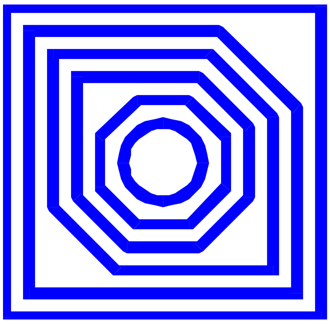
CHARON™-TB / W32

An API based system emulator for 32 bit Windows host systems

Production version

CHAPI source code listings

This release has undergone basic testing, but was not subject to a product QA test, which still has to be defined. This release is intended for familiarization with the product concept, not for production use.



CHAPI source code listings for CHARON™-MA / W32, an API based system emulator for 32 bit Windows host systems.

Document number: 30-09-06 (Production version)

Draft version:

Deleted: 14

Deleted: Aug

26 Oct 2007

Ownership Notice

Software Resources International SA owns the rights, including proprietary rights, copyrights, trademarks, and world-wide distribution rights to a methodology for the execution of applications and system software by emulating the original hardware components of a legacy system in software and/or microcode. This methodology is based on a portable software architecture henceforth referred to as CHARON™.

Software Resources International makes no representations that the implementation of the CHARON software architecture as described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Trademarks

The CHARON name with logo is a trademark of Software Resources International. VAX, Q-bus, VMS and OpenVMS 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.

Life support applications

The CHARON products from Software Resources International are not designed for use in systems where malfunction of a CHARON product can reasonably be expected to result in a personal injury. Software Resources International's customers using or selling our CHARON products for use in such applications do so at their own risk and agree to fully indemnify Software Resources International for any damages resulting from such improper use or sale.

Preface

This document is a part of User Manual for CHARON™-TB / W32. It contains source code listings which are referenced from the mentioned manual.

1. Source code listings

All source code files maybe found in the "<CHARON-MA_Install_Path>\Unsupported\Chapi_lib_<version>" after CHARON-TB installation. They consist of mentioned below header files for CHAPI support libraries and a few examples of CHAPI devices based on these libraries.

1.1. Header files for CHAPI support libraries

1.1.1. 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 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_H__)
#define __CHAPI_H__

//
// CHAPI versioning information - do not mix up with product versioning
// information. CHAPI functionality changes will always be described in
// release notes, so it will always be possible to see when some new
// functionality is added to CHAPI in terms of CHAPI version. Old
// functionality is never removed from CHAPI. It means that when somebody
// is using any new functionality it should check CHAPI version supported by
// the emulator under which CHAPI module is running in order to be sure that
// used functionality is supported there. CHAPI version numbers can be get
// from running emulator using a couple of CHAPI methods defined below:
// get_chapi_major_version(); get_chapi_minor_version().
//

// Current CHAPI version is 3.0
#define CHAPI_MAJOR_VERSION_NO  3
#define CHAPI_MINOR_VERSION_NO  0


#if defined(_MSC_VER)
#define CHAPI __cdecl
#if defined(__cplusplus)
#define CHAPI_INIT(n) extern "C" \
__declspec(dllexport) void * __cdecl n##_INIT
#else // defined(__cplusplus)
```

```
#define CHAPI_INIT(n) \
__declspec(dllexport) void * __cdecl n##_INIT
#endif // defined(__cplusplus)
#else // defined(_MSC_VER)
#define CHAPI
#endif

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


//=============================================================================
// Definition of CHAPI input context
//

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


//-----------------------------------------------------------------------------
// AST/SST relative stuff
//
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 int (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 int (CHAPI * __chapi_put_sst_procedure_p)
    (const struct __chapi_in * ci,
     unsigned long delay,
     __chapi_sst_handler_p fun, void * arg1, int arg2);


//-----------------------------------------------------------------------------
// IRQ/BRQ relative stuff.
//
typedef 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)

//
```

```
// Obsolete way to handle IRQ - left only to support old applications.
// This interface uses internal IRQ queues which makes differencies with the
// actual way of interrupts processing in hardware. Use new interface defined
// below.
//
typedef int (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);

//
// New way to handle IRQ - no queues.
// This way should be used for newly developed applications in a couple with
// chapi_brq_t class defined in chapi_lib.h header which wrappes interface
// conveniently.
//
typedef unsigned int brq_handle_t;
typedef __chapi_irq_handler_p __chapi_brq_acknowledge_p;

