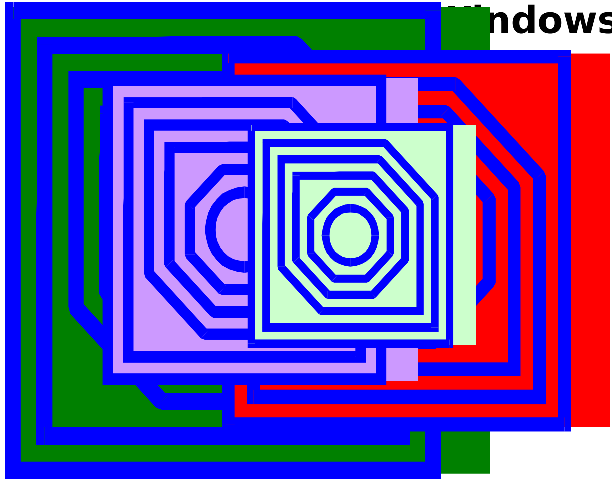
CHAPI/VAX-Qbus

The CHARON-VAX Application Programming Interface

(CHAPI) for Qbus peripheral

ndows



SRI

Software Resources International

CHAPI/VAX-Qbus

The CHARON-VAX Application Programming Interface

(CHAPI) for Qbus peripheral emulation in Windows

© 2003 -2004 Software Resources International S.A

Copyright © 2003 - 2004 Software Resources International S.A.

All rights reserved. Under the copyright laws, this publication and the software described within may not be copied, in whole or in part, without the written consent of Software Resources International. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold. Under the law, copying includes translating into another language or format.

The CHARON name and logo is a trademark of Software Resources International. PDP-11, VAX, MicroVAX and Qbus are trademarks of the Hewlett-Packard Company. Windows is a registered trademark in the United States and other countries, licensed exclusively through Microsoft Corporation, USA. All other trademarks and registered trademarks are the property of their respective holders.

Software Resources International makes no representations that the description of the CHAPI/VAX-Qbus interface in this publication will not infringe on existing or future patent rights, nor does this description imply the availability of relevant products or granting of licenses to make, use, or sell equipment or software in accordance with the description.

Document number: 30-16-013

16 June 2004

Printed in Switzerland.

Contents

Conventions....................vi

Chapter 1 Introduction....................1

1.1 Scope....................1
1.2 Implementation requirements....................2

Chapter 2 CHAPI structure....................3

2.1 CHAPI requirements and solution architecture....................3
2.2 Loadable component naming conventions....................4
2.3 Component loading and initialization....................4
2.4 Communication context binding....................6
2.5 Run-time communication....................7
2.6 The CHAPI communication context descriptors....................10
2.7 Initialization steps....................18
2.8 Run-time execution contexts....................19

Chapter 3 CHAPI operation....................21

3.1 Reading the device control and status register....................21
3.2 Writing device control and status register....................21
3.3 Creating additional I/O space....................22
3.4 Destroying I/O space....................23
3.5 Changing I/O space bus location....................24
3.6 Requesting a bus interrupt....................25
3.7 Removing a bus interrupt request....................26
3.8 Bus interrupt acknowledge....................27
3.9 Reading emulator memory (DATA-OUT DMA)....................28
3.10 Writing emulator memory (DATA-IN DMA)....................29
3.11 Synchronizing execution with the VAX CPU thread....................30
3.12 Synchronized invocation with the VAX CPU thread 30
3.13 Delaying synchronized execution....................31
3.14 Delayed synchronized invocation....................32
3.15 Processing a bus power-up condition....................33
3.16 Processing a bus power-down condition....................34
3.17 Processing a bus reset condition....................34
3.18 Changing the configuration....................35
3.19 Message logging....................35

© 2003 -2004 Software Resources International S.A

3.20 Decrypting critical data..36

1. CHAPI.H..38

2. LPV11 implementation..44

A.2 LPV11.CXX source file..44

1. VAXPrint application..65

A.3 VAXPRINT.CPP source file..65

A.4 VAXPRINT.H source file..83

A.5 WINSOCK.CPP source file..87

A.6 WINSOCK.H source file..93

A.7 Configuration file entry..94

Index..95

Conventions

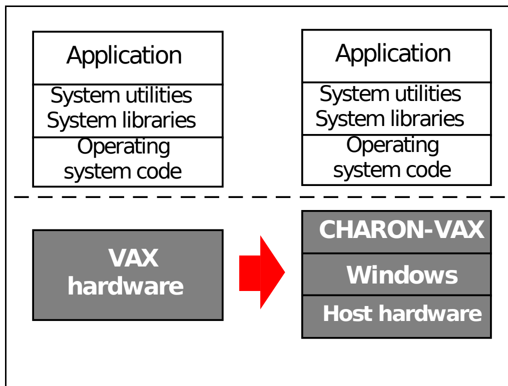
Throughout this manual these conventions are followed:

Notation	*Description*
`$ or >`	The dollar sign or the right angle bracket in interactive examples indicates operating system prompt.
User Input	Bold type in examples indicates source code.
<path>	Bold type enclosed by angle brackets indicates command parameters and parameter values.
`[]`	In syntax definitions, brackets indicate items that are optional.
...	In syntax definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.

Chapter 1 Introduction

1.1 Scope

CHARON-VAX is the generic name of a family of VAX system emulators made by Software Resources International. These emulators provide an accurate model of a complete VAX system: CPU, memory, disks, tapes, serial lines, and other controllers.



Several implementations of CHARON-VAX allow runtime access to the emulated peripheral busses using an application programming interface (CHAPI). This CHAPI concept was originally developed for the CHARON-11 PDP-11 emulator. The CHAPI/11 implementation was successfully used by our resellers to extend the emulator functionality with peripheral device support not available in the standard product release.

The concept of the CHAPI/11 has also been implemented for CHARON-VAX peripherals on the emulated Qbus. The CHAPI/VAX-Qbus version (referred to as CHAPI further on) described in this manual covers several CHARON-VAX implementations for a Windows 2000/XP host system. The use of CHARON Application Programming Interfaces for other host platforms, emulated VAX systems or other peripheral busses is not covered in this manual.

© 2003 -2004 Software Resources International S.A

1.2 Implementation requirements

Every Qbus peripheral has a specific bus interface through which it communicates with the VAX. In order to implement a Qbus peripheral with CHAPI you must have access to the hardware design details of the peripheral. For standard peripherals this is often described in its user guide or visible from the device driver sources, for custom peripheral designs you need the original design data. Design details include register definitions, interrupt vectors, DMA features and programming and test information.

For a standard VAX operating system to work with a CHAPI based peripheral implementation, the user written controller emulation must present the same CSRs, VECTORs and DMA features of its hardware equivalent. A full understanding of the device operation should be acquired before proceeding to implement an emulation of of a peripheral device using CHAPI.

The implementation of a user written Qbus peripheral device using CHAPI usually has the form of a C++ WIN32 DLL. Good C++ design skills are required to successfully implement and test such a CHAPI based peripheral. Appendix B and C list as an example an LPV11 emulation implementation mapping the output to the default Windows host printer.

CHAPI is a licensed option for certain CHARON-VAX emulators. Licensing CHAPI does not imply any verification or approval of Software Resources International that emulation of a peripheral using CHAPI is technically possible, will work as intended or will not infringe on any patent or licensing rule.

 © 2003 - 2004 Software Resources International S.A

Chapter 2 CHAPI structure

2.1 CHAPI requirements and solution architecture

The CHAPI provides the ability to load dynamically (at run-time, during configuration phase) additional emulator components that implement customer specific devices (for the QBUS in the version discussed in this manual). The CHAPI architecture defines a programming interface of communication between such a loadable component and the core of the CHARON-VAX emulator. This interface must contain the following elements:

- Inform the loadable component when it is necessary to create, initialize, terminate, and destroy instances of a device.
- Provide a means of configuring instances of a device as specified in the emulated VAX configuration.
- Deliver bus signals (such as BUS RESET, IRQ ACKNOWLEDGE) to instances of a device.
- Provide access of the emulated VAX CPU(s) to device control/status registers for each instance of device.
- Deliver bus requests (interrupt requests) on behalf of an instance of a device.
- Provide a means of reading/writing memory for DMA capable devices.
- Provide a means of license verification in order to protect the loadable component from uncontrolled distribution.
- Provide a means of message logging coordinated with message logging of the CHARON-VAX emulator itself.

© 2003 -2004 Software Resources International S.A

To meet the above needs the loadable components are implemented as .DLL modules in the Windows host platform. They define a set of procedures, data structures, and behaviors that allow the core of the CHARON-VAX emulator and the loadable component to communicate as required.

2.2 Loadable component naming conventions

Each loadable component must have a name, typically assigned by the developer of the component. This name must be sufficiently unique to allow users to identify the component. As far as the loadable component usually represents a class of peripheral devices, it is recommended to derive the name of the loadable component from the name of the device class.

The name of the .DLL module is constructed as follows:

<*COMPONENT-NAME*>.DLL

where <*COMPONENT-NAME*> represents the name of the loadable component. The use of delimiting characters in <*COMPONENT-NAME*> (spaces, tabs, and other “invisible” characters) is not allowed.

2.3 Component loading and initialization

.The CHARON-VAX emulator is responsible for loading a loadable component, using the Win32 API. To do this, CHARON-VAX processes a pair of **load** and **set** directives in the corresponding configuration file:

load chapi <*CFG-NAME*>
set <*CFG-NAME*> dll=<*PATH-TO-DLL*>

where <*CFG-NAME*> and <*PATH-TO-DLL*> are the relevant values for the component. After loading the module into memory, the

© 2003 - 2004 Software Resources International S.A

CHARON-VAX emulator core creates the necessary context and calls the module initialization routine.

Note that if a particular CHAPI loadable component is configured to load any more than once, the corresponding .DLL module is loaded only once, but the initialization routine is called for each configured instance. For example, suppose the configuration file contains the following lines:

```
load chapi TTA
set TTA dll=dl11.dll
load chapi TTB
set TTB dll=dl11.dll
```

Processing the above lines, the CHARON-VAX emulator loads the DL11.DLL once, and then calls the corresponding initialization routine first for TTA and then for TTB.

The moment of calling the initialization routine corresponds to the processing of the **set** command, when the value *<PATH-TO-DLL>* (**dl11.dll** in this case) is assigned to the dll parameter.

The module initialization routine has a predefined name and calling conventions, and must be declared in the source code (assuming the Visual C++ programming language) as follows:

```
__declspec(dllexport)
void * __cdecl <COMPONENT-NAME>_INIT
    (const chapi_in * ci, chapi_out * co, const char *
instance_name);
```

where *<COMPONENT-NAME>* represents the name of the loadable component (see also “Loadable component naming conventions” above). Note that the name of initialization routine must be converted to upper case. For example, the loadable module called dl11.dll shall declare its initialization procedure as follows:

```
__declspec(dllexport)
void * __cdecl DL11_INIT
```

(const chapi_in * ci, chapi_out * co, const char * instance_name);

The moment when the CHARON-VAX emulator calls the component initialization routine shall be considered by the loadable component as a request to confirm the creation of a new instance of the device provided by the component.

The component initialization routine responds to the confirmation request with its return value. The CHARON-VAX emulator considers any value other than zero as a confirmation, and a zero value as a rejection. In case of confirmation, the CHARON-VAX emulator uses this value as an opaque identifier. Later it passes this identifier as an additional parameter to certain procedures of the CHAPI protocol.

2.4 Communication context binding

The CHAPI protocol defines a communication channel between the CHARON-VAX emulator and the loadable component. Since the emulator can load multiple loadable components, and each component can create several instances of devices, this can result in many communication channels.

Each communication channel binds the CHARON-VAX emulator to one instance of a device provided by a loadable component. The communication channel is described at each end by the communication contexts. Each side of the communication is responsible for creating and supporting its own contexts and setting up those contexts on behalf of the other side. The CHAPI protocol defines a set of rules of this context setup and support, which is called context binding.

The CHARON-VAX emulator creates its own communication context when processing the load command to load its configuration. It creates a descriptor of the chapi_in communication context (the chapi_in descriptor for short), fills it with zeros, and then fills certain fields with non-zero values. It also creates a descriptor of the

 © 2003 - 2004 Software Resources International S.A

chapi_out communication context (the chapi_out descriptor) and fills it with zeros. At that moment the CHARON-VAX emulator is in principle ready to issue a creation confirmation request by calling the component initialization routine.

The loadable component shall create its own contexts (if any) in its initialization routine, just before confirming the creation of new device instance. The loadable component shall fill all required fields of the provided descriptor of the chapi_out communication context with their respective values.

The loadable component is allowed to remember the data path to both the chapi_in and the chapi_out descriptors provided to the component by the CHARON-VAX emulator. The loadable component is not allowed to modify any fields in the descriptor of the chapi_in communication context that is provided to the component by the CHARON-VAX emulator.

