideoHdr))
    {
        return(RIO_INVALID_POINTER);
    }
    return(RIO_OK);
}


int RioCheckStreamGetFilledBuffer(int BoardId,
                                  RIO_INPUT_MODULE InputModule,
                                  PRIO_VIDEO_HDR *VideoHdr,
                                  OVERLAPPED *pOverlapped)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckPtr((void *) pOverlapped))
    {
        return(RIO_INVALID_OVERLAPPED);
    }
    return(RIO_OK);
}


int RioCheckStreamGetStartTime(int BoardId, RIO_INPUT_MODULE InputModule, DWORD
*StartTime)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (RioCheckPtr((void *) StartTime))
    {
        return(RIO_INVALID_POINTER);
    }
    return(RIO_OK);
}


int RioCheckTriggerCaptureTimeOut(int BoardId, RIO_INPUT_MODULE InputModule,
long TimeOut)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckTriggerCapture(int BoardId,
                           RIO_INPUT_MODULE InputModule,
                           BOOL SquarePixels,
                                                       BOOL TopDown,
                           PRIO_FLASH_PARAMS FlashParams,
                           PRIO_TBUFFER_PARAMS TBufferParams,
                                                  RECT *SrcRect,
                                      RECT *DestRect,
                                                       int Pitch,
                                                       OVERLAPPED *pOverlapped)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    if (FlashParams != NULL)
    {
        if (RioCheckGpioPin(FlashParams->FlashPin))
        {
            return(RIO_INVALID_FLASH_PIN);
        }
        if (RioCheckFlashLine(FlashParams->StartLine, FlashParams->EndLine,
FlashParams->Length))
        {
            return(RIO_INVALID_FLASH_LINE);
        }
        if (RioCheckNotZero((uint32) FlashParams->VideoOutDelay))
        {
            return(RIO_INVALID_VIDEO_OUT_DELAY);
        }
    }
    if (RioCheckPtr((void *) TBufferParams))
    {
        return(RIO_INVALID_TBUFFER_PARAMS);
    }
    if (RioCheckGpioPin(TBufferParams->TriggerPin))
    {
        return(RIO_INVALID_TRIGGER_PIN);
    }
    if (RioCheckNotZero(TBufferParams->NrBuffers))
    {
        return(RIO_INVALID_NR_BUFFERS);
    }
    if (RioCheckPtr((void *) TBufferParams->ImageBuffer))
    {
        return(RIO_INVALID_IMAGE_BUFFER);
    }
    if (RioCheckTriggerPos(TBufferParams->TriggerPosition, TBufferParams-
>NrBuffers))
    {
        return(RIO_INVALID_TRIGGER_POSITION);
    }
    if (RioCheckPtr((void *) TBufferParams->TriggerBufferNr))
    {
        return(RIO_INVALID_TRIGGER_BUFFER_NR);
    }
    if (RioCheckNotZero((uint32) TBufferParams->CapturePeriod))
    {
        return(RIO_INVALID_CAPTURE_PERIOD);
    }
    if (RioCheckRect(SrcRect))
    {
        return(RIO_INVALID_SRC_RECT);
    }
    if (RioCheckRect(DestRect))
    {
        return(RIO_INVALID_DEST_RECT);
    }
    return(RIO_OK);
}


int RioCheckTriggerCaptureCancel(int BoardId, RIO_INPUT_MODULE InputModule)
{
    if (RioCheckInputModule(BoardId, InputModule))
    {
        return(RIO_INVALID_INPUT_MODULE);
    }
    return(RIO_OK);
}


int RioCheckCreateRiodl(PRIODL pRiodl)
{
    if (RioCheckPtr((void *) pRiodl))
    {
        return(RIO_INVALID_POINTER);
    }
    return(RIO_OK);
}