#if !defined(brq_acknowledge)
#define brq_acknowledge __chapi_brq_acknowledge_p
#endif // !defined(brq_acknowledge)

typedef int (CHAPI * __chapi_sa_handler_p)
    (void * arg1, int arg2, int c_no);


typedef brq_handle_t (CHAPI * __chapi_connect_bus_request_p)
    (const struct __chapi_in * ci, int vector, int ipl,
     __chapi_brq_acknowledge_p brq_ack, void *arg1, int arg2);

typedef void (CHAPI * __chapi_set_bus_request_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle);

typedef void (CHAPI * __chapi_clear_bus_request_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle);

typedef void (CHAPI * __chapi_enable_bus_request_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle, bool enable);

typedef void (CHAPI * __chapi_set_bus_request_affinity_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle, unsigned int mask);

typedef void (CHAPI * __chapi_set_affinity_callback_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle,
    __chapi_sa_handler_p sa_callback, void *arg1, int arg2);

typedef unsigned int (CHAPI * __chapi_get_bus_server_mask_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle);

typedef bool (CHAPI * __chapi_get_attention_objects_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle, int cpu_no,
    volatile unsigned long *& attention_object, unsigned long &
attention_value);

typedef bool (CHAPI * __chapi_get_brq_objects_p)
```

```
    (const struct __chapi_in * ci, brq_handle_t brq_handle, int cpu_no,
    volatile unsigned long *& brq_object, unsigned long & brq_mask);

typedef int (CHAPI * __chapi_get_vector_p)
    (const struct __chapi_in * ci, int vector);

typedef void (CHAPI * __chapi_set_brq_vector_p)
    (const struct __chapi_in * ci, brq_handle_t brq_handle, int vector);


//-----------------------------------------------------------------------------
// Memory access methods.
//

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,
     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);


//-----------------------------------------------------------------------------
// Licensing routines
//
typedef bool (CHAPI * __chapi_get_license_no_procedure_p)
    (const struct __chapi_in * ci,
     unsigned int *license_serial_no);

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

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

//-----------------------------------------------------------------------------
// Message logging / debugging routines
//
```

```
// Possible types of messages to log
typedef enum _log_message_type_t {
    error_msg_type,
    warning_msg_type,
    info_msg_type
} log_message_type_t;

//
// CHAPI message code decoding. CHAPI message code is used as subcode within
// CHARON message preallocated for CHAPI use... the format is as follows:
//
// 31       24 23       16 15         0
// <vendor_id> <device_id> <message_id>
//
// Note: CHAPI messages subcoding is used now to distinguish between different
//       CHAPI messages mapped to single CHARON message code from message
database.
//       It is planned in the future to add some kind of CHAPI message database
//       and primitives to get the message from that database in different
languages
//       instead of hardcoding these messages withing the device code like it is
//       done for the moment.
//
typedef unsigned int log_message_id_t;

// vendor_id = 0 is reserved for SRI devices
enum {
    VENDOR_ID_SRI   =   0
};

// A few SRI device ids - some for libraries, others for devices...
enum {
    DEVICE_ID_HW        =   0,  // CHAPI_HW.DLL objects
    DEVICE_ID_SERIAL,           // CHAPI_SERIAL.DLL objects
    DEVICE_ID_STORAGE,          // CHAPI_STORAGE.DLL objects
    DEVICE_ID_PARALLEL,         // CHAPI_PARALLEL.DLL objects
    DEVICE_ID_QBUS,             // CHAPI QBUS adapter implementatiion
    DEVICE_ID_UNIBUS,           // CHAPI UNIBUS adapter implementatiion
    DEVICE_ID_DHV11,            // CHAPI DHV11 device implementation
    DEVICE_ID_DLV11,            // CHAPI DLV11/DL11 devices implementation
    DEVICE_ID_DRV11,            // CHAPI DRV11/DR11-C devices implementation
    DEVICE_ID_DRV11_621_PORT,   // CHAPI DRV11/DR11-C port mapped to SENSORAY
621 adapter
    DEVICE_ID_DRV11WA,          // CHAPI DCI-1100 based DRV11-WA implementation
    DEVICE_ID_DRV11WA_621,      // CHAPI DRV11WA mapped to SENSORAY 621 adapter
    DEVICE_ID_DRV11WA_621_PORT, // CHAPI DRV11WA port mapped to SENSORAY 621
adapter
    DEVICE_ID_HTIME,            // CHAPI HTIME pseudo device
    DEVICE_ID_RLV12,            // CHAPI RL11/RLV12 devices implementation
    DEVICE_ID_TSV05,            // CHAPI TS11/TSV05 devices implementation
    DEVICE_ID_VCB02,            // CHAPI VCB02 device implementation
    DEVICE_ID_LPV11,            // CHAPI LP11/LPV11 device implementation
    DEVICE_ID_SDZV11,           // CHAPI_SHELL_DZV11
    DEVICE_ID_DH11,             // CHAPI DH11 devices implementation
    DEVICE_ID_VT30H             // CHAPI VT30-H devices implementation
};