Upon receiving a confirmation, the CHARON-VAX emulator completes setting up its own context by filling the remaining fields of the descriptor of the chapi_in communication context. The CHARON-VAX emulator is allowed to terminate setting up its contexts when receiving a rejection from the initialization routine of the loadable component.

The contexts binding process shall be considered done if, and only if:

1. The whole configuration has been loaded, and
2. The loadable component has confirmed the creation of a device instance.

2.5 Run-time communication

The run-time communication over the CHAPI communication channel is defined in terms of operations or transactions. Each operation is either initiated by the CHARON-VAX emulator or by the instance of the device provided by the loadable component. The side

originating the operation is called the initiator of the operation. Each operation must be processed by the other side of the communication channel, which is called the target of the operation.

Both the CHARON-VAX emulator and the loadable component are designed to work in a multithreaded environment. Therefore no assumptions shall be made regarding the thread in which any particular operation is initiated. Nevertheless, certain operations put some restrictions on the way in which both sides of the CHAPI communication are allowed to initiate those operations.

Before the operation is initiated, the initiator shall prepare the operation context. When prepared, the initiator calls the corresponding routine provided by the counterpart to process the operation. Entry points to these routines are supplied in the corresponding fields of the chapi_in and the chapi_out descriptors by both sides of the CHAPI communication.

The CHAPI introduces the following types of operations:

- Access to the device control and status registers (both read and write). Such an operation is only initiated by the thread interpreting VAX CPU instructions. Therefore the target shall finish processing such an operation as soon as possible. Note that the CHARON-VAX emulator is allowed to run several threads interpreting VAX CPU instructions.
- Request a bus interrupt. Such an operation is only initiated by the device instance. The target of the operation must remember the bus interrupt request.
- Request for a bus interrupt acknowledge. Such an operation is only initiated by the thread(s) interpreting VAX CPU instructions. Therefore the target shall finish processing such an operation as soon as possible.
- Direct memory access. Such an operation is only initiated by the device instance.

 © 2003 - 2004 Software Resources International S.A

- Synchronization request. Such an operation is only initiated by the device instance. The target of the operation must remember the synchronization request.
- Synchronization request acknowledgement. Such an operation is only initiated by the thread interpreting VAX CPU instructions. Therefore the target shall finish processing such an operation as soon as possible.
- A request for processing bus power events.
- A request for processing bus reset events.
- A request for changing the configuration.
- Message log request.
- Protection and license verification.

The side of the CHAPI communication that is a target of a transaction provides a routine for processing the transaction. This routine is identified by an entry point stored in either the chapi_in or the chapi_out descriptors, depending on the operation. Thus transactions initiated by the CHARON-VAX emulator are processed by routines specified in the chapi_out descriptor, and transactions initiated by the device instance are processed by routines specified in the chapi_in descriptor. This is why the device instance must memorize the data path to at least the chapi_in descriptor.

Both the CHARON-VAX emulator and the loadable component are allowed to omit (that is set to 0) some or all entry points in the chapi_in and the chapi_out descriptors respectively. An absent entry point for a certain operation means that the operation is not supported and shall not be initiated by the counterpart. An entry point set to zero is considered absent.

© 2003 - 2004 Software Resources International S.A

2.6 The CHAPI communication context descriptors

The paragraph covers the details of the communication context descriptors and the component initialization process.

The CHAPI_IN communication context descriptor

The chapi_in descriptor contains entry points to routines provided by the CHARON-VAX emulator, as well as base address of control and status registers (CSRs) and base interrupt vector, which might be specified in the configuration file (see below). The chapi_in descriptor structure is defined as follows:

```
typedef struct {
    void * const    context;
    unsigned int    base_b_address;
    unsigned int            base_i_vector;
    void            (CHAPI * put_ast)
                      (const chapi_in * ci,
                       unsigned long           delay,
                       ast_handler             fun,
                       void *          arg1,
                       int             arg2);
    void            (CHAPI * put_sst)
                      (const chapi_in * ci,
                       unsigned long           delay,
                       sst_handler             fun,
                       void *          arg1,
                       int             arg2);
    void            (CHAPI * put_irq)
                      (const chapi_in * ci,
                       unsigned int            vec,
                       unsigned long           delay,
                       irq_handler             fun,
```

 © 2003 - 2004 Software Resources International S.A

```
                        void *          arg1,
                        int             arg2);
void            (CHAPI * clear_irq)
                       (const chapi_in * ci,
                        unsigned int            vec);
unsigned int (CHAPI * read_mem)
                       (const chapi_in * ci,
                        unsigned int            addr,
                        unsigned int            len,
                        char *          buf);
unsigned int (CHAPI * write_mem)
                       (const chapi_in * ci,
                        unsigned int            addr,
                        unsigned int            len,
                        const char *            buf);
io_space_id_t           (CHAPI * create_io_space)
                       (const chapi_in * ci,
                        unsigned int            addr,
                        unsigned int            len);
void            (CHAPI * move_io_space)
                       (const chapi_in * ci,
                        io_space_id_t           space_id,
                        unsigned int            addr,
                        unsigned int            len);
void            (CHAPI * destroy_io_space)
                       (const chapi_in * ci,
                        io_space_id_t
space_id);
void            (CHAPI * decrypt_data_block)
                       (const chapi_in * ci,
                        void *          buf,
                        unsigned int            len);
void            (CHAPI * log_message)
                       (const chapi_in * ci,
```

```
                                const char *            buf,
                                unsigned int            len);
    } chapi_in;
```

The **context** field is defined by the CHAPI and initialized by the CHARON-VAX emulator solely for its own use and shall not be changed as well as used by the component in any way.

The **base_b_address** field contains the starting bus address of the device instance I/O region. Usually it is an address of the device's control and status register (CSR). The CHARON-VAX emulator provides a value in this field upon completion loading the configuration. Initially it is set to 0. The CHAPI allows user to override this value with configuration parameters (see below).

The **base_i_vector** field contains the starting interrupt vector address, assigned to the device. The CHARON-VAX emulator provides a value in this field upon completion loading the configuration. Initially it is set to 0. The CHAPI allows user to override this value with configuration parameters (see below).

The **put_ast** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **put_sst** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **put_irq** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially

 © 2003 - 2004 Software Resources International S.A

set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **clear_irq** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **read_mem** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **write_mem** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **create_io_space** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **move_io_space** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization

© 2003 - 2004 Software Resources International S.A

routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **destroy_io_space** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **decrypt_data_block** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

The **log_message** field contains an entry point to the corresponding routine. The CHARON-VAX emulator provides a value in this field before calling the loadable component initialization routine. Initially set to 0. Later but before calling the initialization routine the CHARON-VAX emulator is allowed to put non-zero value into the field.

Note that the CHARON-VAX emulator is not required to initialize all the fields of the chapi_in communication context descriptor before calling the initialization routine. The CHARON-VAX emulator is allowed to initialize these fields later. More details follow.

The CHAPI_OUT communication context descriptor

The chapi_out descriptor contains entry points to routines provided by the loadable component, as well as address range of control and status register I/O space and number of interrupt vectors. The chapi_out descriptor structure is defined as follows:

```
typedef struct {
```

 © 2003 - 2004 Software Resources International S.A

```
        void * const    context;
        unsigned int    b_address_range;
        unsigned int    n_of_i_vector;
        unsigned int    i_priority;
        void            (CHAPI * stop)
                          (const chapi_out *     co);
        void            (CHAPI * start)
                          (const chapi_out *     co);
        void            (CHAPI * reset)
                          (const chapi_out *     co);
        void            (CHAPI * write)
                          (const chapi_out *     co,
                           unsigned int          addr,
                           int                   val,
                           bool                  is_byte);
        int             (CHAPI * read)
                          (const chapi_out *     co,
                           unsigned int          addr,
                           bool                  is_byte);
        int             (CHAPI * set_configuration)
                          (const chapi_out *     co,
                           const char *
        parameters);

    } chapi_out;
```

The **context** field is defined by the CHAPI solely for use by the loadable component. The CHARON-VAX emulator initializes the field with 0. Later it updates the field with the value returned by the component initialization routine. Afterwards the CHARON-VAX emulator does not make any attempts to use the value of the field in any way. The loadable component is supposed to use the **context** field to bind the communication context to private data structures.

The **b_address_range** field contains the length of the I/O space in bytes the CHARON-VAX emulator shall reserve for control and status registers of the device instance. The field is obligatory. So the loadable component shall provide a non-zero value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine). The combination of the **base_b_address** and **b_address_range** shall meet the following requirements:

- The length of address range must be a power of two. Or more formally:

  (b_address_range & (b_address_range - 1)) == 0

- The address range shall be naturally aligned. Which means that the **base_b_address** shall be aligned to the boundary, which is multiple of **b_address_range**. Or more formally:

  (base_b_address + (b_address_range - 1))
  == (base_b_address | (b_address_range - 1))

The **n_of_i_vector** field contains number of interrupt vectors the CHARON-VAX emulator shall allocate for the device instance. The field is obligatory, and the loadable component shall provide a non-zero value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **i_priority** field contains a priority at which the CHARON-VAX emulator shall process the interrupt requests originated by the device instance. The field is obligatory provided that **n_of_i_vector** field is non-zero, otherwise the field might be left unspecified. So the loadable component shall provide a non-zero value in this field if necessary when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **stop** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in

 © 2003 - 2004 Software Resources International S.A

this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **start** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **reset** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **write** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **read** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

The **set_configuration** field contains an entry point to the corresponding routine. The field is optional. The loadable component provides a value in this field when processing the request to confirm creation of device (i.e. in the corresponding ***<COMPONENT_NAME>*****_INIT** routine).

Note that the component initialization routine is required to properly initialize **b_address_range**, **n_of_i_vector**, **i_priority**, **start**, and **stop** fields of the chapi_out communication context descriptor. Other fields might be setup later.

© 2003 - 2004 Software Resources International S.A

2.7 Initialization steps

Now when structure of communication contexts is available, it is time to present more information on component initialization process . The initialization steps are as follows:

1. As soon as the CHARON-VAX emulator has finished processing the load command, it creates necessary internal structures representing the device and allocates resources for both the chapi_in and the chapi_out communication context descriptors.

2. The communication context descriptors are initialized with all zeros.

3. The CHARON-VAX emulator then creates all required private data structures and binds the communication context to them properly initializing the **context** field of the chapi_in communication context descriptor.

4. If the corresponding routines are supported the CHARON-VAX emulator initializes the **decrypt_data_block** and the **log_message** fields of the chapi_in communication context descriptor. So the component initialization routine might use them.

5. As soon as the CHARON-VAX emulator assigns a value to the **dll** parameter, it loads the .DLL module, if necessary, and calls the component initialization routine.

6. As soon as the component initialization routine returns, the value returned is checked against zero. If the value is zero then the CHARON-VAX emulator considers the initialization failed, releases all the allocated so far resources (if any), and reports an error. Otherwise the CHARON-VAX emulator updates the context field of the chapi_out communication context descriptor with this value and proceeds with initialization.

© 2003 - 2004 Software Resources International S.A

2.8 Run-time execution contexts

The CHAPI is based on a multithreaded software execution model. This means that procedures defined by the CHAPI run in different threads. The CHAPI selects the CPU instruction interpretation thread as a special execution context, so that all the procedures invoked in that thread are invoked in the execution context synchronized to the CPU instruction interpretation thread.

The CHAPI restricts the use of certain procedures to execution context in the following way:

- All the procedures identified by entry points stored in **put_sst**, **put_irq**, **clear_irq**, and **move_io_space** fields of a chapi_in communication context descriptor shall be invoked in the execution context synchronized to the CPU instruction interpretation thread.
- All the procedures identified by entry points stored in **create_io_space** and **destroy_io_space** fields of a chapi_in communication context descriptor shall be invoked in the same execution context. To guarantee that the loadable component is required to call those routines only from the routines identified by the **start** and the **stop** fields of the chapi_out communication context descriptor.

The CHAPI also guarantees that:

- All the procedures identified by entry points stored in **read**, **write**, and **reset** fields of a chapi_out communication context descriptor are invoked in the execution context synchronized to the CPU instruction interpretation thread.
- All the procedures identified by **fun** argument in the procedure calls to **put_ast**, **put_sst**, and **put_irq** procedures, identified by corresponding fields of a chapi_in communication context

© 2003 - 2004 Software Resources International S.A

descriptor, are invoked in the execution context synchronized to the CPU instruction interpretation thread.

- All the procedures identified by entry points stored in start and stop fields of a chapi_out communication context are invoked in the same execution context.
- Any procedure call does not change the execution context.
- Each thread has its own execution context.