int RioCheckGetOverlappedResult(OVERLAPPED *pOverlapped,
                                int *ReturnCode,
                                BOOL bWait)
{
    if (RioCheckPtr((void *) pOverlapped))
    {
        return(RIO_INVALID_OVERLAPPED);
    }
    if (RioCheckPtr((void *) ReturnCode))
    {
        return(RIO_INVALID_POINTER);
    }
    return(RIO_OK);
}
```

19. Contoh program untuk melakukan penangkapan satu citra

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#if defined(__WATCOMC__)
#include <time.h>
#include <dos.h>
#include <conio.h>
#include <graph.h>
#include “i860.asm”
#endif
#include <etype.h>
#include <edef.h>
```

```
#include <riotype.h>
#include <riodef.h>
#include <rioerror.h>
#include <rio.h>

#define SCREEN_AREA 0xa0000
#define SCREEN_WIDTH    320
#define SCREEN_HEIGHT   200
#define FIELD_XOFF        0
#define FIELD_YOFF       16
#define FIELD_WIDTH     384
#define FIELD_HEIGHT    288
#define FIELD_WIDTH       SCREEN_WIDTH
#define FIELD_HEIGHT      SCREEN_HEIGHT
#define FIELD_DEPTH       1

#define RGB(r,g,b) ((long) ((b) << 16) | ((g) << 8) | (r))
#define delay_one_usec() inp(0x80)

/* utility functions */
long _myremappalette(short pixval, long color)
{
 uchar ok, c[3], oldc[3];
 int i, j;
 long oldcolor;
 for (i = 0; i < 3; i++)
 {
  c[i] = color & 0xFF;
  color >>= 8;
 }
 _disable();
 outp(0x3C7, pixval);
 delay_one_usec();
 for (i = 0; i < 3; i++)
  oldc[i] = inp(0x3C9);

 for (i = 0, ok = 0; (i < 10) && !ok; i++) /* 10 retries max */
 {
  outp(0x3C8, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   outp(0x3C9, c[j]);
   delay_one_usec();
  }
  ok = 1;
  outp(0x3C7, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   if (inp(0x3C9) != c[j])
    ok = 0;
   delay_one_usec();
  }
 }
 _enable();
 for (i = 2, oldcolor = 0; i >= 0; i--)
 {
  oldcolor |= oldc[i];
  oldcolor <<= 8;
 }
 if (!ok)
  return(-1);
 else
  return(oldcolor);
}
short _myremapallpalette(long *colors)
{
 int i;
 for (i = 0; i < 256; i++)
  if (_myremappalette(i, colors[i]) == -1)
   return(-1);
 return(0);
}
void usage()
{
   exit(1);
}
void errorval()
{
   printf("\n");
   exit(9);
}
int32 DoRioSetup(int32 boardId)
{
    int32 retVal;
    RIO_MODULE Module;

    if (FIELD_DEPTH == 1)
    {
#if defined(TEST_HQ)
        Module.Mode = RIO_HQ;
#if defined(TEST_BRS)
        Module.Def.Hq.Scaler = RIO_BRS;
#else
        Module.Def.Hq.Scaler = RIO_HPS;
#endif
        Module.Def.Hq.VideoStandard = RIO_CCIR;
        Module.Def.Hq.TvOrVtr = RIO_VTR;
        Module.Def.Hq.FieldOrFrame = RIO_FIELD;
        Module.Def.Hq.OutputFormat = RIO_Y8;
#else
        Module.Mode = RIO_BW;
        Module.Def.Bw.InputModule = RIO_IM_0;
#if defined(TEST_BRS)
        Module.Def.Bw.Scaler = RIO_BRS;
#else
        Module.Def.Bw.Scaler = RIO_HPS;
#endif
        Module.Def.Bw.VideoStandard = RIO_CCIR;
        Module.Def.Bw.TvOrVtr = RIO_VTR;
        Module.Def.Bw.FieldOrFrame = RIO_FIELD;
        Module.Def.Bw.OutputFormat = RIO_Y8;
#endif
    } else
    {
        Module.Mode = RIO_COLOR;
        Module.Def.Color.InputModule = RIO_IM_0;
        Module.Def.Color.Scaler = RIO_HPS;
        Module.Def.Color.VideoStandard = RIO_PAL;
        Module.Def.Color.CvbsOrYc = RIO_CVBS;
        Module.Def.Color.CvbsOrYc = RIO_YC;
        Module.Def.Color.TvOrVtr = RIO_VTR;
        Module.Def.Color.FieldOrFrame = RIO_FIELD;
        Module.Def.Color.OutputFormat = RIO_RGB24;
    }

    if ((retVal = RioSetInputModule(boardId, &Module)) != RIO_OK)