// Generates message code from its components.
```

```
#define CHAPI_MSG_ID(vendor, device, code)  ((vendor << 24) | (device << 16) |
code)


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

typedef void (CHAPI * __chapi_log_message_ex_procedure_p)
    (const struct __chapi_in * ci, log_message_type_t log_msg_type,
    const char *file, int line, log_message_id_t log_msg_id, const char *fmt,
...);

// Trace level will be in range [0, 10]
typedef unsigned char trace_level_t;

typedef void (CHAPI * __chapi_debug_trace_procedure_p)
    (const struct __chapi_in * ci,
     trace_level_t trace_level, const char *fmt, ...);


//---------------------------------------------------------------------------
// Configuration options relative routines
//

// Possible types of configuration options
typedef enum _config_option_t {
    option_type_integer,
    option_type_boolean,
    option_type_string
} config_option_t;

typedef void (CHAPI * __chapi_add_config_option_p)
    (const struct __chapi_in * ci,
     const char *opt_name, config_option_t opt_type, int opt_vals_count,
     void *opt_buffer, size_t opt_size);

typedef bool (CHAPI * __chapi_set_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx,
    void *val);

typedef bool (CHAPI * __chapi_set_and_disable_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx,
    void *val);

typedef void (CHAPI * __chapi_undo_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef void (CHAPI * __chapi_commit_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef bool (CHAPI * __chapi_is_option_value_specified_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef bool (CHAPI * __chapi_is_option_value_changed_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef void (CHAPI * __chapi_option_value_change_ack_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);
```

```
typedef void (CHAPI * __chapi_enable_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx, bool
force);

typedef void (CHAPI * __chapi_freeze_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef void (CHAPI * __chapi_disable_option_value_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);

typedef bool (CHAPI * __chapi_is_option_value_hidden_p)
    (const struct __chapi_in * ci, const char *opt_name, int opt_val_idx);


//----------------------------------------------------------------------------
// Product versioning information stuff
//

// Get product identification string
typedef const char* (CHAPI * __chapi_get_product_ident_p)
    (const struct __chapi_in * ci);

// Get harware model
typedef const char* (CHAPI * __chapi_get_hardware_model_p)
    (const struct __chapi_in * ci);

// Get harware name
typedef const char* (CHAPI * __chapi_get_hardware_name_p)
    (const struct __chapi_in * ci);

// Get product Copyright string
typedef const char* (CHAPI * __chapi_get_product_copyright_p)
    (const struct __chapi_in * ci);

// Get product custom string
typedef const char* (CHAPI * __chapi_get_product_custom_string_p)
    (const struct __chapi_in * ci);

// Get major version number for the product
typedef int (CHAPI * __chapi_get_product_major_version_p)
    (const struct __chapi_in * ci);

// Get minor version number of the product
typedef int (CHAPI * __chapi_get_product_minor_version_p)
    (const struct __chapi_in * ci);

// Get build number for the project
typedef int (CHAPI * __chapi_get_product_build_version_p)
    (const struct __chapi_in * ci);

// Get major version number for the CHAPI supported by the product
typedef int (CHAPI * __chapi_get_chapi_major_version_p)
    (const struct __chapi_in * ci);

// Get minor version number for the CHAPI supported by the product
typedef int (CHAPI * __chapi_get_chapi_minor_version_p)
    (const struct __chapi_in * ci);