 © 2003 - 2004 Software Resources International S.A

Chapter 3 CHAPI operation

3.1 Reading the device control and status register

The operation of reading a device control and status register belongs to the class of operations previously called “Access to device control and status registers”. The CHARON-VAX emulator initiates such an operation. More precisely, one of the VAX CPU instruction interpretation threads is the initiator. The device is considered to be a target of the operation.

In order to perform the operation, the initiator invokes a routine identified by the **read** field of the chapi_out descriptor. The initiator provides in **addr** parameter a bus address of the register to read from, and in **is_byte** parameter the length of transaction. The procedure is invoked as follows:

```
const chapi_out * co = ...;
if (co->read) {
    val = co->read(co, addr, is_byte);
}
```

The example above shows that the device is not obliged to support the indicated operation.

3.2 Writing device control and status register

The operation of writing device control and status register belongs to the class of operations previously called “Access to device control and status registers”. The CHARON-VAX emulator initiates such an operation. More precisely, one of CPU instruction interpretation

threads is the initiator. The device is defined as a target of the operation.

In order to perform the operation, initiator invokes a routine identified by the **write** field of the chapi_out descriptor. The initiator provides in **addr** parameter a bus address of the register to read from, in **val** parameter a value to be written, and in **is_byte** parameter the length of transaction. The procedure is invoked as follows:

```
const chapi_out * co = …;
if (co->write) {
    co->write(co, addr, val, is_byte);
}
```

The example above shows that the device is not obliged to support the indicated operation.

3.3 Creating additional I/O space

The operation of creating additional I/O space belongs to the class of operations previously called “Access to device control and status registers”. Such an operation is initiated by the device instance. The CHARON-VAX emulator is defined as a target of the operation.

In order to perform the operation, initiator invokes a routine identified by the **create_io_space** field of the chapi_in descriptor. The initiator is to provide initial bus location of the I/O space in the **addr** and the **len** arguments. The combination of the **addr** and the **len** shall meet the following requirements:

- The length of address range indicated by the **len** must be a power of two. Or more formally:

  (len & (len - 1)) == 0

- The address range shall be naturally aligned. Which means that the **addr** shall be aligned to the boundary, which is multiple of the **len**. Or more formally:

 © 2003 - 2004 Software Resources International S.A

```
(addr + (len - 1)) == (addr | (len - 1))
```

The procedure is invoked as follows:

```
io_space_id_t sid = 0;
const chapi_in * ci = …;
if (ci->create_io_space) {
    sid = ci->create_io_space(ci, addr, len);
}
if (sid == 0) {
    /* failed to create the io space*/
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it gives a clue why the device instance is supposed to remember the chapi_in descriptor.

In the example above the device is supposed to store the (non-zero) result of **create_io_space** procedure. Later this value is to be used as I/O space identifier for subsequent calls (if any) to **move_io_space** and **destroy_io_space** procedures.

The device is to make sure that any additional I/O space created by the CHARON-VAX emulator on behalf of the device is destroyed with **destroy_io_space** procedure (see below). The suggested behavior is to create additional I/O spaces (if necessary) when processing bus power-up condition and destroy it when processing bus power-down condition (see below).

3.4 Destroying I/O space

The operation of destroying (additional) I/O space belongs to the class of operations previously called “Access to device control and status registers”. Such an operation is initiated by the device instance. The CHARON-VAX emulator is defined as a target of the operation.

In order to perform the operation, initiator invokes a routine identified by the **destroy_io_space** field of the chapi_in descriptor. The initiator is to provide an I/O space identifier in the **sid** argument. The procedure is invoked as follows:

```
io_space_id_t sid = ...;
const chapi_in * ci = ...;
if (ci->destroy_io_space) {
    ci->destroy_io_space(ci, sid);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. But nevertheless it is guaranteed that the CHARON-VAX emulator does either support both the **create_io_space** and the **destroy_io_space** procedures, or does not support any of them. Also it gives a clue why the device instance is supposed to remember the I/O space identifier returned in the previous call to **create_io_space** procedure (see above), and the chapi_in descriptor.

3.5 Changing I/O space bus location

The operation of changing (additional) I/O space bus location belongs to the class of operations previously called “Access to device control and status registers”. Such an operation is initiated by the device instance, when in execution context synchronized to the CPU instruction interpretation thread. The CHARON-VAX emulator is defined as a target of the operation.

In order to perform the operation, initiator invokes a routine identified by the **move_io_space** field of the chapi_in descriptor. The initiator is to provide an I/O space identifier in the **sid** argument and new bus location of the I/O space in the **addr** and the **len** arguments. The combination of the **addr** and the **len** shall meet the following requirements:

 © 2003 - 2004 Software Resources International S.A

- The length of address range indicated by the **len** must be a power of two. Or more formally:

  (len & (len - 1)) == 0

- The address range shall be naturally aligned. Which means that the **addr** shall be aligned to the boundary, which is multiple of the **len**. Or more formally:

  (addr + (len - 1)) == (addr | (len - 1))

The procedure is invoked as follows:

```
io_space_id_t sid = …;
const chapi_in * ci = …;
if (ci->move_io_space) {
    ci->move_io_space(ci, sid, addr, len);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it gives a clue why the device instance is supposed to remember the I/O space identifier returned in the previous call to **create_io_space** procedure (see above), and the chapi_in descriptor.

3.6 Requesting a bus interrupt

The operation of requesting a bus interrupt belongs to the group of operations previously called “Request for bus interrupt”. Such an operation is initiated by the device instance, when in execution context synchronized to the CPU instruction interpretation thread. The CHARON-VAX emulator is defined as a target of the operation. Actually the CPU instruction interpretation thread is supposed to respond to the bus interrupt request.

In order to perform the operation, initiator invokes a routine identified by the **put_irq** field of the chapi_in descriptor. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->put_irq) {
    ci->put_irq(ci, vec, delay, fun, arg1, arg2);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it gives a clue why the device instance is supposed to remember the chapi_in descriptor.

3.7 Removing a bus interrupt request

Removing a bus interrupt request belongs to the “Request for bus interrupt” group of operations. Such an operation is initiated by the device instance, when in context synchronized to the CPU instruction interpretation thread. The CHARON-VAX emulator is defined as a target of the operation. It is the task of the CPU instruction interpretation thread to respond to the operation of removing bus interrupt request.

To perform the operation, the initiator invokes a routine identified by the **clear_irq** field of the chapi_in descriptor. The initiator shall indicate through the **vec** parameter the vector of the interrupt requests to clear. The target shall clear all the interrupt requests previously originated by the initiator through the vector supplied. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->clear_irq) {
    ci->clear_irq(ci, vec);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it gives a clue why the device instance is supposed to retain the chapi_in descriptor.

 © 2003 - 2004 Software Resources International S.A

3.8 Bus interrupt acknowledge

Acknowledging the bus interrupt request is currently the only operation defined within the group of operations called “Request for bus interrupt acknowledge”. The CHARON-VAX emulator (one of the CPU instruction interpretation threads) initiates such an operation. The device instance is defined as a target of the operation.

In order to perform the operation, the initiator invokes a routine identified by the **fun** parameter, supplied during corresponding request for bus interrupt. See the “Requesting a bus interrupt” above. The procedure is invoked as follows:

```
irq_handler fun = …;
if (fun) {
    vec = fun(arg1, arg2);
}
```

The example above shows that the device is not obliged to support the indicated operation. Which means that it is allowed to supply 0 for **fun** parameter when requesting a bus interrupt through the **put_irq** entry point of the chapi_in descriptor. Note that the CHAPI defines additional argument supplied through the **arg1** and the **arg2** parameter to help the target dispatching requests.

The bus interrupt acknowledge procedure indicated by the **fun** is supposed to return the interrupt vector in case of acknowledgement. In this case the CHARON-VAX emulator uses the returned value as interrupt vector instead of the vector supplied through the previous call to the **put_irq** procedure. If the bus interrupt is not acknowledged, the bus interrupt acknowledge procedure shall return zero (0) indicating that the interrupt vector is not valid.

The CHAPI guarantees that the bus interrupt acknowledge procedure identified by the **fun** argument is invoked in the execution context synchronized to the CPU instruction interpretation thread.

© 2003 - 2004 Software Resources International S.A

3.9 Reading emulator memory (DATA-OUT DMA)

Reading the CHARON-VAX emulator memory belongs to the “Direct Memory Access” class of operations. This operation is initiated by the device instance. The CHARON-VAX emulator is defined as a target of the operation.

To perform the operation, the initiator invokes a routine identified by the **read_mem** field of the chapi_in descriptor. The initiator provides in the **addr** parameter a starting bus address of the memory region to read from, in the **len** parameter the length of transaction (length of transfer), and in the **buf** parameter the address of private buffer to transfer to. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->read_mem) {
    unsigned int t_len = ci->read_mem(ci, addr, len, buf);
    if (t_len < len) {
    }
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it shows why the device instance must remember the chapi_in descriptor. Note the result processing in the example above. The condition **(t_len < len)** might be considered by the initiator as an attempt to read non-existent memory.

 © 2003 - 2004 Software Resources International S.A

3.10 Writing emulator memory (DATA-IN DMA)

Writing the CHARON-VAX emulator memory belongs to the “Direct Memory Access” class of operations. Such an operation is initiated by the device instance. The CHARON-VAX emulator is defined as a target of the operation.

To perform the operation, the initiator invokes a routine identified by the **write_mem** field of the chapi_in descriptor. The initiator provides in the **addr** parameter a starting bus address of the memory region to write to, in the **len** parameter the length of transaction (length of transfer), and in the **buf** parameter the address of private buffer to transfer from. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->write_mem) {
    unsigned int t_len = ci->write_mem(ci, addr, len, buf);
    if (t_len < len) {
    }
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. It also shows why the device instance is must remember the chapi_in descriptor. Note the result processing in the example above. The condition **(t_len < len)** might be considered by the initiator as an attempt to write non-existent memory.

© 2003 - 2004 Software Resources International S.A

3.11 Synchronizing execution with the VAX CPU thread

Synchronizing execution with the VAX CPU instruction interpretation thread belongs to the “Synchronization request” class of operations. Such an operation is initiated by the device instance. The CHARON-VAX emulator is defined as the target of the operation. More precisely the VAX CPU instruction interpretation thread is required to respond to the operation.

To perform the operation, the initiator invokes a routine identified by the **put_ast** field of the chapi_in descriptor. The initiator provides in the **fun** argument an entry point to the procedure to be synchronized with the VAX CPU instruction interpretation thread, and in the **arg1** and the **arg2** arguments additional parameters to be passed later on to that procedure. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->put_ast) {
    ci->put_ast(ci, fun, arg1, arg2);
}
```

The example above shows that the CHARON-VAX emulator is not required to support the indicated operation. It also shows why the device instance must memorize the chapi_in descriptor.

3.12 Synchronized invocation with the VAX CPU thread

Synchronized invocation (with the CPU instruction interpretation thread) belongs to the class of operations called “Synchronization request acknowledge”. The CHARON-VAX emulator initiates such an operation. The device instance is defined as the target of the operation.

 © 2003 - 2004 Software Resources International S.A

To perform the operation, the initiator invokes a routine identified by the **fun** argument previously supplied in the corresponding request for synchronizing execution to the VAX CPU instruction interpretation thread. The initiator provides in the **arg1** and the **arg2** arguments additional parameters previously supplied together with the **fun** argument requesting for synchronizing execution to the VAX CPU instruction interpretation thread. The procedure is invoked as follows:

```
ast_handler fun = ...;
if (fun) {
    fun(arg1, arg2);
}
```

The example above shows that the device is not obliged to support the indicated operation. It is allowed to supply 0 for the **fun** parameter when requesting synchronizing execution with the VAX CPU instruction interpretation thread. Note that the CHAPI defines additional parameters supplied through the **arg1** and the **arg2** arguments to help the target dispatching requests.

The device shall consider the synchronized invocation request as a reaction of the CPU instruction interpretation thread of the CHARON-VAX emulator to the previously issued request for “Synchronizing execution”.

The CHAPI guarantees that the procedure identified by the **fun** argument is invoked in the execution context synchronized with the VAX CPU instruction interpretation thread.

3.13 Delaying synchronized execution

Delaying synchronized (with the CPU instruction interpretation thread) execution belongs to the “Synchronization request” class of operations. Such an operation is initiated by the device instance. The CHARON-VAX emulator is defined as a target of the operation. More precisely the CPU instruction interpretation thread is required to respond to the operation.

To perform the operation, the initiator invokes a routine identified by the **put_sst** field of the chapi_in descriptor. The initiator provides in **delay** argument a number of CPU instructions to interpret, in **fun** argument an entry point to the procedure to be synchronized with the CPU instruction interpretation thread after the specified number of CPU instructions are interpreted, and in the **arg1** and the **arg2** arguments additional parameters to be passed later on to that procedure. The **put_sst** routine is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->put_sst) {
    ci->put_sst(ci, delay, fun, arg1, arg2);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. Also it gives a clue why the device instance is supposed to remember the chapi_in descriptor.