        return (retVal);

    if((retVal = RioSelectCamera(boardId, RIO_IM_0, 0)) != RIO_OK)
      return (retVal);
}
```

```
void main(int argc, char *argv[])
{
    PRIODL pRiodl;
    int32 boardId;
    int32 retVal;
    int32 h, i, j;
    long colors[256];
    RECT SrcRect = {FIELD_XOFF, FIELD_YOFF, FIELD_XOFF + FIELD_WIDTH, FIELD_YOFF
+ FIELD_HEIGHT};
    RECT DestRect = {0, 0, FIELD_WIDTH, FIELD_HEIGHT};
    void *image;
    uchar *screenPtr, *dataPtr;
    char ch;
    HANDLE imagebh;
    setvbuf( stdout, NULL, _IONBF, 0 );
    if (RioOpen() != RIO_OK)
    {
        printf("Could not find Rio");
        exit(1);
    }
    printf("found Rio\n");
    pRiodl = (PRIODL) calloc(RioSizeOfRiodl(), 1);
    retVal = RioCreateRiodl(pRiodl);
    if (retVal != RIO_OK)
    {
        printf("RioCreateRiodl returned error 0x%x", retVal);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    boardId = pRiodl->BoardId[0];

    if (DoRioSetup(boardId) != RIO_OK)
    {
        free(pRiodl);
        RioClose();
        exit(2);
    }
    image = calloc(FIELD_HEIGHT * FIELD_WIDTH * FIELD_DEPTH, 1);
    if ((retVal = RioScatterLock(image, FIELD_HEIGHT * FIELD_WIDTH * FIELD_DEPTH,
&imagebh)) != RIO_OK)
    {
        free(image);
        free(pRiodl);
        RioClose();
        printf("RioScatterLock returned error 0x%x", retVal);
        exit(2);
    }
    if ((retVal = RioCapture(boardId, RIO_IM_0, FALSE, FALSE, TRUE, &SrcRect,
&DestRect,
        FIELD_WIDTH * FIELD_DEPTH, imagebh, NULL)) != RIO_OK)
    {
        RioScatterUnlock(imagebh);
        free(image);
        free(pRiodl);
        RioClose();
        printf("RioCapture returned error 0x%x", retVal);
        exit(2);
    } else
    {
        printf("Capture done\n");
    }

    /* rest of program displays image on screen */

    _setvideomode(_MRES256COLOR);   /* switch to 320x200x8 */
    _clearscreen(_GCLEARSCREEN);

/* load a standardgrey palette */
    for (i = 0; i < 256; i++)
        colors[i] = RGB(i >> 2, i >> 2, i >> 2);
    _myremapallpalette(colors);

/* init vga */
#if defined(PHARLAP)
    /* map the VGA in the data segment at offset 64K */
    screenPtr = (uchar *) 0x10000;
    if((retVal = _dx_map_pgsn(screenPtr, 0x10000L, 0xA0000L))!= 0)
    {
        RioScatterUnlock(imagebh);
        free(image);
        free(pRiodl);
        RioClose();
        printf("Can't map VGA physical memory, error: %d\n", retVal);
        exit(1);
    }
#elif defined(DOS4GW)
    screenPtr = (uchar *)SCREEN_AREA;
#endif
    dataPtr = (uchar *)image;
    if (FIELD_DEPTH == 1)
    {
        if ((FIELD_WIDTH == SCREEN_WIDTH) && (FIELD_HEIGHT == SCREEN_HEIGHT))
            memcpy(screenPtr, dataPtr, SCREEN_HEIGHT*SCREEN_WIDTH);
        else
        {
            for (h = 0; h < SCREEN_HEIGHT; h++)
                memcpy(screenPtr  +  h*SCREEN_WIDTH,  dataPtr  +  h*FIELD_WIDTH,
SCREEN_WIDTH);
        }

        while (!kbhit())
            ;
        ch = getch();
    } else
    {
        for (i = 0; i < 3; i++)
        {
            for (h = 0; h < SCREEN_HEIGHT; h++)
            {
                for (j = 0; j < SCREEN_WIDTH; j++)
                    screenPtr[j   +   h*SCREEN_WIDTH]   =   dataPtr[i   +   j*3   +
h*FIELD_WIDTH*FIELD_DEPTH];
            }

            while (!kbhit())
                ;
            ch = getch();
        }
    }
    _setvideomode(_TEXTC80);
    RioScatterUnlock(imagebh);
    free(image);
    free(pRiodl);
    RioClose();
}
```