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

```
// Support for bus adapter
//

typedef bool (CHAPI * __chapi_intercept_bus_address_space_p)
    (const struct __chapi_in * ci);

typedef void (CHAPI * __chapi_release_bus_address_space_p)
    (const struct __chapi_in * ci);

typedef unsigned int (CHAPI * __chapi_get_configured_ram_size_p)
    (const struct __chapi_in * ci);

typedef unsigned int (CHAPI * __chapi_get_ram_segment_p)
    (const struct __chapi_in * ci, int n_of_segment, unsigned int &addr,
        char * &base);

typedef void (CHAPI * __chapi_read_bus_timeout_p)
    (const struct __chapi_in * ci);

typedef void (CHAPI * __chapi_read_bus_abort_p)
    (const struct __chapi_in * ci);

typedef void (CHAPI * __chapi_write_bus_timeout_p)
    (const struct __chapi_in * ci);

typedef void (CHAPI * __chapi_write_bus_abort_p)
    (const struct __chapi_in * ci);

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


//----------------------------------------------------------------------------
// Support for different buses.
//

// Supported buses
typedef enum _supported_buses_t {
    UNKNOWN_BUS     =   0x00000000,
    QBUS            =   0x00000001,
    UNIBUS          =   0x00000002
} supported_buses_t;

enum {
    QBUS_IO_OFFSET      =   0x3FE000,
    QBUS_IO_SIZE        =   0x002000,

    UNIBUS_IO_OFFSET    =   0x03E000,
    UNIBUS_IO_SIZE      =   0x002000,

    QBUS_IO_LAST        =   QBUS_IO_OFFSET + QBUS_IO_SIZE - 1,
    QU_IO_SHIFT         =   - QBUS_IO_OFFSET + UNIBUS_IO_OFFSET,

    UNIBUS_IO_LAST      =   UNIBUS_IO_OFFSET + UNIBUS_IO_SIZE - 1,
    UQ_IO_SHIFT         =   - UNIBUS_IO_OFFSET + QBUS_IO_OFFSET
};

typedef supported_buses_t (CHAPI * __chapi_get_bus_type_p)
    (const struct __chapi_in * ci);
```

```
typedef struct __chapi_in {
    __chapi_in_context_p context;

    unsigned int base_b_address;
    unsigned int base_i_vector;

    // Synchronization calls
    __chapi_put_ast_procedure_p put_ast;
    __chapi_put_sst_procedure_p put_sst;

    // Old way of IRQ processing calls
    __chapi_put_irq_procedure_p put_irq;
    __chapi_clear_irq_procedure_p clear_irq;

    // New way of IRQ processing calls
    __chapi_connect_bus_request_p connect_bus_request;
    __chapi_set_bus_request_p set_bus_request;
    __chapi_clear_bus_request_p clear_bus_request;
    __chapi_enable_bus_request_p enable_bus_request;
    __chapi_set_bus_request_affinity_p set_bus_request_affinity;
    __chapi_set_affinity_callback_p set_affinity_callback;
    __chapi_get_vector_p get_vector;

    // IRQ processing support for hardware replacement boards
    __chapi_get_bus_server_mask_p get_bus_server_mask;
    __chapi_get_attention_objects_p get_attention_objects;
    __chapi_get_brq_objects_p get_brq_objects;

    // Emulated DMA calls
    __chapi_read_mem_procedure_p read_mem;
    __chapi_write_mem_procedure_p write_mem;

    // Address spaces handling calls
    __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;

    // Licensing calls
    __chapi_get_license_no_procedure_p get_license_no;
    __chapi_encrypt_data_block_procedure_p encrypt_data_block;
    __chapi_decrypt_data_block_procedure_p decrypt_data_block;

    // Message logging calls
    __chapi_log_message_procedure_p log_message;
    __chapi_log_message_ex_procedure_p log_message_ex;
    __chapi_debug_trace_procedure_p debug_trace;

    // Configuration option processing calls
    __chapi_add_config_option_p add_config_option;
    __chapi_set_option_value_p set_option_value;
    __chapi_undo_option_value_p undo_option_value;
    __chapi_commit_option_value_p commit_option_value;
    __chapi_is_option_value_specified_p is_option_value_specified;
    __chapi_is_option_value_changed_p is_option_value_changed;
    __chapi_option_value_change_ack_p option_value_change_ack;

    // Bus adapter and support calls
    __chapi_intercept_bus_address_space_p intercept_bus_address_space;
    __chapi_release_bus_address_space_p release_bus_address_space;
    __chapi_get_configured_ram_size_p get_configured_ram_size;
    __chapi_get_ram_segment_p get_ram_segment;

    __chapi_read_bus_timeout_p read_bus_timeout;
    __chapi_read_bus_abort_p read_bus_abort;
    __chapi_write_bus_timeout_p write_bus_timeout;
    __chapi_write_bus_abort_p write_bus_abort;

    // New way of IRQ processing calls continuation ...

    //
    // This method is used in order to change interrupt vector for specified
    // bus request.
    //
    __chapi_set_brq_vector_p set_brq_vector;

    //
    // This method is used in order to translate specified bus address into the
    // emulated system RAM address.
    //
    __chapi_translate_for_dma_p translate_for_dma;

    //
    // This method is used to get bus type of the device owning bus.
    // This is required to support different bus types within the single
    // implementational module, e.g. QBUS device very often have its UNIBUS
    // equivalent with almost the same functionality, but address processing is
    // different, so we have to know bus type we are connected to at runtine.
    //
    __chapi_get_bus_type_p get_bus_type;

    //
    // Simple extension of configuration options interface which allows
    // disabling/freezing of option values. This is required in the light
    // of access to CHARON interactive console which requires protection
    // of option values even of CHAPI devices...
    //
    __chapi_set_and_disable_option_value_p set_and_disable_option_value;
    __chapi_enable_option_value_p enable_option_value;
    __chapi_freeze_option_value_p freeze_option_value;
    __chapi_disable_option_value_p disable_option_value;
    __chapi_is_option_value_hidden_p is_option_value_hidden;

    // Product versioning information
    __chapi_get_product_ident_p get_product_ident;
    __chapi_get_hardware_model_p get_hardware_model;
    __chapi_get_hardware_name_p get_hardware_name;
    __chapi_get_product_copyright_p get_product_copyright;
    __chapi_get_product_custom_string_p get_product_custom_string;
    __chapi_get_product_major_version_p get_product_major_version;
    __chapi_get_product_minor_version_p get_product_minor_version;
    __chapi_get_product_build_version_p get_product_build_version;

    __chapi_get_chapi_major_version_p get_chapi_major_version;
    __chapi_get_chapi_minor_version_p get_chapi_minor_version;

    //
    // All new methods/data have to be added to the end of this structure,
otherwise
    // all customer chapi devices have to be rebuilt with the new header file!!!
    //

} chapi_in;


//============================================================================
// Definition of CHAPI output context
//

#if !defined(__chapi_out_context_p)
typedef void * __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 void (CHAPI * __chapi_mapping_register_updated_p)
    (const struct __chapi_out * co,
     int reg_set, int reg_no, int val);

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

typedef int (CHAPI * __chapi_set_configuration_ex_procedure_p)
    (const struct __chapi_out * co);

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

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

typedef unsigned int (CHAPI * __chapi_get_bus_address_range_procedure_p)
    (const struct __chapi_out * co, supported_buses_t owning_bus_type);

typedef struct __chapi_out {
    __chapi_out_context_p context;

    unsigned int base_b_address;
    unsigned int b_address_range;
    unsigned int base_i_vector;
    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_mapping_register_updated_p mapping_register_updated;

    __chapi_set_configuration_procedure_p set_configuration;
    __chapi_set_configuration_ex_procedure_p set_configuration_ex;

    __chapi_setup_bus_requests_procedure_p setup_bus_requests;

    __chapi_run_interactive_command_procedure_p run_interactive_command;

    //
    // Almost the same device (functionally the same) may have different
    // address range of their I/O page window, e.g. RL11 has 4 registers and
    // 8 bytes window range while RLV11 has 5 registers and 16 bytes window
range.
    // This makes sense to ask device for address range dynamically depending on
    // bus type this device is connected (when device supports a number of
different
    // buses).
    //
    __chapi_get_bus_address_range_procedure_p get_bus_address_range;

    // This data member defines supported buses
    unsigned int supported_buses;

    //
    // All new methods/data have to be added to the end of this structure,
otherwise
    // all customer chapi devices have to be rebuilt with the new header file!!!
    //

} chapi_out;

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

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

1.1.2. CHAPI_LIB.H

This appendix contains listing of the **chapi_lib.h** file which contains definitions for C++ wrappers to CHAPI protocol and some general functions which maybe useful in CHAPI development. C++ implementation file is not provided in sources – just use **chapi.dll**/**chapi.lib** and header.