3.14 Delayed synchronized invocation

The operation of delayed invocation is part of the class of operations called “Synchronization request acknowledge”. The CHARON-VAX emulator initiates such an operation. The device instance is defined as the target of the operation.

In order to perform the operation, the initiator invokes a routine identified by the **fun** argument - previously supplied in the corresponding request - delaying synchronized (with the CPU instruction interpretation thread) execution. The initiator provides in **arg1** and **arg2** arguments additional parameters previously supplied together with the **fun** argument requesting delaying synchronized (with the CPU instruction interpretation thread) execution. The procedure is invoked as follows:

```
sst_handler fun = ...;
if (fun) {
```

 © 2003 - 2004 Software Resources International S.A

```
        fun(arg1, arg2);
    }
```

The example above shows that the device is not obliged to support the indicated operation. This implies that it is allowed to supply 0 for the **fun** argument when requesting delaying synchronized (to the CPU instruction interpretation thread) execution. Note that the CHAPI defines additional parameters supplied through the **arg1** and **arg2** arguments to help the target dispatching requests.

The device shall consider the delayed synchronized invocation request as a reaction of the CPU instruction interpretation thread of the CHARON-VAX emulator to the previously issued request for “Delaying synchronized execution”.

The CHAPI guarantees that the procedure identified by the **fun** argument is invoked in the execution context synchronized to the CPU instruction interpretation thread.

3.15 Processing a bus power-up condition

Handling the bus power-up condition belongs to the class of operations called “Request for processing bus power events”. The CHARON-VAX emulator initiates such an operation when “powering” up the configured node.

In order to perform the operation, the initiator invokes a routine identified by the **start** field of the chapi_out descriptor. The procedure is invoked as follows:

```
    const chapi_out * co = ...;
    if (co->start) {
        co->start(co);
    }
```

The example above shows that the device is not obliged to support the indicated operation.

3.16 Processing a bus power-down condition

Handling the bus power-down condition belongs to the class of operations called “Request for processing bus power events”.. The CHARON-VAX emulator initiates such an operation when “powering” down the configured node.

In order to perform the operation, initiator invokes a routine identified by the **stop** field of the chapi_out descriptor. The procedure is invoked as follows:

```
const chapi_out * co = ...;
if (co->stop) {
    co->stop(co);
}
```

The example above shows that the device is not obliged to support the indicated operation.

3.17 Processing a bus reset condition

Handling the bus reset condition is currently the only operation defined within the class of operations called “Request for processing bus reset events”. CHARON-VAX initiates such an operation when resetting the I/O subsystem of the configured node. The CPU instruction interpretation thread is likely (but not necessarily) the initiator of such an operation.

In order to perform the operation, initiator invokes a routine identified by the **reset** field of the chapi_out descriptor. The procedure is invoked as follows:

```
const chapi_out * co = ...;
if (co->reset) {
    co->reset(co);
}
```

 © 2003 - 2004 Software Resources International S.A

The example above shows that the device is not obliged to support the indicated operation.

3.18 Changing the configuration

Changing the configuration is currently the only operation within the class of operations called “Request for changing the configuration”. The CHARON-VAX emulator initiates such an operation when loading the configuration. The CHAPI defines the additional configuration option **parameters**, and passes device specific configuration information to the device instance as indicated by the following example:

```
load chapi <CFG-NAME>
set <CFG-NAME> dll=<PATH-TO-DLL> parameters=”…”
```

To perform the operation, the initiator invokes a routine identified by the **set_configuration** field of the chapi_out descriptor. The initiator provides in the **parameters** argument a character string, as specified in the configuration for the **parameters** option. The procedure is invoked as follows:

```
const chapi_out * co = …;
if (co->set_configuration) {
    co->set_configuration(co, parameters);
}
```

The example above shows that the device is not obliged to support the indicated operation.

3.19 Message logging

Message logging is currently the only operation defined in the class of operations called “Message log request”.

© 2003 - 2004 Software Resources International S.A

The CHARON-VAX emulator enables the component that logs messages in the emulator's log file. Both the CHARON-VAX emulator and the loadable component may initiate such an operation, but only the emulator can be a target.

To perform the operation, the initiator invokes a routine identified by the **log_message** field of the chapi_in descriptor. The initiator provides a formatted message in a buffer, pointed to by **buf** argument, of length (in bytes) as indicated by **len** argument. The procedure is invoked as follows:

```
const chapi_in * ci = ...;
if (ci->log_message) {
    ci->log_message(ci, buf, len);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. It also shows why the loadable component instance must remember the chapi_in descriptor.

3.20 Decrypting critical data

The operation of decrypting protected data is currently the only operation defined in the class of operations called “Protection and license verification”. The CHARON-VAX emulator enables the component decrypting encrypted data . Both the CHARON-VAX emulator and the loadable component may initiate such an operation, but only the emulator can be a target.

In order to perform the operation, the initiator invokes a routine identified by the **decrypt_data_block** field of the chapi_in descriptor. The initiator provides a buffer containing encrypted data. The length of the buffer (in bytes) must be a multiple of 8. The buffer's address and length are indicated by **buf** and **len** arguments respectively. The procedure is invoked as follows:

 © 2003 - 2004 Software Resources International S.A

```
const chapi_in * ci = ...;
if (ci->decrypt_data_block) {
    ci->decrypt_data_block(ci, buf, len);
}
```

The example above shows that the CHARON-VAX emulator is not obliged to support the indicated operation. It also shows why the loadable component instance must retain the chapi_in descriptor.

© 2003 - 2004 Software Resources International S.A

1. CHAPI.H

This appendix provides the source code listing of the CHAPI.H file. This file contains the general declarations and definitions necessary for building loadable components.

```
//
// Copyright (C) 1999-2003 Software Resources
International.
// All rights reserved.
//
// The software contained on this media is proprietary
to and embodies
// the confidential technology of Software Resources
International.
// Posession, use, duplication, or dissemination of the
software and
// media is authorized only pursuant to a valid written
license from
// Software Resources International.
//

#if !defined(__CHAPI_H__)
#define __CHAPI_H__

#if defined(__cplusplus)
extern "C" {
#endif // defined(__cplusplus)

#define CHAPI /* nothing is here */

#if !defined(__chapi_in_context_p)
```

```
typedef void * const __chapi_in_context_p;
#endif // !defined(__chapi_in_context_p)

typedef void (CHAPI * __chapi_ast_handler_p)
    (void * arg1, int arg2);

#if !defined(ast_handler)
#define ast_handler __chapi_ast_handler_p
#endif // !defined(ast_handler)

typedef void (CHAPI * __chapi_put_ast_procedure_p)
    (const struct __chapi_in * ci,
     unsigned long delay,
     __chapi_ast_handler_p fun, void * arg1, int arg2);

typedef void (CHAPI * __chapi_sst_handler_p)
    (void * arg1, int arg2);

#if !defined(sst_handler)
#define sst_handler __chapi_sst_handler_p
#endif // !defined(sst_handler)

typedef void (CHAPI * __chapi_put_sst_procedure_p)
    (const struct __chapi_in * ci,
     unsigned long delay,
     __chapi_sst_handler_p fun, void * arg1, int arg2);

typedef unsigned int (CHAPI * __chapi_irq_handler_p)
    (void * arg1, int arg2);

#if !defined(irq_handler)
#define irq_handler __chapi_irq_handler_p
#endif // !defined(irq_handler)
```

© 2003 - 2004 Software Resources International S.A

```
typedef void (CHAPI * __chapi_put_irq_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int vec, unsigned long delay,
     __chapi_irq_handler_p fun, void * arg1, int arg2);

typedef void (CHAPI * __chapi_clear_irq_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int vec);

typedef unsigned int (CHAPI *
__chapi_read_mem_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int addr, unsigned int len,
     char * buf);

typedef unsigned int (CHAPI *
__chapi_write_mem_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int addr, unsigned int len,
     const char * buf);

#if !defined(__chapi_io_space_id_t)
typedef void * __chapi_io_space_id_t;
#endif // !defined(__chapi_io_space_id_t)

typedef __chapi_io_space_id_t (CHAPI *
__chapi_create_io_space_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int addr, unsigned int len);

typedef void (CHAPI *
__chapi_move_io_space_procedure_p)
    (const struct __chapi_in * ci,
     __chapi_io_space_id_t space_id,
```

 © 2003 - 2004 Software Resources International S.A

```
    unsigned int addr, unsigned int len);

typedef void (CHAPI *
__chapi_destroy_io_space_procedure_p)
    (const struct __chapi_in * ci,
     __chapi_io_space_id_t space_id);

typedef void (CHAPI *
__chapi_decrypt_data_block_procedure_p)
    (const struct __chapi_in * ci,
     void * buf, unsigned int len);

typedef void (CHAPI *
__chapi_log_message_procedure_p)
    (const struct __chapi_in * ci,
     const char * buf, unsigned int len);

typedef struct __chapi_in {
    __chapi_in_context_p context;
    unsigned int base_b_address;
    unsigned int base_i_vector;
    __chapi_put_ast_procedure_p put_ast;
    __chapi_put_sst_procedure_p put_sst;
    __chapi_put_irq_procedure_p put_irq;
    __chapi_clear_irq_procedure_p clear_irq;
    __chapi_read_mem_procedure_p read_mem;
    __chapi_write_mem_procedure_p write_mem;
    __chapi_create_io_space_procedure_p
create_io_space;
    __chapi_move_io_space_procedure_p move_io_space;
    __chapi_destroy_io_space_procedure_p
destroy_io_space;
    __chapi_decrypt_data_block_procedure_p
decrypt_data_block;
```

```
    __chapi_log_message_procedure_p log_message;
} chapi_in;

#if !defined(__chapi_out_context_p)
typedef void * const __chapi_out_context_p;
#endif // !defined(__chapi_out_context_p)

typedef void (CHAPI * __chapi_start_procedure_p)
    (const struct __chapi_out * co);

typedef void (CHAPI * __chapi_stop_procedure_p)
    (const struct __chapi_out * co);

typedef void (CHAPI * __chapi_reset_procedure_p)
    (const struct __chapi_out * co);

typedef int (CHAPI * __chapi_read_procedure_p)
    (const struct __chapi_out * co,
     unsigned int addr,
     bool is_byte);

typedef void (CHAPI * __chapi_write_procedure_p)
    (const struct __chapi_out * co,
     unsigned int addr,
     int val,
     bool is_byte);

typedef int (CHAPI *
__chapi_set_configuration_procedure_p)
    (const struct __chapi_out * co,
     const char * parameters);

typedef struct __chapi_out {
    __chapi_out_context_p context;
```

 © 2003 - 2004 Software Resources International S.A

```
    unsigned int b_address_range;
    unsigned int n_of_i_vector;
    unsigned int i_priority;
    __chapi_start_procedure_p start;
    __chapi_stop_procedure_p stop;
    __chapi_reset_procedure_p reset;
    __chapi_read_procedure_p read;
    __chapi_write_procedure_p write;
    __chapi_set_configuration_procedure_p
set_configuration;
} chapi_out;

#if defined(__cplusplus)
} /* extern "C" */;
#endif // defined(__cplusplus)

#endif // !defined(__CHAPI_H__)
```

2. LPV11 implementation

This appendix provides the source code listing of the LPV11.CXX file. This file provides as an example of CHAPI device programming a CHAPI compatible implementation in CHARON-VAX of the LPV11 Qbus printer interface. The VAXprint example listed in the next appendix will send the LPV11 output to the default printer in the Windows host system.

Note that this code is provided 'as is' to illustrate the CHAPI concepts described in this manual. Use of this code is entirely at your own risk and Software Resources International does not warrant that this code would not violate any patents nor that it will function correctly in an operational environment. The CHAPI interface can be changed at any time without prior notice.

A.2 LPV11.CXX source file

```
//
// Copyright (C) 2004 Software Resources
International.
// All rights reserved.
//
// This code is provided on an 'as is' basis.
// Software Resources International SA carries no
responsibility for
// the application of this code in any way or form.
//

// Link project using input Ws2_32.lib library
#include <WinSock2.h>

#include <chapi.h>
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
//
// Default bus address & interrupt vector
// Used if not specified in .cfg file
//

#define DEFAULT_ADDRESS 017777514
#define DEFAULT_VECTOR  0200

//
// Output file extension
//

#define OUTFILE_EXT ".lpv11"

//
// Debug trace routine
//

static void trace(const chapi_in * ci, const char *str)
{
    if(ci != 0 && ci->log_message != 0) {
        ci->log_message(ci, str, strlen(str));
    }
}

//
// This class implements LPV11 device.
// LPV11_INIT routine will create a new instance,
// and a pointer to it will be used as a "context" in
CHAPI_OUT
//

class lpv11 {
private:
    void trace(const char *str) { ::trace(ci, str); }    //
debug trace routine
```

```
   const chapi_in * ci;    // pointer to CHARON-supplied
structure
   char * outfile_name;    // output file/device name

   //
   // LPV11 uses only 1 I/O region and 1 interrupt
vector.
   // If they aren't specified in .cfg file, default values
will be used.
   // So, to avoid additional checking
(specified/unspecified)
   // in all places where we need these values, they are
calculated
   // during device start and stored here.
   //
   unsigned int b_address; // Bus address
   unsigned int i_vector;  // Interrupt vector

   enum {  // LPCS & LPDB bits definitions
      lpcs_error = 0x8000,    // Error condition detected
      lpcs_ready = 0x0080,    // Device ready flag
      lpcs_int   = 0x0040,    // Interrupt enable bit
      lpdb_data  = 0x007f,    // Data mask
   };

   bool int_enabled;   // Interrupt enabled flag
   bool error;         // Error flag

   // Ring buffer.
   // In order to speed up completion of write()
procedure,
   // data written to LPDB register is just stored here.
   // Working thread will do the real job in the
background.
   enum {
      ring_buf_size = 1 << 10,    // Must be a power of 2
   };
   char ring_buf[ring_buf_size];
   unsigned int volatile ring_start;
   unsigned int volatile ring_done;
   unsigned int volatile ring_end;
```

 © 2003 - 2004 Software Resources International S.A

```
   // Device is ready if we have some space to store
data
   bool ready() { return (ring_end + 1 - ring_start) %
ring_buf_size != 0; }

   // When data is written to LPDB, sst will be
requested to wake up
   // working thread <kick_delay> instructions later.
   enum {
      kick_delay = 1000,
   };

   bool sst_pending;   // And we don't want more then 1
sst request at a time.