20. Contoh program untuk melakukan penangkapan dua citra dari dua kamera (satu citra untuk satu kamera) yang di-*overlap*.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#if defined(__WATCOMC__)
#include <time.h>
#include <dos.h>
```

```
#include <conio.h>
#endif
#include <etype.h>
#include <edef.h>
#include <riotype.h>
#include <riodef.h>
#include <rioerror.h>
#include <rio.h>

#define SCREEN_AREA 0xa0000
#define SCREEN_WIDTH    320
#define SCREEN_HEIGHT   200
#define FIELD_XOFF        0
#define FIELD_YOFF       16
#define FIELD_WIDTH       SCREEN_WIDTH
#define FIELD_HEIGHT      SCREEN_HEIGHT
#define FIELD_DEPTH       1
#define RGB(r,g,b) ((long) ((b) << 16) | ((g) << 8) | (r))
#define delay_one_usec() inp(0x80)

/* utility functions */
long _myremappalette(short pixval, long color)
{
 uchar ok, c[3], oldc[3];
 int i, j;
 long oldcolor;
 for (i = 0; i < 3; i++)
 {
  c[i] = color & 0xFF;
  color >>= 8;
 }
 _disable();
 outp(0x3C7, pixval);
 delay_one_usec();
 for (i = 0; i < 3; i++)
  oldc[i] = inp(0x3C9);
 for (i = 0, ok = 0; (i < 10) && !ok; i++) /* 10 retries max */
 {
  outp(0x3C8, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   outp(0x3C9, c[j]);
   delay_one_usec();
  }
  ok = 1;
  outp(0x3C7, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   if (inp(0x3C9) != c[j])
    ok = 0;
   delay_one_usec();
  }
 }
 _enable();
 for (i = 2, oldcolor = 0; i >= 0; i--)
 {
  oldcolor |= oldc[i];
  oldcolor <<= 8;
 }
 if (!ok)
  return(-1);
 else
  return(oldcolor);
}
short _myremapallpalette(long *colors)
{
 int i;

 for (i = 0; i < 256; i++)
  if (_myremappalette(i, colors[i]) == -1)
   return(-1);
 return(0);
}
void usage()
{
   exit(1);
}
void errorval()
{
   printf("\n");
   exit(9);
}
int32 DoRioSetup(int32 boardId)
{
    int32 retVal;
    RIO_MODULE Module;
    Module.Mode = RIO_BW;
    Module.Def.Bw.InputModule = RIO_IM_0;
    Module.Def.Bw.Scaler = RIO_HPS;
    Module.Def.Bw.VideoStandard = RIO_CCIR;
    Module.Def.Bw.TvOrVtr = RIO_VTR;
    Module.Def.Bw.FieldOrFrame = RIO_FIELD;
    Module.Def.Bw.OutputFormat = RIO_Y8;
    if ((retVal = RioSetInputModule(boardId, &Module)) != RIO_OK)
        return (retVal);
    if ((retVal = RioSelectCamera(boardId, RIO_IM_0, 1)) != RIO_OK)
        return (retVal);