```
//
// Copyright 2006 Software Resources International
```

```
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_LIB_H__)
#define __CHAPI_LIB_H__

#undef _WIN32_WINNT
#define _WIN32_WINNT 0x0500

#include <windows.h>

#if defined(IN_CHAPI_DLL)
#define __chapi_lib__ __declspec(dllexport)
#endif // defined(IN_CHAPI_DLL)

#if defined(USE_CHAPI_DLL)
#define __chapi_lib__ __declspec(dllimport)
#endif // defined(USE_CHAPI_DLL)

#if !defined(__chapi_lib__)
#define __chapi_lib__
#endif // !defined(__chapi_lib__)


// Disable a number of annoying warnings...
#if defined(_MSC_VER)

// warning C4355: 'this' : used in base member initializer list
#pragma warning(disable:4355)

// warning C4800: 'BOOL' : forcing value to bool 'true' or 'false' (performance
warning)
#pragma warning(disable:4800)
#endif

#include "chapi.h"


//----------------------------------------------------------------------------

//
// Debug/trace and message logging default macroses to simplify messaging/
// debugging inside library and third-party devices.
//
// Format is as follows:
//
// LOGMSG((_ERR_MSG_, <message_id>, <format string>, ...))
// LOGMSG((_WARN_MSG_, <message_id>, <format string>, ...))
// LOGMSG((_INFO_MSG_, <message_id>, <format string>, ...))
//
// TRACE((L(k), <format string>, ...)),
// where k in [0, 10] defines trace level.
//
```

```
// Message logging ...
#define _ERR_MSG_ ci, error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ ci, warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ ci, info_msg_type, __FILE__, __LINE__

#define LOGMSG(x)\
    if(ci != 0 && ci->log_message_ex != 0) { \
        ci->log_message_ex x; \
    }

// Debug trace ...
#define L(n) ci, n
#define TRACE(x)\
    if(ci != 0 && ci->debug_trace != 0) { \
        ci->debug_trace x; \
    }