   // sst callback for kicking working thread
   static void kick_wrk_thread(void * arg1, int arg2);
   void kick_wrk_thread();

   // This function will request interrupt and free ring
buffer
   static void irq_requestor(void * arg1, int arg2);
   void irq_requestor(unsigned int delta);

   // This function is used to cleanup ring buffer
   static void set_ring_start(void * arg1, int arg2);
   void set_ring_start(unsigned int new_ring_start);

   enum {
      wrk_event = 0,      // Working thread is waiting for
this event to process data
      io_event,           // Auxiliary event used by the
working thread
      cancel_event,       // Auxiliary event used to inform
working thread that it should cleanup I/O
      termination_event,  // This event is used to
terminate working thread
      total_events        // Events count
   };
```

```
   HANDLE events[total_events];    // Event handles
   HANDLE wrk_thread;              // Working thread
handle

   HANDLE outfile;     // Handle to the output file
   OVERLAPPED ovl;     // And overlapped structure for
it

   // Working thread procedure
   static DWORD WINAPI wrk_thread_proc(LPVOID
lpParameter);
   DWORD wrk_thread_proc();

public:
   // Constructor/destructor
   lpv11(const chapi_in * _ci, const char *
instance_name);
   ~lpv11();

   // Tells us if the instance was created/initialized
properly
   bool valid();

   // Callbacks provided to CHARON in CHAPI_OUT
   void start();
   void stop();
   void reset();
   int read(unsigned int addr, bool is_byte);
   void write(unsigned int addr, int val, bool is_byte);
   int set_configuration(const char * parameters);

protected:
   // start VAXPrint using LPV11 command parameters
   bool bStartApp;
   char m_AppCmd[1024];

   bool UseTCPIP;
   bool bReset;

   STARTUPINFO si;
   PROCESS_INFORMATION pi;
```

 © 2003 - 2004 Software Resources International S.A

```
    SOCKET sock;
    SOCKET connection_sock;

    char Message[256];
    void GetSystemErrorMessage( DWORD dwErr );

protected:
    // TCPIP connection parameters
    char hostname[256];
    long portnum;
};

// ----------------------------------------------------------------
void lpv11::GetSystemErrorMessage( DWORD dwErr )
{
    LPTSTR lpszTemp = NULL;
    DWORD dwRet = FormatMessage(
                          FORMAT_MESSAGE_ALLOCATE_BUF
FER
                          |
FORMAT_MESSAGE_FROM_SYSTEM
                          |
FORMAT_MESSAGE_ARGUMENT_ARRAY,
                          NULL,
                          dwErr,
                          LANG_NEUTRAL,
                          (LPTSTR)&lpszTemp,
                          0,
                          NULL );
    if( !dwRet ) {
        trace( "Unable to detect the problem" );
        sprintf( Message, "0x%X\n", dwErr );
    }
    else {
        //remove cr and newline character
        lpszTemp[strlen(lpszTemp)-2] = '\0';
        sprintf( Message, "Communication error: %s (0x
%X)\n", lpszTemp, dwErr );
    }
```

```
   if( lpszTemp != NULL ) {
      LocalFree((HLOCAL) lpszTemp);
   }

   trace( Message );

   return;
}

// -----------------------------------------------------------------
void lpv11::kick_wrk_thread(void * arg1, int arg2)
{
   ((lpv11 *)arg1)->sst_pending = false;   // clear
indicator
   ((lpv11 *)arg1)->kick_wrk_thread();     // and call real
procedure
}

// -----------------------------------------------------------------
void lpv11::kick_wrk_thread()
{
   SetEvent(events[wrk_event]);    // wake up working
thread
}

// -----------------------------------------------------------------
void lpv11::irq_requestor(void * arg1, int arg2)
{
   ((lpv11 *)arg1)->irq_requestor(arg2);   // call real
procedure
}

// -----------------------------------------------------------------
void lpv11::irq_requestor(unsigned int delta)
{
   // Free some space in the ring buffer.
   // This will make device ready again.
   ring_start += delta;

   // And request interrupt, if enabled.
   if(int_enabled)
```

 © 2003 - 2004 Software Resources International S.A

```
      ci->put_irq(ci, ci->base_i_vector, 0, 0, this, 0);
}

// ---------------------------------------------------------------
void lpv11::set_ring_start(void * arg1, int arg2)
{
   ((lpv11 *)arg1)->set_ring_start(arg2);  // call real
procedure
}

// ---------------------------------------------------------------
void lpv11::set_ring_start(unsigned int new_ring_start)
{
   ring_start = new_ring_start;
}

// ---------------------------------------------------------------
DWORD WINAPI lpv11::wrk_thread_proc(LPVOID
lpParameter)
{
   return ((lpv11 *)lpParameter)-
>wrk_thread_proc();   // call real procedure
}

// ---------------------------------------------------------------
DWORD lpv11::wrk_thread_proc()
{
   if( UseTCPIP ) {
      int rc = 0;
      int so_true = 1;
      WSADATA info;

      if (WSAStartup(MAKEWORD(2,0), &info)) {
         trace( "Socket: WSAStartup error" );
         return( FALSE );
      }

      sock = ::WSASocket( AF_INET, SOCK_STREAM, 0,
0, 0,
                                WSA_FLAG_OVERLAPPED );
      if( sock == INVALID_SOCKET ) {
```

```
        rc = ::WSAGetLastError();
        trace( "Socket: Can't create socket" );
        return( FALSE );
    }

    if (::setsockopt(sock, SOL_SOCKET,
SO_REUSEADDR,
                (const char *)&so_true, sizeof(so_true))
== SOCKET_ERROR ) {
        rc = ::WSAGetLastError();
        closesocket( sock );
        trace( "Socket: Can't setsockopt
SO_REUSEADDR" );
        return( rc );
    }

    sockaddr_in sa;
    memset( &sa, 0, sizeof(sa) );

    sa.sin_family = PF_INET;
    sa.sin_port = htons( (short)portnum );

    // bind the socket to the internet address
    if( bind(sock,(sockaddr
*)&sa,sizeof(sockaddr_in))==SOCKET_ERROR ) {
        rc = ::WSAGetLastError();
        closesocket( sock );
        trace( "Socket: Can't bind" );
        return( FALSE );
    }

    if( listen(sock, SOMAXCONN) == SOCKET_ERROR )
{
        rc = ::WSAGetLastError();
        closesocket( sock );
        trace( "Socket: Can't listen" );
        return( FALSE );
    }

    // Start VAXPrint
    ZeroMemory( &si, sizeof(si) );
```

 © 2003 - 2004 Software Resources International S.A

```
      si.cb = sizeof(si);
      ZeroMemory( &pi, sizeof(pi) );

      if( bStartApp ) {
         if( !CreateProcess( NULL, // No module name
(use command line).
                              m_AppCmd,  // Command line.
                              NULL, // Process handle not
inheritable.
                              NULL, // Thread handle not
inheritable.
                              FALSE,// Set handle inheritance to
FALSE.
                              0,    // No creation flags.
                              NULL, // Use parent's environment
block.
                              NULL, // Use parent's starting
directory.
                              &si,
                              &pi ) ) {
            trace( "VAXPrint start failed" );
         }
      }

      connection_sock = accept( sock, NULL, NULL );
      if( connection_sock == INVALID_SOCKET ) {
         rc = ::WSAGetLastError();
         if( rc != WSAEWOULDBLOCK ) {
            closesocket( sock );
            trace( "Socket: Can't accept" );
            return( FALSE );
         }
      }
   }

bool io_pending = false;    // We don't want more then
1 I/O operation at a time
DWORD written;             // Number of bytes written by
the last operation

   for(;;) {
```

```
    DWORD signaled_object =
      WaitForMultipleObjects(total_events, events,
FALSE, INFINITE);
    switch(signaled_object - WAIT_OBJECT_0) {
     case cancel_event:
      CancelIo(outfile);           // cancel pending I/
O
      // cleanup ring buffer
      ring_done = ring_end;
      ci->put_ast(ci, 0, set_ring_start, this, ring_end);
      continue;
     case io_event:   // I/O completed
      ovl.Offset += written;
      io_pending = false;
      // fall through
     case wrk_event:  // we have something to work
on...
      if(io_pending)
        continue;

      unsigned int num_to_print = ring_end -
ring_done;
      if(! num_to_print)
        continue;

      // Allocate contiguous buffer to use in WriteFile
      char * outbuf = new char [num_to_print];
      // Copy data there... this may be optimized...
      for(unsigned int i = 0; i < num_to_print; i++) {
        outbuf[i] = ring_buf[(ring_done + i) %
ring_buf_size];
      }

      if( UseTCPIP ) {
        written = send( connection_sock, outbuf,
num_to_print, 0 );
        if( written != num_to_print ) {
          DWORD lasterror = GetLastError();
          GetSystemErrorMessage( lasterror );
```

 © 2003 - 2004 Software Resources International S.A

```
                connection_sock = accept( sock, NULL,
NULL );
                if( connection_sock == INVALID_SOCKET )
{
                   trace( "Socket: Can't listen" );
                   error = true;
                }
             }
          }
          else {
             if(! WriteFile(outfile, outbuf, num_to_print,
&written, &ovl)) {
                if(GetLastError() != ERROR_IO_PENDING) {
                   trace("WriteFile failed");
                   error = true;   // IRQ will be requested a
few lines later anyway
                } else {
                   io_pending = true;
                }
             }
          }

          delete[] outbuf;

          // Advance ring_done. This will NOT change
device status to READY.
          ring_done += num_to_print;
          // Device will be made READY in this AST
          ci->put_ast(ci, 0, irq_requestor, this,
num_to_print);
          continue;
       }
       break;
    }

    return 0;
}

// ---------------------------------------------------------------
lpv11::lpv11(const chapi_in * _ci,
          const char * instance_name)
```

```
{
   UseTCPIP = false;
   bReset = false;
   bStartApp = false;
   strcpy( &m_AppCmd[0], "" );

   ci = _ci;
   int_enabled = false;
   error = false;
   sst_pending = false;

   ring_start = ring_done = ring_end = 0;

   for(int i = 0; i < total_events; i++) {
      events[i] = CreateEvent(NULL, FALSE, FALSE,
NULL);
      if(events[i] == NULL) {
         trace("cannot create event");
      }
   }

   outfile = INVALID_HANDLE_VALUE;

   memset(&ovl, 0, sizeof(ovl));
   ovl.hEvent = events[io_event];

   outfile_name = new char[strlen(instance_name) +
sizeof(OUTFILE_EXT)];
   if(outfile_name != NULL) {
      strcpy(outfile_name, instance_name);
   }

   wrk_thread = CreateThread(NULL, 0,
wrk_thread_proc, this, 0, NULL );
   if(wrk_thread == NULL) {
      trace("cannot create thread");
   }
}

// ----------------------------------------------------------------
lpv11::~lpv11()
```

 © 2003 - 2004 Software Resources International S.A