    Module.Mode = RIO_BW;
    Module.Def.Bw.InputModule = RIO_IM_1;
    Module.Def.Bw.Scaler = RIO_BRS;
    Module.Def.Bw.VideoStandard = RIO_CCIR;
    Module.Def.Bw.TvOrVtr = RIO_VTR;
    Module.Def.Bw.FieldOrFrame = RIO_FIELD;
    Module.Def.Bw.OutputFormat = RIO_Y8;
    if ((retVal = RioSetInputModule(boardId, &Module)) != RIO_OK)
        return (retVal);
//    if ((retVal = RioSelectCamera(boardId, RIO_IM_1, 4)) != RIO_OK)
    if ((retVal = RioSelectCamera(boardId, RIO_IM_1, 2)) != RIO_OK)
        return (retVal);
    return (retVal);
}
void main(int argc, char *argv[])
{
    PRIODL pRiodl;
    int32 boardId;
    int32 retVal;
    int32 h, i;
    long colors[256];
    RECT SrcRect = {FIELD_XOFF, FIELD_YOFF, FIELD_XOFF + FIELD_WIDTH, FIELD_YOFF
+ FIELD_HEIGHT};
    RECT DestRect = {0, 0, FIELD_WIDTH, FIELD_HEIGHT};
    void *image0, *image1;
    uchar *screenPtr, *dataPtr;
    char ch;
    HANDLE imagebh0, imagebh1;
    OVERLAPPED Overlapped0, Overlapped1;
    int32 CapResult0, CapResult1;
    BOOL CapDone0 = FALSE, CapDone1 = FALSE;
    setvbuf( stdout, NULL, _IONBF, 0 );
    if (RioOpen() != RIO_OK)
    {
        printf("Could not find Rio");
        exit(1);
    }
    printf("found Rio\n");
    pRiodl = (PRIODL) calloc(RioSizeOfRiodl(), 1);
    retVal = RioCreateRiodl(pRiodl);
    if (retVal != RIO_OK)
    {
        printf("RioCreateRiodl returned error 0x%x", retVal);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    boardId = pRiodl->BoardId[0];
    retVal = DoRioSetup(boardId);
    if (retVal != RIO_OK)
    {
        printf("DoRioSetup returned error 0x%x", retVal);
        free(pRiodl);
        RioClose();
        exit(2);
    }

    image0 = calloc(FIELD_HEIGHT * FIELD_WIDTH * FIELD_DEPTH, 1);
    if   ((retVal   =   RioScatterLock(image0,   FIELD_HEIGHT   *   FIELD_WIDTH   *
FIELD_DEPTH, &imagebh0)) != RIO_OK)
    {
        printf("RioScatterLock returned error 0x%x", retVal);
        free(image0);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    image1 = calloc(FIELD_HEIGHT * FIELD_WIDTH * FIELD_DEPTH, 1);
    if   ((retVal   =   RioScatterLock(image1,   FIELD_HEIGHT   *   FIELD_WIDTH   *
FIELD_DEPTH, &imagebh1)) != RIO_OK)
    {
        printf("RioScatterLock returned error 0x%x", retVal);
        RioScatterUnlock(imagebh0);
        free(image0);
        free(image1);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    if ((retVal = RioCapture(boardId, RIO_IM_0, FALSE, FALSE, TRUE, &SrcRect,
&DestRect,
        FIELD_WIDTH * FIELD_DEPTH, imagebh0, &Overlapped0)) != RIO_PENDING)
    {
        printf("RioCapture returned error 0x%x", retVal);
        RioScatterUnlock(imagebh1);
        RioScatterUnlock(imagebh0);
        free(image1);
        free(image0);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    if ((retVal = RioCapture(boardId, RIO_IM_1, FALSE, FALSE, TRUE, &SrcRect,
&DestRect,
        FIELD_WIDTH * FIELD_DEPTH, imagebh1, &Overlapped1)) != RIO_PENDING)
    {
        printf("RioCapture returned error 0x%x", retVal);
        RioScatterUnlock(imagebh1);
        RioScatterUnlock(imagebh0);
        free(image1);
        free(image0);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    while (!(CapDone0 && CapDone1))
    {
        retVal = RioGetOverlappedResult(&Overlapped0, &CapResult0, FALSE);
        if (retVal == RIO_OK)
            CapDone0 = TRUE;
        retVal = RioGetOverlappedResult(&Overlapped1, &CapResult1, FALSE);
        if (retVal == RIO_OK)
            CapDone1 = TRUE;
        if (kbhit())
        {
            ch = getch();
            break;
        }
    }
    if (!(CapDone0 && CapDone1))
    {
        if (CapDone0 == FALSE)
        {
            if (RioCaptureCancel(boardId, RIO_IM_0) != RIO_OK)
                printf("Capture 0 cancel failed\n");
            else
                printf("Capture 0 cancelled\n");
        } else
        {
            if (RioCaptureCancel(boardId, RIO_IM_1) != RIO_OK)
                printf("Capture 1 cancel failed\n");
            else
                printf("Capture 1 cancelled\n");
        }
    } else
    {
        printf("Captures done\n");
    }
    /* rest of program displays image on screen */
    if (CapDone0 && CapDone1)
    {
        _setvideomode(_MRES256COLOR);   /* switch to 320x200x8 */
        _clearscreen(_GCLEARSCREEN);