//---------------------------------------------------------------------------
// Some useful type definitions
//

// typical lengths of io transactions on the emulated bus
enum io_size_t {
    io_byte_t = 0,      // LOG2(sizeof(byte_t))
    io_word_t = 1,      // LOG2(sizeof(word_t))
    io_dword_t = 2,     // LOG2(sizeof(dword_t))
    io_qword_t = 3,     // LOG2(sizeof(qword_t))
};

typedef char            byte_t;
typedef unsigned char   ubyte_t;

typedef short           word_t;
typedef unsigned short  uword_t;

typedef int             dword_t;
typedef unsigned int    udword_t;


// Some pointers to this types
typedef udword_t *      udword_lp_t;
typedef const udword_t * const_udword_lp_t;
typedef volatile udword_t * volatile_udword_lp_t;


// Register access functions
inline word_t write_byte(int adr, byte_t byte, word_t word)
{
    return (word & ~(0xFF << ((adr & 1) << 3)))
         | ((byte & 0xFF) << ((adr & 1) << 3));
}

inline word_t read_byte(int adr, word_t word)
{
    return (byte_t)(word >> ((adr & 1) << 3));
}

inline word_t read_reg(int adr, io_size_t type, word_t word)
{
    return type == io_byte_t ? (word_t)(read_byte(adr, word)) : word;
};

inline word_t write_reg
    (int adr, dword_t val, io_size_t type, word_t word)
{
    return (word_t)(type == io_byte_t ? write_byte(adr, val, word) : val);
};

//----------------------------------------------------------------------------
// Bi-directional list template
//
template <class t> class l2list {
public:


    t * next() const {
        return (t *)list_next;
    };

    t * prev() const {
        return (t *)list_prev;
    };

protected:

    l2list()
      : list_next(this), list_prev(this), list_entrance(0)
    {;};

    //
    // Insert the list entry after the given one in the list.
    //

    virtual void insert(l2list * after, t ** entrance = 0) {
        list_next = after ? after->list_next : this;
        list_prev = after ? after : this;
        list_next->list_prev = list_prev->list_next = this;
        list_entrance = entrance;
        if (list_entrance && *list_entrance == 0) {
            *list_entrance = (t *)this;
        }
    };

    //
    // Insert the list entry before the given one in the list.
    //

    virtual void append(l2list * before, t ** entrance = 0) {
        list_next = before ? before : this;
        list_prev = before ? before->list_prev : this;
        list_next->list_prev = list_prev->list_next = this;
        list_entrance = entrance;
        if (list_entrance && *list_entrance == 0) {
            *list_entrance = (t *)this;
        }
    };

    virtual void remove() {
        if (list_entrance) {
            if (*list_entrance == this) {
                *list_entrance = (t *)list_next;
                if (*list_entrance == this) {
                    *list_entrance = 0;
                }
            }
        }
        list_next->list_prev = list_prev;
        list_prev->list_next = list_next;
        list_prev = list_next = this;
        list_entrance = 0;
    };

    virtual ~l2list() {

        //
        // Make sure we keep consistent the list in which the entry might be
        // linked.
        //

        remove();
    };

private:

    l2list  *list_prev;
    l2list  *list_next;

    t       **list_entrance;
};


//-----------------------------------------------------------------------------
// Simple wrapper to work with new style IRQ processing via CHAPI.
// It is the simplest way to work with BRQs in a newly defined way.
//
class __chapi_lib__ chapi_brq_t {
public:

    //-------------------------------------------------------------------------

    chapi_brq_t();
    chapi_brq_t(int ipl, __chapi_brq_acknowledge_p brq_ack, void *arg1, int arg2
= 0);

    ~chapi_brq_t();

    //-------------------------------------------------------------------------

    //
    // Abstract:
    //
    //  Connect BRQ to the bus. All methods are mapped to
    //  chapi_in::connect_bus_request() CHAPI protocol routine.
    //
    // Arguments:
    //
    //  ci          -   CHAPI input context specified by the CHARON kernel;
    //  vector      -   BRQ vector to use;
    //  ipl         -   BRQ level to use;
    //  brq_ack     -   BRQ acknowledge procedure;
    //  arg1, arg2  -   BRQ acknowledge procedure arguments.
    //
    // Returns:
    //
    //  Operation status.
    //
    bool connect(const struct __chapi_in *ci, int vector);
    bool connect(const struct __chapi_in *ci, int ipl,
        __chapi_brq_acknowledge_p brq_ack, void *arg1 = 0, int arg2 = 0);
    bool connect(const struct __chapi_in *ci, int vector, int ipl,
        __chapi_brq_acknowledge_p brq_ack, void *arg1 = 0, int arg2 = 0);