```
{
   if(wrk_thread != NULL) {
      SetEvent(events[termination_event]);
      WaitForSingleObject(wrk_thread, INFINITE);
      wrk_thread = NULL;
   }

   for(int i = total_events - 1; i >= 0; i--) {
      if(events[i] != NULL) {
         CloseHandle(events[i]);
         events[i] = NULL;
      }
   }
}

// ---------------------------------------------------------------
bool lpv11::valid()
{
   for(int i = 0; i < total_events; i++) {
      if(events[i] == NULL)
         return false;
   }

   if(wrk_thread == NULL) {
      return false;
   }

   if(outfile_name == NULL) {
      return false;
   }

   return true;
}

// ---------------------------------------------------------------
void lpv11::start()
{
   // we have not so much to do here...
   b_address = ci->base_b_address;
   i_vector = ci->base_i_vector;
   int_enabled = false;
```

```
   error = false;

   if( !UseTCPIP ) {
      if(outfile_name != NULL) {
         outfile = CreateFile(outfile_name,
GENERIC_WRITE,
                FILE_SHARE_READ, NULL,  OPEN_ALWAYS,
                FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_OVERLAPPED, NULL);
         if(outfile != INVALID_HANDLE_VALUE) {
            ovl.Offset = GetFileSize(outfile, NULL);
         } else {
            trace("cannot open file");
         }
      }
   }
}

// -------------------------------------------------------------------
void lpv11::stop()
{
   if( bReset ) {
      bReset = false;
   }
   else {
      if( UseTCPIP ) {
         if( pi.hProcess != INVALID_HANDLE_VALUE ) {
            UINT uExitCode = 0;
            BOOL res = TerminateProcess(  pi.hProcess,
uExitCode );
            if( res == 0 ) {
               DWORD l_err = GetLastError();
            }

            CloseHandle( pi.hProcess );
            if( pi.hThread != INVALID_HANDLE_VALUE ) {
               CloseHandle( pi.hThread );
            }
         }

         WSACleanup();
```

 © 2003 - 2004 Software Resources International S.A

```
        }
        else {
            if(outfile_name != NULL) {
                DeleteFile( outfile_name );
                delete[] outfile_name;
                outfile_name = NULL;
            }

            if(outfile != INVALID_HANDLE_VALUE) {
                CloseHandle(outfile);
                outfile = INVALID_HANDLE_VALUE;
            }
        }
    }

    SetEvent(events[cancel_event]); // request working
thread to cancel I/O

    // And close the output
    if( !UseTCPIP ) {
        if(outfile != INVALID_HANDLE_VALUE) {
            CloseHandle(outfile);
            outfile = INVALID_HANDLE_VALUE;
        }
    }
}

// -----------------------------------------------------------------
void lpv11::reset()
{
    bReset = true;

    stop();
    start();
}

// -----------------------------------------------------------------
int lpv11::read(unsigned int addr, bool is_byte)
{
    if((addr - b_address) == 0)
        return (ready() ? lpcs_ready : 0)
```

```
            | (int_enabled ? lpcs_int : 0)
            | ((error && ! is_byte) ? lpcs_error : 0);

   // return 0 if LPDB is read
   return 0;
}

// -------------------------------------------------------------
void lpv11::write(unsigned int addr, int val, bool
is_byte)
{
   // Do nothing if attempting to write to high-order
byte of LPCS/LPDB
   switch(addr - b_address) {
     case 0:   // LPCS: only bother about interrupt
enable/disable
       if(val & lpcs_int) {
          int_enabled = true;
       } else {
          int_enabled = false;
       }
       break;
     case 2:   // LPDB: use lower 7 bits only. If not
ready, do nothing.
       if(ready()) {
          ring_buf[ring_end++ % ring_buf_size] = val &
lpdb_data;
          if(!ready()) {  // Ring buffer is full => kicj
working thread immediately
             kick_wrk_thread();
    } else if(! sst_pending) {  // Else, kick it a bit later...
if not already directed so
             sst_pending = true;
             ci->put_sst(ci, kick_delay, kick_wrk_thread,
this, 0);
          }
       }
       break;
   }
}
```

 © 2003 - 2004 Software Resources International S.A

```
// -----------------------------------------------------------------
int lpv11::set_configuration(const char * parameters)
{
   // Parse command line, obtain required parameters

   char *opt_host = "host=";
   char *opt_port = "port=";
   char *opt_app  = "application='";

   UseTCPIP = false;

   char *ptr = NULL;
   strlwr( (char *)parameters );

   if( (ptr=strstr(parameters, opt_port)) != NULL ) {
      UseTCPIP = true;

      strcpy( hostname, "localhost" );
      portnum = 0;

      ptr += strlen( opt_port );
      portnum = atoi( ptr );
   }

   if( (ptr=strstr(parameters, opt_host)) != NULL ) {
      ptr += strlen( opt_host );
      int i = 0;
      while( *(ptr+i) && isalnum((int)*(ptr+i)) ) {
         hostname[i] = *(ptr+i);
         i += 1;
      }
      hostname[i] = '\0';
   }

   if( (ptr=strstr(parameters, opt_app)) != NULL ) {
      UseTCPIP = true;

      ptr += strlen( opt_app );

      int j = 0;
      while( *ptr && (*ptr != '\'') ) {
```

© 2003 - 2004 Software Resources International S.A

```
            m_AppCmd[j] = *ptr;
            j ++;
            ptr ++;
        }
        m_AppCmd[j] = '\0';

        if( ptr ) {
            ptr ++;
        }

        if( strlen(m_AppCmd) ) {
            bStartApp = true;
        }
    }

    return 0;
}

// ---------------------------------------------------------------------
//
// A number of wrappers to be passed to CHARON
//
#define CALL(co, func, arglist) ((lpv11 *)(co-
>context))->func arglist

static void CHAPI lpv11_start(const struct __chapi_out *
co)
{ CALL(co, start, ()); }

static void CHAPI lpv11_stop(const struct __chapi_out *
co)
{ CALL(co, stop, ()); }

static void CHAPI lpv11_reset(const struct __chapi_out
* co)
{ CALL(co, reset, ()); }

static int CHAPI lpv11_read(const struct __chapi_out *
co,
                    unsigned int addr, bool is_byte)
{ return CALL(co, read, (addr, is_byte)); }
```

 © 2003 - 2004 Software Resources International S.A

```
static void CHAPI lpv11_write(const struct __chapi_out
* co,
                    unsigned int addr, int val, bool is_byte)
{ CALL(co, write, (addr, val, is_byte)); }

static int CHAPI lpv11_set_configuration(const struct
__chapi_out * co,
                              const char * parameters)
{ return CALL(co, set_configuration, (parameters)); }

// ---------------------------------------------------------------
//
// Initialization routine
//
CHAPI_INIT(LPV11)
   (const chapi_in * ci, chapi_out * co, const char *
instance_name)
{
   if(ci == NULL) {
      trace(ci, "CHARON didn't provide us CHAPI_IN.
Giving up");
      return 0;
   }

   if((ci->put_irq == 0)
   || (ci->put_ast == 0)
   || (ci->put_sst == 0))
   {
      trace(ci, "CHARON didn't provide us one of the
following procedures:\n"
            "\tput_irq\n"
            "\tput_ast\n"
            "\tput_sst"
            "Giving up"
      );
      return 0;
   }

   lpv11 *dev = new lpv11(ci, instance_name);
   if((dev == NULL)
```

```
    || (! dev->valid()))
    {
        trace(ci, "cannot create LPV11 instance. Giving
up");
        if(dev != NULL)
            delete dev;
        return 0;
    }

    co->context = dev;
    co->base_b_address = DEFAULT_ADDRESS;
    co->b_address_range = 4;
    co->base_i_vector = DEFAULT_VECTOR;
    co->n_of_i_vector = 1;
    co->i_priority = 4;
    co->start = lpv11_start;
    co->stop = lpv11_stop;
    co->reset = lpv11_reset;
    co->read = lpv11_read;
    co->write = lpv11_write;
    co->set_configuration = lpv11_set_configuration;

    return dev;
}
```

 © 2003 - 2004 Software Resources International S.A

1. VAXPrint application

This appendix provides the source code listing of the CHARON-VAX VAXPrint application. For the purpose of demonstrating the CHAPI LPV11 implementation, this application is started from LPV11 dll (as a parameter in configuration file), connected to the LPV11 via a TCP/IP socket (host name and port number are configuration file parameters).

VAXprint sends the characters received from the socket to the Windows default printer. The user can specify the font face name and the font size as parameters in the CHARON-VAX configuration file.

A.3 VAXPRINT.CPP source file

```
//
// Copyright (C) 2004 Software Resources International
SA
//
// This code is provided on an 'as is' basis.
// Software Resources International SA carries no
responsibility for
// the application of this code in any way or form.
//

// -------------------------------------------------------------------
#include "StdAfx.h"
#include "MainFrm.h"
#include "VAXPRINT.h"
#include "WinSock.h"

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

```
#include <iostream> // basic_string
#include <tchar.h>


// ----------------------------------------------------------------
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// ----------------------------------------------------------------
#define countof(x)  (sizeof(x)/sizeof(x[0]))
#define DATA_BUFSIZE 4096

/////////////////////////////////////////////////////////////////////////////
// CVAXPRINTApp

BEGIN_MESSAGE_MAP(CVAXPRINTApp, CWinApp)
	//{{AFX_MSG_MAP(CVAXPRINTApp)
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVAXPRINTApp construction

// ----------------------------------------------------------------
CVAXPRINTApp::CVAXPRINTApp()
{
   // No line is in buffer
   buffer_size = 0;

   //Default settings for printer
   strcpy( m_FontName, "" );
   FontSize = DEFAULT_FONT_SIZE;
   offset = FontSize;
   MaxCharactersInLine = -1;
	TabSize=8;
```

 © 2003 - 2004 Software Resources International S.A

```
    //No Document is open
    DocumentIsOpen = false;
    reading_status = 0;

    // Initialize the critical section.
    InitializeCriticalSection(&GlobalCriticalSection);

    //Initialize Icon
    memset(&Icon, 0 , sizeof(NOTIFYICONDATA));
    Icon.cbSize = sizeof(NOTIFYICONDATA);
}

// -----------------------------------------------------------------
CVAXPRINTApp::~CVAXPRINTApp()
{
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CVAXPRINTApp object

CVAXPRINTApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CVAXPRINTApp initialization

// -----------------------------------------------------------------
BOOL CVAXPRINTApp::InitInstance()
{
    AfxEnableControlContainer();

#ifdef _AFXDLL
    Enable3dControls(); // MFC in a shared DLL
#else
    Enable3dControlsStatic(); // MFC is statically linked
#endif

    // Get Socket Name, name of serial line and
application name
    char *cmdLine = GetCommandLine();
    strlwr( cmdLine );
```

```
strcpy( hostname, "localhost" );
portnum = 0;

char *opt_host = "host=";
char *opt_port = "port=";
char *opt_font = "font=";
char *opt_fontsize = "fontsize=";
char *ptr = NULL;

// check for host parameter
if( (ptr=strstr(cmdLine, opt_host)) != NULL ) {
   ptr += strlen( opt_host );
   int i = 0;
   while( *(ptr+i) && isalnum((int)*(ptr+i)) ) {
      hostname[i] = *(ptr+i);
      i += 1;
   }
   hostname[i] = '\0';
}

// check for port parameter
if( (ptr=strstr(cmdLine, opt_port)) != NULL ) {
   ptr += strlen( opt_port );
   portnum = atoi( ptr );
}

if( portnum <= 0 ) {
   return( FALSE );
}

// check for font parameter
if( (ptr=strstr(cmdLine, opt_font)) != NULL ) {
   ptr += strlen( opt_font );
   ptr += 1; // '\'
   int i = 0;
   while( *(ptr+i) && (*(ptr+i) != '\\') ) {
      m_FontName[i] = *(ptr+i);
      i += 1;
   }
   m_FontName[i] = '\0';
```


© 2003 - 2004 Software Resources International S.A

```
      if( strlen(m_FontName) > 30 ) {
         // MSDN: The length of typeface name of the
font
         //      must not exceed 30 characters
         m_FontName[30] = '\0';
      }
   }

   // check for fontsize parameter
   if( (ptr=strstr(cmdLine, opt_fontsize)) != NULL ) {
      ptr += strlen( opt_fontsize );
      FontSize = atoi( ptr );
      if( (FontSize <=0) || (FontSize >= 100) ) {
         FontSize = DEFAULT_FONT_SIZE;
      }
      else {
         FontSize = FontSize * 10;
      }
      offset = FontSize;
   }


   DWORD lasterror = 0;

   lasterror = m_LPV11Socket.Initialize();
   if( lasterror ) {
      GetSystemErrorMessage( lasterror );
   }

   // try to connect...
   do {
      lasterror =
m_LPV11Socket.ConnectSocket( hostname, portnum );
      if( lasterror && (lasterror !=
WSAECONNREFUSED) ) {
         GetSystemErrorMessage( lasterror );
      }
      else if(lasterror == WSAECONNREFUSED ) {
         Sleep( 200 );
      }
   } while( lasterror == WSAECONNREFUSED );
```