    /* load a standardgrey palette */
        for (i = 0; i < 256; i++)
            colors[i] = RGB(i >> 2, i >> 2, i >> 2);
        _myremapallpalette(colors);
/* init vga */
#if defined(PHARLAP)
        /* map the VGA in the data segment at offset 64K */
        screenPtr = (uchar *) 0x10000;
        if((retVal=_dx_map_pgsn(screenPtr,0x10000L,0xA0000L))!= 0)
        {
            RioScatterUnlock(imagebh1);
            RioScatterUnlock(imagebh0);
            free(image1);
            free(image0);
            free(pRiodl);
            RioClose();
            printf("Can't map VGA physical memory,error:%d\n", retVal);
            exit(1);
        }
#elif defined(DOS4GW)
        screenPtr = (uchar *)SCREEN_AREA;
#endif
        dataPtr = (uchar *)image0;
        if((FIELD_WIDTH==SCREEN_WIDTH)&&(FIELD_HEIGHT==SCREEN_HEIGHT))
```

```
            memcpy(screenPtr,dataPtr, SCREEN_HEIGHT*SCREEN_WIDTH);
        else
        {
            for (h = 0; h < SCREEN_HEIGHT; h++)
                memcpy(screenPtr  +  h*SCREEN_WIDTH,  dataPtr  +  h*FIELD_WIDTH,
SCREEN_WIDTH);
        }
        while (!kbhit());
        ch = getch();
        dataPtr = (uchar *)image1;
        if ((FIELD_WIDTH == SCREEN_WIDTH) && (FIELD_HEIGHT == SCREEN_HEIGHT))
            memcpy(screenPtr,dataPtr, SCREEN_HEIGHT*SCREEN_WIDTH);
        else
        {
            for (h = 0; h < SCREEN_HEIGHT; h++)
                memcpy(screenPtr  +  h*SCREEN_WIDTH,  dataPtr  +  h*FIELD_WIDTH,
SCREEN_WIDTH);
        }
        while (!kbhit());
        ch = getch();
    }
    _setvideomode(_TEXTC80);
    RioScatterUnlock(imagebh1);
    RioScatterUnlock(imagebh0);
    free(image1);
    free(image0);
    free(pRiodl);
    RioClose();
}
```

21. Contoh program untuk melakukan penangkapan citra secara kontiniu, yang meng-*overlap* penangkapan citra yang terakhir.

```
#define FIELD_CAPTURE
#define SCALER          RIO_HPS
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#if defined(__WATCOMC__)
#include <time.h>
#include <dos.h>
#include <conio.h>
#endif
#include <etype.h>
#include <edef.h>
#include <riotype.h>
```

```
#include <riodef.h>
#include <rioerror.h>
#include <rio.h>

#define FIELD_XOFF        0
#define FIELD_YOFF       16
#define FIELD_WIDTH     384
#define FIELD_HEIGHT    288
#define FRAME_XOFF        0
#define FRAME_YOFF       16
#define FRAME_WIDTH     768
#define FRAME_HEIGHT    576
#if defined(FIELD_CAPTURE)
#define IN_FRAME        RIO_FIELD
#define IN_XOFF         FIELD_XOFF
#define IN_YOFF         FIELD_YOFF
#define IN_WIDTH        FIELD_WIDTH
#define IN_HEIGHT       FIELD_HEIGHT
#define OUT_WIDTH       2*FIELD_WIDTH
#else
#define IN_FRAME        RIO_FRAME
#define IN_XOFF         FRAME_XOFF
#define IN_YOFF         FRAME_YOFF
#define IN_WIDTH        FRAME_WIDTH
#define IN_HEIGHT       FRAME_HEIGHT
#define OUT_WIDTH       FRAME_WIDTH
#endif

#define SCREEN_AREA     0xa0000
#define SCREEN_WIDTH    320
#define SCREEN_HEIGHT   200