© 2003 - 2004 Software Resources International S.A

```
   // Load Icons
   hDefIcon = AfxGetApp()-
>LoadIcon(IDI_MAINFRAMEALTER);
   hWorkIcon = AfxGetApp()-
>LoadIcon(IDR_MAINFRAME);

   // Set Tips
   sprintf( DefaultMessage,"VAXprint: Connected to
CHARON-VAX" );
   sprintf( WorkingMessage,"VAXprint: Receiving
information from CHARON-VAX..." );

   // Initialize printer
   InitializePrinter();

   // Start Mainframe based on CDialog class
   CMainFrame* pMainFrame = new CMainFrame;
   if(!pMainFrame->Create(IDD_MAINDIALOG) ) {
      return FALSE;
   }
   pMainFrame->ShowWindow(SW_HIDE);
   m_pMainWnd = pMainFrame;

   // Start Printing Thread
   CWinThread* pPrintingThread =
AfxBeginThread(printing_thread, this);

   return TRUE;
}

// -----------------------------------------------------------------
int CVAXPRINTApp::ExitInstance()
{
   // Remove Icon from the tray
   Shell_NotifyIcon( NIM_DELETE, &Icon );

   if( hDefIcon ) {
      DestroyIcon( hDefIcon );
   }
```

 © 2003 - 2004 Software Resources International S.A

```
   if( hWorkIcon ) {
      DestroyIcon( hWorkIcon );
   }

   return CWinApp::ExitInstance();
}

//-----------------------------------------------------
// Read Information
//-----------------------------------------------------
UINT CVAXPRINTApp::reading_thread(LPVOID pParam)
{
   return ((CVAXPRINTApp*)pParam)-
>reading_thread();
}

// -------------------------------------------------------------------
UINT CVAXPRINTApp::reading_thread()
{
   while( true ) {
      // Wait till the buffer is sent to printer and only
then receive new line
      WaitForSingleObject( GetNextLine, INFINITE );

      // Read pipe and get status of reading
      reading_status = read_buffer();

      // If connection to CHARON-VAX is broken exit the
thread
      if( reading_status == 1 ) {
         // Signal that the new line is received
         SetEvent( LineIsReceived );
      }
      else if( reading_status == 0 ) {
         // Signal that the new line is received
         SetEvent( NoDataReceived );
      }
      else {
         SetEvent( BadConnection );
         return 0;
```

```
        }
    } // while
}

// ------------------------------------------------------------------
int CVAXPRINTApp::read_buffer()
{
    char          character;
    unsigned long received;

    // Intitialization
    __try {
        // Enter critical section
        EnterCriticalSection( &GlobalCriticalSection );
        buffer_size = 0;
        memset( (void *)&buffer[0], 0,
MAX_BUFFER_SIZE );
    }
    __finally {
        // Release ownership of the critical section.
        LeaveCriticalSection( &GlobalCriticalSection );
    }

    bool ReadCharacter = false;

    while (true) {

        received = 0;
        received = m_LPV11Socket.ReceiveByte();

        if( received < 1 ) {
            ReadCharacter = false;
            return( received );
        }
        else {
            ReadCharacter = true;
        }

        // Request ownership of the critical section.
        __try {
            EnterCriticalSection( &GlobalCriticalSection );
```

 © 2003 - 2004 Software Resources International S.A

```
        if( ReadCharacter ) {

           character = m_LPV11Socket.sock_buf[0];

           switch (character) {
           case '\f': // page terminator (form feed)
received
           case '\n': // line terminator (line feed)
received
              if( !buffer_size ) {
                 buffer[buffer_size++] = ' ';
              }
              buffer[buffer_size] = '\0';
              return( 1 );

           case '\r': // return of carret is ignored
              continue;

           default: // copy the character to the buffer
              if( character == 0x20 ) {
                 character = ' ';
              }

              buffer[buffer_size++] = character;

              // Check whether the line is too long to fit
              // in one line on paper
              if( buffer_size >= MaxCharactersInLine ) {
                 buffer[buffer_size] = '\0';
                 return( 1 );
              }
           }
        }
     }
     __finally {
        // Release ownership of the critical section.
        LeaveCriticalSection( &GlobalCriticalSection );
     }
  } // while
```

© 2003 - 2004 Software Resources International S.A

```
   return( 0 );
}

//-------------------------------------------------
// Print Information
//-------------------------------------------------

// Printing Thread function. It reads buffer from
CHARON-VAX pipe
// and sends it to printer.
// ------------------------------------------------------------------
UINT CVAXPRINTApp::printing_thread(LPVOID pParam)
{
   return ((CVAXPRINTApp*)pParam)-
>printing_thread();
}

// ------------------------------------------------------------------
UINT CVAXPRINTApp::printing_thread()
{

   // Create Events for data exchange between threads
   BadConnection = CreateEvent( NULL, FALSE, FALSE,
NULL );
   NoDataReceived = CreateEvent( NULL, FALSE,
FALSE, NULL );
   LineIsReceived = CreateEvent( NULL, FALSE, FALSE,
NULL );
   GetNextLine      = CreateEvent( NULL, FALSE, TRUE,
NULL );

   m_events[NoDataReceivedEvent] =
NoDataReceived;
   m_events[BadConnectionEvent]  = BadConnection;
   m_events[LineIsReceivedEvent] = LineIsReceived;

   //Start Reading Thread
   CWinThread* pReadingThread =
AfxBeginThread(reading_thread, this);
```

 © 2003 - 2004 Software Resources International S.A

```
   // Wait till some information is received from
CHARON-VAX
   // or timeout occurs
   while (true) {

      DWORD signaled_object =
WaitForMultipleObjects( TotalEvents, m_events,
FALSE, 5000 );

      switch( signaled_object - WAIT_OBJECT_0 ) {
      case BadConnectionEvent:
         if( DocumentIsOpen ) {
            EndDocumentPrinting();
         }

         // Exit
                  AfxGetMainWnd()-
>PostMessage(WM_QUIT);
                  return( 0 );

      case LineIsReceivedEvent:
         // Create new document if the document is not
created
         if( !DocumentIsOpen ) {
            if( buffer_size ) {
               if( !StartDocumentPrinting() ) {
                  return( 0 );
               }
            }
         }

         // Do actual printing
         if( buffer_size ) {
                     print_buffer();
            buffer_size = 0;
         }

         // Get next line
         SetEvent( GetNextLine );
         break;
```

```
    case NoDataReceivedEvent:
        case WAIT_TIMEOUT: // Reading thread keeps
on reading
      // Print last line if the document is open and
then
      // close the open document
      if( DocumentIsOpen ) {
        __try {
          EnterCriticalSection( &GlobalCriticalSectio
n );
          if( buffer_size ) {
            print_buffer();
            buffer_size = 0;
          }
        }
        __finally {
          LeaveCriticalSection( &GlobalCriticalSecti
on );
        }
        EndDocumentPrinting();
      }
      else {
        if( buffer_size ) {
          if( !StartDocumentPrinting() ) {
            return 0;
          }
        }

        if( buffer_size ) {
          print_buffer();
          buffer_size = 0;
        }
      }
      SetEvent( GetNextLine ); // Get next line
            break;

    default:
      // some error occured
      return( 0 );
    }
  } // while
```

 © 2003 - 2004 Software Resources International S.A

```
   return 1;
}

// -----------------------------------------------------------------
print_buffer()
int CVAXPRINTApp::print_buffer()
{
   // Set tabulation parameters
        TEXTMETRIC tm;
   dcPrinter.GetTextMetrics( &tm );

        int TabStopPosition[2];
        TabStopPosition[0] = TabSize *
tm.tmAveCharWidth;

   // Do printing
   dcPrinter.TabbedTextOut( 0, offset, _T(buffer),
strlen(buffer), 1, TabStopPosition, 0 );

   // Correct offset and continue on next page if
required
   offset+=FontSize;
   if( offset > (VRes-FontSize) ) {
      offset = FontSize;
      dcPrinter.EndPage();
      dcPrinter.StartPage();
   }

   return true;
} // print_buffer()

//--------------------------------------------------
// Initialization & Preparation
//--------------------------------------------------
int CVAXPRINTApp::InitializePrinter()
{
   // get the default printer
   CPrintDialog dlg( FALSE );
   dlg.GetDefaults();
```

© 2003 - 2004 Software Resources International S.A

```
   // is a default printer set up?
   hdcPrinter = dlg.GetPrinterDC();
   if( hdcPrinter == NULL ) {
#if DEBUG
      AfxMessageBox( "Cannot find default printer!" );
#endif
      return( false );
   }

   // initialize docinfo
   memset(&docinfo, 0, sizeof(docinfo));
   docinfo.cbSize = sizeof(docinfo);
   docinfo.lpszDocName = _T("CHARON-VAX output");

   // Calculate maximum character number in one line
   // attach DC to the default printer
   dcPrinter.Attach(hdcPrinter);

   // get height and width of the page to print
   HRes = dcPrinter.GetDeviceCaps(HORZRES);
   VRes = dcPrinter.GetDeviceCaps(VERTRES);

   // Calculate maximum character number in one line
   TEXTMETRIC tm;
   dcPrinter.GetTextMetrics(&tm);
   if( tm.tmAveCharWidth ) {
      MaxCharactersInLine = HRes /
tm.tmAveCharWidth;
   }
   else {
      MaxCharactersInLine = 40;
   }

   // detach DC
   dcPrinter.Detach();
   // Calculate maximum character number in one line

   return( true );
}

// ---------------------------------------------------------------
```

 © 2003 - 2004 Software Resources International S.A

```
int CVAXPRINTApp::StartDocumentPrinting()
{
   // attach DC to the default printer
   dcPrinter.Attach(hdcPrinter); // should be done only
once

   // get height and width of the page to print
   HRes = dcPrinter.GetDeviceCaps(HORZRES);
   VRes = dcPrinter.GetDeviceCaps(VERTRES);

   // if it fails, complain and exit gracefully
   if( dcPrinter.StartDoc(&docinfo) < 0 ) {
#if DEBUG
      AfxMessageBox( "Printer wouldn't initalize" );
#endif
      return false;
   }
   else {
     // start a page
     if( dcPrinter.StartPage() < 0 ) {
#if DEBUG
        AfxMessageBox( "Could not start page" );
#endif
        dcPrinter.AbortDoc();
        return false;
     }

     // Choose some font
     char *font_face = &m_FontName[0];
     if( !strlen(font_face) ) {
        font_face = "Arial";
     }
     font.CreatePointFont( FontSize, font_face,
&dcPrinter );
     pOldFont = dcPrinter.SelectObject( &font );

     // Calculate maximum character number in one
line
     TEXTMETRIC tm;
     dcPrinter.GetTextMetrics( &tm );
     if( tm.tmAveCharWidth ) {
```

```
        MaxCharactersInLine = HRes /
tm.tmAveCharWidth;
     }
     else {
        MaxCharactersInLine = -1;
     }

     // Reset Offset
     offset = FontSize;
   }

   // Set Printing Icon
   Icon.hIcon = hWorkIcon;
   _tcsncpy( Icon.szTip, WorkingMessage,
countof(Icon.szTip) );
   Shell_NotifyIcon( NIM_MODIFY , &Icon );

   // Set the flag that document is open
   DocumentIsOpen = true;
   return true;
}

// -----------------------------------------------------------------
int CVAXPRINTApp::EndDocumentPrinting()
{
   DocumentIsOpen = false;

   // Do last operations of document printing
   dcPrinter.EndPage();
   dcPrinter.EndDoc();
   dcPrinter.SelectObject(pOldFont);

   // Delete font and detach DC
   font.DeleteObject();
   dcPrinter.Detach();

   // Set Default Icon
   Icon.hIcon = hDefIcon;
   _tcsncpy(Icon.szTip, DefaultMessage,
countof(Icon.szTip));
   Shell_NotifyIcon( NIM_MODIFY , &Icon );
```

 © 2003 - 2004 Software Resources International S.A

```
   return true;
}

// ----------------------------------------------------------------
void
CVAXPRINTApp::GetSystemErrorMessage( DWORD
dwErr )
{
   char buff[256];

   LPTSTR lpszTemp = NULL;
   DWORD dwRet =
FormatMessage( FORMAT_MESSAGE_ALLOCATE_BUFFE
R
                        |
FORMAT_MESSAGE_FROM_SYSTEM
                        |
FORMAT_MESSAGE_ARGUMENT_ARRAY,
                        NULL,
                        dwErr,
                        LANG_NEUTRAL,
                        (LPTSTR)&lpszTemp,
                        0,
                        NULL );
   if( !dwRet ) {
      sprintf( buff, _TEXT("0x%X"), dwErr );
   }
   else {
      // remove cr and newline character
      lpszTemp[_tcslen(lpszTemp)-2] = TEXT('\0');
      sprintf( buff, "Communication error: %s (0x%X)",
lpszTemp, dwErr );
   }

   if( lpszTemp != NULL ) {
      LocalFree((HLOCAL) lpszTemp);
   }

#if DEBUG
   AfxMessageBox( out, MB_ICONSTOP );
```

```
#endif

   return;
}
```