#define CAP_BUFFERS     4
#define RGB(r,g,b) ((long) ((b) << 16) | ((g) << 8) | (r))
#define delay_one_usec() inp(0x80)
long _myremappalette(short pixval, long color)
{
 uchar ok, c[3], oldc[3];
 int i, j;
 long oldcolor;
 for (i = 0; i < 3; i++)
 {
  c[i] = color & 0xFF;
  color >>= 8;
 }
 _disable();
 outp(0x3C7, pixval);
 delay_one_usec();
 for (i = 0; i < 3; i++)
  oldc[i] = inp(0x3C9);
 for (i = 0, ok = 0; (i < 10) && !ok; i++) /* 10 retries max */
 {
  outp(0x3C8, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   outp(0x3C9, c[j]);
   delay_one_usec();
  }
  ok = 1;
  outp(0x3C7, pixval);
  delay_one_usec();
  for (j = 0; j < 3; j++)
  {
   if (inp(0x3C9) != c[j])
    ok = 0;
   delay_one_usec();
  }
 }
 _enable();
 for (i = 2, oldcolor = 0; i >= 0; i--)
 {
  oldcolor |= oldc[i];
  oldcolor <<= 8;
 }
 if (!ok)
  return(-1);
 else
  return(oldcolor);
}
short _myremapallpalette(long *colors)
{
 int i;
 for (i = 0; i < 256; i++)
  if (_myremappalette(i, colors[i]) == -1)
   return(-1);
 return(0);
}
void usage()
{
   exit(1);
}
void errorval()
{
   printf("\n");
   exit(9);
}
int32 DoRioSetup(int32 boardId)
{
    int32 retVal;
    RIO_MODULE Module;
    Module.Mode = RIO_BW;
    Module.Def.Bw.InputModule = RIO_IM_0;
    Module.Def.Bw.Scaler = SCALER;
    Module.Def.Bw.VideoStandard = RIO_CCIR;
    Module.Def.Bw.TvOrVtr = RIO_VTR;
    Module.Def.Bw.FieldOrFrame = IN_FRAME;
    Module.Def.Bw.OutputFormat = RIO_Y8;
    if ((retVal = RioSetInputModule(boardId, &Module)) != RIO_OK)
        return (retVal);
    if((retVal = RioSelectCamera(boardId, RIO_IM_0, 0)) != RIO_OK)
        return (retVal);
    return (retVal);
}
void main(int argc, char *argv[])
{
    PRIODL pRiodl;
    int32 boardId;
    int32 retVal;
    int32 h;
    int32 i, j;
    RECT SrcRect = {IN_XOFF, IN_YOFF, IN_XOFF + OUT_WIDTH, IN_YOFF + IN_HEIGHT};
    RECT DestRect = {0, 0, FIELD_WIDTH, FIELD_HEIGHT};
    void *image[CAP_BUFFERS];
    HANDLE imagebh[CAP_BUFFERS];
    int32 curcapbuf, oldcapbuf;
    uchar *screenPtr, *dataPtr;
    char ch = 0, capDone;
    long colors[256];
    double start, end;
    uint32 fieldcount;
    OVERLAPPED Overlapped[CAP_BUFFERS];
    int32 CapResult;
    setvbuf( stdout, NULL, _IONBF, 0 );
    if (RioOpen() != RIO_OK)
    {
        printf("Could not find Rio");
        exit(1);
    }
    printf("found Rio\n");
    pRiodl = (PRIODL) calloc(RioSizeOfRiodl(), 1);
    retVal = RioCreateRiodl(pRiodl);
    if (retVal != RIO_OK)
    {
        printf("RioCreateRiodl returned error 0x%x", retVal);
        free(pRiodl);
        RioClose();
        exit(2);
    }
    boardId = pRiodl->BoardId[0];
    if (DoRioSetup(boardId) != RIO_OK)
    {
        free(pRiodl);
        RioClose();
        exit(2);
    }
    for (i = 0; i < CAP_BUFFERS; i++)
    {
 image[i] = calloc(IN_HEIGHT * IN_WIDTH, 1);
        if   ((retVal   =   RioScatterLock(image[i],   IN_HEIGHT   *   IN_WIDTH,
&imagebh[i])) != RIO_OK)
        {
            free(image[i]);
            for (j = i - 1; j >= 0; j--)
            {
                RioScatterUnlock(imagebh[j]);
                free(image[j]);
            }
            free(pRiodl);
            RioClose();
            exit(2);
        }
}
/* init vga */
#if defined(PHARLAP)
    /* map the VGA in the data segment at offset 64K */
    screenPtr = (uchar *) 0x10000;
    if((retVal=_dx_map_pgsn(screenPtr, 0x10000L, 0xA0000L)) != 0)
    {
        for (i = 0; i < CAP_BUFFERS; i++)
        {
            RioScatterUnlock(imagebh[i]);
            free(image[i]);
        }
        free(pRiodl);
        RioClose();
        printf("Can't map VGA physical memory, error: %d\n", retVal);
        exit(1);
    }
#elif defined(DOS4GW)
    screenPtr = (uchar *)SCREEN_AREA;
#endif
    fieldcount = 0;
    start = utimer();
    curcapbuf = 0;
    if  ((retVal  =  RioCapture(boardId,  RIO_IM_0,  FALSE,  FALSE,  TRUE,  &SrcRect,
&DestRect,
        IN_WIDTH, imagebh[curcapbuf], &Overlapped[curcapbuf])) != RIO_PENDING)
    {
        for (i = 0; i < CAP_BUFFERS; i++)
        {
            RioScatterUnlock(imagebh[i]);
            free(image[i]);
        }
        free(pRiodl);
        RioClose();
        printf("Capture failed\n");
        exit(2);
    }
    capDone = 0;
    _setvideomode(_MRES256COLOR);   /* switch to 320x200x8 */