© 2003 - 2004 Software Resources International S.A

A.4 VAXPRINT.H source file

```
//
// Copyright (C) 2004 Software Resources International
SA
//
// This code is provided on an 'as is' basis.
// Software Resources International SA carries no
responsibility for
// the application of this code in any way or form.
//

#ifndef __VAXPRINT_H__
#define __VAXPRINT_H__

// ------------------------------------------------------------
#include "WinSock.h"

// ------------------------------------------------------------
#if !defined _VAXPRINT
#define _VAXPRINT

#if !
defined(AFX_VAXPRINT_H__54DCE038_9078_11D4_A11
E_0080C853BDF9__INCLUDED_)
#define
AFX_VAXPRINT_H__54DCE038_9078_11D4_A11E_0080C
853BDF9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this
file for PCH
#endif

#include "resource.h"		// main symbols
```

```
//////////////////////////////////////////////////////////////////////////////
// CVAXPRINTApp:

// The buffer size id fixed. It's equal to 2048 symbols,
#define MAX_BUFFER_SIZE    2048

// Font size is fixed as well
#define DEFAULT_FONT_SIZE  100

// ----------------------------------------------------------------
// Id for event used
enum {
   NoDataReceivedEvent = 0,
        LineIsReceivedEvent = 1,
   BadConnectionEvent  = 2,
   TotalEvents        = 3
};

// ----------------------------------------------------------------
class CVAXPRINTApp : public CWinApp
{
public:
   // Constructors and destructors
   CVAXPRINTApp();
   ~CVAXPRINTApp();

   // Printing Thread definition
   static UINT printing_thread(LPVOID pParam);
   UINT   printing_thread();

   // Reading Thread definition
   static UINT reading_thread(LPVOID pParam);
   UINT   reading_thread();

   // Auxilliary functions
   int read_buffer();
   int print_buffer();
   int InitializePrinter();
   int StartDocumentPrinting();
   int EndDocumentPrinting();
```


© 2003 - 2004 Software Resources International S.A

```
// Auxilliary flags
int reading_status;
int DocumentIsOpen;

// Buffer for printing
char buffer[MAX_BUFFER_SIZE];
int  buffer_size;

// Specific Parameters for printing
int     FontSize;
int     offset;
HDC     hdcPrinter;
DOCINFO docinfo;
CDC     dcPrinter;
CFont   font;
CFont   *pOldFont;

// Page parameters
      int HRes;
      int VRes;
      int MaxCharactersInLine;
      int TabSize;

// Synchronization
HANDLE m_events[TotalEvents];    // Event handles

// Message handlers
HANDLE NoDataReceived;
      HANDLE LineIsReceived;
HANDLE BadConnection;
HANDLE GetNextLine;

CRITICAL_SECTION GlobalCriticalSection;

// Tray Icons
NOTIFYICONDATA Icon;
HICON hDefIcon;
HICON hWorkIcon;
char DefaultMessage[80];
char WorkingMessage[80];
```

```
    void GetSystemErrorMessage( DWORD dwErr );

protected:
    // Connection parameters
    char hostname[256];
    long portnum;

    // font face name
    char m_FontName[256];

    int SockErr;

    // socket for communicaion with LPV11
    Socket m_LPV11Socket;

        //{{AFX_VIRTUAL(CVAXPRINTApp)
        public:
        virtual BOOL InitInstance();
        virtual int ExitInstance();
        //}}AFX_VIRTUAL

        //{{AFX_MSG(CVAXPRINTApp)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional
declarations immediately before the previous line.

#endif // !
defined(AFX_VAXPRINT_H__54DCE038_9078_11D4_A11
E_0080C853BDF9__INCLUDED_)
#endif

#endif
```

 © 2003 - 2004 Software Resources International S.A

A.5 WINSOCK.CPP source file

This is the TCP/IP socket used inside the VAXPrint application

```
//
// Copyright (C) 2004 Software Resources International
SA
//
// This code is provided on an 'as is' basis.
// Software Resources International SA carries no
responsibility for
// the application of this code in any way or form.
//

// -----------------------------------------------------
#include <errno.h>
#include "WinSock.h"

// -----------------------------------------------------
int Socket::m_SocketCount = 0;

// -----------------------------------------------------
int Socket::Start()
{
   if( !m_SocketCount ) {
      WSADATA info;
      if (WSAStartup(MAKEWORD(2,0), &info)) {
         return( WSAGetLastError() );
      }
   }
   ++m_SocketCount;

   return( 0 );
}

// -----------------------------------------------------
void Socket::End()
{
#if 0
```

```
   // Already done by Charon, overwise Charon can't
terminate until
   // VAXPrint has manually finished
   WSACleanup();
#endif
}

// -----------------------------------------------------
Socket::Socket() : sock( 0 )
{
   Start();

   refCounter = new int( 1 );
}

// -----------------------------------------------------
int Socket::Initialize()
{
   int rc = 0;
   int so_true = 1;

   sock = ::WSASocket( AF_INET, SOCK_STREAM, 0, 0,
0, 0 );
   if( sock == INVALID_SOCKET ) {
      rc = ::WSAGetLastError();
      return( rc );
   }

   if (::setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
                (const char *)&so_true, sizeof(so_true))
== SOCKET_ERROR ) {
      rc = ::WSAGetLastError();
      closesocket( sock );
      return( rc );
   }

   return( rc );
}

// -----------------------------------------------------
Socket::Socket( SOCKET s ) : sock( s )
```

 © 2003 - 2004 Software Resources International S.A

```
{
   Start();
   refCounter = new int( 1 );

   SockErr = 0;
};

// ------------------------------------------------------
Socket::~Socket()
{
   if (! --(*refCounter)) {
      Close();
      delete refCounter;
   }

   --m_SocketCount;
   if( !m_SocketCount ) {
      End();
   }
}

// ------------------------------------------------------
Socket::Socket( const Socket& other )
{
   refCounter = other.refCounter;
   (*refCounter)++;
   sock = other.sock;

   m_SocketCount++;

   SockErr = other.SockErr;
}

// ------------------------------------------------------
Socket &Socket::operator=( Socket& other )
{
   (*other.refCounter)++;

   refCounter = other.refCounter;
   sock = other.sock;
```

© 2003 - 2004 Software Resources International S.A

```
   m_SocketCount++;

   SockErr = other.SockErr;

   return *this;
}

// ------------------------------------------------------
void Socket::Close()
{
#if 0
   // WSA library already closed by Charon,
   // overwise Charon can't terminate until VAXPrint
manually finished
   if( sock != INVALID_SOCKET ) {
      if( closesocket(sock) != 0 ) {
         SockErr = ::WSAGetLastError();
      }
   }
#endif
}

// ------------------------------------------------------
int Socket::ReceiveByte()
{
   unsigned long received = 0;
   unsigned long flags;
   WSABUF        buf_dsc;
   OVERLAPPED    ovl;

   if( ::WSAWaitForMultipleEvents(1, &hSocketEvent,
false, 1000, false)
      != WSA_WAIT_FAILED ) {

      WSANETWORKEVENTS NetEv;
      if( WSAEnumNetworkEvents(sock,hSocketEvent,&
NetEv) == 0 ) {

         // now data inside read queue
         if( NetEv.lNetworkEvents & FD_READ ) {
            buf_dsc.buf = &sock_buf[0];
```

 © 2003 - 2004 Software Resources International S.A

```
            buf_dsc.len = 1; // reading byte

            flags = MSG_PARTIAL;
            ovl.Offset = 0;
            ovl.OffsetHigh = 0;

            if( ::WSARecv( sock, &buf_dsc, 1, &received,
&flags, 0, 0) != 0 ) {
               int err = WSAGetLastError();
               if( (err == WSA_OPERATION_ABORTED) ||
                  (err == WSAECONNABORTED) ||
                  (err == WSAESHUTDOWN) ||
                  (err == WSAENOTSOCK) ||
                  (err == WSAENOTCONN) ||
                  (err == WSAENETDOWN) ||
                  (err == WSANOTINITIALISED) ) {
                  return( -1 );
               }
               return( 0 );
            }
            else {
               return( 1 );
            }
         }
         else
         if( NetEv.lNetworkEvents & FD_CLOSE ) {
            // host close connection

            WSACloseEvent( hSocketEvent );
            return( -1 );
         }
      }
   }
   else {
      return( 0 );
   }

   return( received );
}

// -------------------------------------------------------
```

```
int Socket::SendByte( const char ch )
{
   int sended;
   sended = send( sock, &ch, 1, 0 );
   return( sended );
}

// -----------------------------------------------------
int Socket::ConnectSocket( const char *host, int port )
{
   hostent *he;
   if( (he=gethostbyname(host)) == NULL ) {
      SockErr = ::WSAGetLastError();
      return( SockErr );
   }

   sockaddr_in addr;
   addr.sin_family = AF_INET;
   addr.sin_port = htons(port);
   addr.sin_addr = *((in_addr *)he->h_addr);
   memset(&(addr.sin_zero), 0, 8);

   if( ::connect(sock, (sockaddr *) &addr,
sizeof(sockaddr)) == SOCKET_ERROR ) {
      SockErr = ::WSAGetLastError();
      return( SockErr );
   }

   hSocketEvent = WSACreateEvent();
   if( ::WSAEventSelect(sock, hSocketEvent,
FD_READ) ) {
      return FALSE;
   }

   return( 0 );
}
```

 © 2003 - 2004 Software Resources International S.A

A.6 WINSOCK.H source file

This is the header file for the TCP/IP socket implementation.

```
//
// Copyright (C) 2004 Software Resources International SA
//
// This code is provided on an 'as is' basis.
// Software Resources International SA carries no responsibility for
// the application of this code in any way or form.
//

#ifndef __SOCKET_H__
#define __SOCKET_H__

// ------------------------------------------------------
#include <WinSock2.h>

// ------------------------------------------------------
class Socket
{
public:
    Socket();
    Socket(SOCKET s);

    virtual ~Socket();
    Socket( const Socket& );
    Socket & operator=( Socket& );

    int Initialize();

    int ConnectSocket( const char *host, int port );

    // Send/Receive data using socket
    int SendByte( char ch );
    int ReceiveByte();
```

```
   void Close();

   int SockErr;

protected:

   HANDLE hSocketEvent;

   int *refCounter;

public:
   SOCKET sock;
   char sock_buf[256]; // only first byte really used

private:
   static int  Start();
   static void End();
   static int  m_SocketCount;
};

#endif
```

A.7 Configuration file entry

The CHAPI interface, the LPV11 DLL and the VAXprint application are defined in the CHARON-VAX configuration file as listed below. Note that a CHAPI enabled version of CHARON-VAX is required and that the "**set lpv1 …"** line must be entered as a single text line:

```
#
# configure a parallel printer port
#
load chapi lpv1
set lpv1 dll=lpv11.dll parameters="port=10004
application='vaxprint.exe -port=10004 -font=\Courier
New\ -fontsize=10'"
```

 © 2003 - 2004 Software Resources International S.A

Index

Components
- Initialization....4, 18
- Naming....4

Descriptor field
- b_address_range....16
- base_b_address....12
- base_i_vector....12
- clear_irq....13
- context....12, 15
- create_io_space....13
- decrypt_data_block....14
- destroy_io_space....14
- i_priority....16
- log_message....14
- move_io_space....13
- n_of_i_vector....16
- put_ast....12
- put_irq....12
- put_sst....12
- read....17
- read_mem....13
- reset....17
- set_configuration....17
- start....17
- stop....16
- write....17
- write_mem....13

Notation conventions....vi

Operation class
- Bus interrupt....8, 25, 26
- Bus interrupt acknowledge....8, 27
- Bus power....9, 33, 34
- Bus reset....9, 34
- Configuration change request....9, 35

Device control and status..8, 21, 22, 23, 24
DMA...8, 28, 29
Message log request...9, 35
Protection and license verification..9, 36
Synchronization request...9, 30, 31
Synchronization request acknowledgement......................................9, 30, 32

 © 2003 - 2004 Software Resources International S.A

Reader's comments

We appreciate your comments, suggestions, criticism and updates of this manual by email. Our email address is: **charon@softresint.com**

Please mention that it concerns the CHAPI/VAX-Qbus Application interface manual, version 16 June 2004, document number 30-16-013.

If you found any errors, please list them with their page number.