/* load a standardgrey palette */
    for (i = 0; i < 256; i++)
        colors[i] = RGB(i >> 2, i >> 2, i >> 2);
    _myremapallpalette(colors);
    while (1)
    {
  if (kbhit())
        {
            ch = getch();
        }
        if (ch == 0x1b)
        {
            break;
        }
        if((retVal= RioGetOverlappedResult(&Overlapped[curcapbuf],
           &CapResult, TRUE)) == RIO_OK)
        {
            fieldcount++;
 oldcapbuf = curcapbuf++;
            if (curcapbuf == CAP_BUFFERS)
                curcapbuf = 0;
            if((retVal = RioCapture(boardId, RIO_IM_0, FALSE, FALSE, TRUE,
               &SrcRect, &DestRect, IN_WIDTH, imagebh[curcapbuf],
               &Overlapped[curcapbuf])) != RIO_PENDING)
            {
                break;
            }
            dataPtr = (uchar *)image[oldcapbuf];
            for (h = 0; h < SCREEN_HEIGHT; h++)
            memcpy(screenPtr+h*SCREEN_WIDTH,dataPtr+h*IN_WIDTH, SCREEN_WIDTH);
            capDone = 0;
        } else if (retVal == RIO_CAPTURE_ERROR)
        {
            capDone = 1;
            break;
        }
    }
    if (capDone == 0)
    {
        double captime;
        captime = utimer();
        while (utimer() < (captime + 0.1))
        {            if((retVal=RioGetOverlappedResult(&Overlapped[curcapbuf],
               &CapResult, TRUE)) == RIO_OK)
            {
                capDone = 1;
                fieldcount++;
                break;
            }
        }
    }
    end = utimer();
    _setvideomode(_TEXTC80);
    if (capDone == 0)
    {
        if (RioCaptureCancel(boardId, RIO_IM_0) != RIO_OK)
            printf("Capture cancel failed\n");
        else
```

```
            printf("Capture cancelled\n");
    } else
    {
        printf("Capture done\n");
    }
#if defined(FIELD_CAPTURE)
    printf("%lu fields in %6.3f sec", fieldcount, dutime(start, end));
    if ((fieldcount != 0) && (end != start))
        printf(" = %6.3f fields / sec", fieldcount / dutime(start, end));
#else
    printf("%lu frames in %6.3f sec", fieldcount, dutime(start, end));
    if ((fieldcount != 0) && (end != start))
        printf(" = %6.3f fields / sec", 2*fieldcount / dutime(start, end));
#endif
    for (i = 0; i < CAP_BUFFERS; i++)
    {
        RioScatterUnlock(imagebh[i]);
        free(image[i]);
    }
    free(pRiodl);
    RioClose(); }
```