    //
    // Abstract:
    //
    //  Set connected BRQ (brq_handle != 0).
    //  chapi_in::set_bus_request() CHAPI protocol call is used here.
    //
    // Arguments:
    //
    //  None.
    // Returns:
    //
    //  None.
    //
    void set();

    //
    // Abstract:
    //
    //  Clear connected BRQ (brq_handle != 0).
    //  chapi_in::clear_bus_request() CHAPI protocol call is used here.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    void clear();

    //
    // Abstract:
    //
    //  Enable / disable BRQ on the bus.
    //  chapi_in::enable_bus_request() CHAPI protocol call is used here.
    //
    // Arguments:
    //
    //  ena -   enable BRQ if true, disable it otherwise.
    //
    // Returns:
    //
    //  None.
    //
    void enable(bool ena = true);
```

```
//
// Abstract:
//
//  Set BRQ vector.
//
// Arguments:
//
//  vector    -   BRQ vector to set.
//
// Returns:
//
//  None.
//
void set_vector(int vector);

//
// Abstract:
//
//  Set BRQ affinity mask. It means that BRQ can be processed by specified
//  CPUs only.
//
// Arguments:
//
//  mask    -   CPU affinity mask for specified BRQ.
//
// Returns:
//
//  None.
//
void set_affinity_mask(unsigned int mask = 1);

//
// Abstract:
//
//  Set BRQ affinity callback. This function is called when CPU affinity
//  fir BRQ is changed.
//
// Arguments:
//
//  sa_callback -   affinity change call;back routine;
//  arg1        -   first argument to the callback;
//  arg2        -   second argument to the callback.
//
// Returns:
//
//  None.
//
void set_affinity_callback(__chapi_sa_handler_p sa_callback,
    void *arg1, int arg2);

//
// Abstract:
//
//  Retreives bus server mask which specifies if particular CPU number is
//  working as bus server.
//
// Arguments:
//
//  None.
//
```

```
    // Returns:
    //
    //  Bus server mask.
    //
    unsigned int get_bus_server_mask();

    //
    // Abstract:
    //
    //  Get CPU attention objects for particular CPU in order to have
possibility
    //  to interrupt CPU loop and force CPU to check for BRQs. Attention objects
    //  are represented by the pair {address, value to write}. This technique is
    //  designed for the case of hardware used in device emulation, pure
emulation
    //  doesn't require this stuff.
    //
    // Arguments:
    //
    //  cpu_no  -   CPU number to get attention objects for;
    //  attention_object    -   pointer to the location which should be written
with
    //                          specific value to force BRQ checking;
    //  attention_value     -   value which should be written to specified above
    //                          location in order to force BRQ checking by the
CPU.
    //
    // Returns:
    //
    //  Operation status.
    //
    bool get_attention_objects(unsigned int cpu_no,
        volatile unsigned long *& attention_object,
        unsigned long & attention_value);

    //
    // Abstract:
    //
    //  Get CPU BRQ objects for particular CPU in order to have direct access
    //  to the CPU BRQ mask. This technique is designed for the case of hardware
    //  used in device emulation, pure emulation doesn't require this stuff.
    //
    // Arguments:
    //
    //  cpu_no      -   CPU number to get attention objects for;
    //  brq_object  -   ;
    //  brq_mask    -   .
    //
    // Returns:
    //
    //  Operation status.
    //
    bool get_brq_objects(unsigned int cpu_no,
        volatile unsigned long  *& brq_object,
        unsigned long & brq_mask);

protected:

    // Pointer to input communication context.
    const struct __chapi_in     *ci;
    // BRQ acknowledge callback
    __chapi_brq_acknowledge_p   brq_ack;

    // BRQ acknowledge callback parameters
    char    *arg1;
    int     arg2;

    // BRQ vector
    int vector;

    // BRQ IPL
    int ipl;

    // Handle to connected BRQ.
    brq_handle_t    brq_handle;

private:
};


//----------------------------------------------------------------------------
// This is a simple wrappers to work conveniently with configuration option
// part of CHAPI protocol.
//

//
// This is a base class for kind of CHAPI option
//
class __chapi_lib__ chapi_cfg_option_value_t;
class __chapi_lib__ chapi_cfg_option_t {
public:

    // Value should have access to the option internals
    friend class chapi_cfg_option_value_t;

    //------------------------------------------------------------------------

    chapi_cfg_option_t(const struct __chapi_in * ci, const char *opt_name,
        config_option_t opt_type, int opt_vals_count, int opt_size);

    // Copy constructor allows to pass option as parameter correctly
    chapi_cfg_option_t(const chapi_cfg_option_t &option);

    virtual ~chapi_cfg_option_t();

    //------------------------------------------------------------------------

protected:

    // Don't allow to assignements - it's dangerous and has no sense
    chapi_cfg_option_t & operator= (const chapi_cfg_option_t &option);

    // CHAPI input communication context
    const struct __chapi_in *ci;

    // Option name
    char *name;

    // Option type
    config_option_t type;

    // The size of option value
    int size;

    // First and last index of allocated options
    int f_index;
    int l_index;

    //
    // Array of option values. It is allocated during the option construction
    // when the number of option values is specified.
    //
    chapi_cfg_option_value_t **opt_vals;

    //
    // Storage for option values which will be passed to CHARON-MA core. The
size
    // of this storage in bytes is calculated as opt_size * opt_vals_count.
    //
    char *opt_vals_buffer;

private:
};


//
// This is a base class for any kind of CHAPI option value
//
class __chapi_lib__ chapi_cfg_option_value_t {
public:

    //-------------------------------------------------------------------------

    chapi_cfg_option_value_t(chapi_cfg_option_t *option, int opt_val_idx,
        void *opt_buffer);
    virtual ~chapi_cfg_option_value_t();

    //-------------------------------------------------------------------------

    //
    // Abstract:
    //
    //  Set the option value using supplied data.
    //  This method wrappes chapi_in::set_option_value entry.
    //
    // Arguments:
    //
    //  val -   pointer to the int, bool or character string, containing value
    //          to set
    //
    // Returns:
    //
    //  Operation result.
    //
    virtual bool set(void *val);

    //
    // Abstract:
    //
```

```
//  Set the option value using supplied data and disable on commit.
//  This method wrappes chapi_in::set_and_disable_option_value entry.
//
// Arguments:
//
//  val -   pointer to the int, bool or character string, containing value
//          to set
//
// Returns:
//
//  Operation result.
//
virtual bool set_and_disable(void *val);

//
// Abstract:
//
//  Undo option value change if any.
//  This method wrappes chapi_in::undo_option_value entry.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  None.
//
virtual void undo();

//
// Abstract:
//
//  Commit option value change if any. Undo impossible after that
//  This method wrappes chapi_in::commit_option_value entry.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  None.
//
virtual void commit();

//
// Abstract:
//
//  Checks if specified value was mentioned in configuration file.
//  This method wrappes chapi_in::is_option_value_specified entry.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  true if option value is specified and false - otherwise.
//
```

```
    virtual bool is_specified() const;

    //
    // Abstract:
    //
    //  Checks if specified value was changed in configuration file sinse the
    //  last call to commit()/change_ack().
    //  This method wrappes chapi_in::is_option_value_changed entry.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  true if option value is changed and false - otherwise.
    //
    virtual bool is_changed() const;

    //
    // Abstract:
    //
    //  Clear 'modify' flag for the option value.
    //  This method wrappes chapi_in::option_value_change_ack entry.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    virtual void change_ack();

    //
    // Abstract:
    //
    //  Enable freezed option to be changed, optionally makes hidden option
    //  value accessible.
    //
    //  This method wrappes chapi_in::enable_option_value entry.
    //
    // Arguments:
    //
    //  force   -   force hidden option to be accessible.
    //
    // Returns:
    //
    //  None.
    //
    virtual void enable(bool force = false);

    //
    // Abstract:
    //
    //  Protect option value from change.
    //  This method wrappes chapi_in::freeze_option_value entry.
    //
    // Arguments:
```

```
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    virtual void freeze();

    //
    // Abstract:
    //
    //  Makes option value hidden, i.e. not accessible for view/change.
    //  This method wrappes chapi_in::disable_option_value entry.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    virtual void disable();

    //
    // Abstract:
    //
    //  Check if option value is hidden or not.
    //  This method wrappes chapi_in::is_option_value_hidden entry.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  true - if option value is hidden; false - otherwise.
    //
    virtual bool is_hidden();

protected:

    // Communication context
    const struct __chapi_in * ci;

    // Option this value belongs to
    chapi_cfg_option_t *option;

    // Option value index within the option
    int opt_val_idx;

    //
    // Pointer to the location allocated by the option where our
    // value is stored.
    //
    char *opt_buffer;

private:
};
```

```
//
// Option and value for integers
//
class __chapi_lib__ chapi_integer_option_value_t;
class __chapi_lib__ chapi_integer_option_t : public chapi_cfg_option_t {
public:

    //--------------------------------------------------------------------------

    chapi_integer_option_t(const struct __chapi_in * ci, const char *opt_name,
        int opt_vals_count);
    chapi_integer_option_t(const chapi_cfg_option_t &option);

    virtual ~chapi_integer_option_t();

    //--------------------------------------------------------------------------

    chapi_integer_option_value_t & operator[] (int idx);

protected:
private:
};

class __chapi_lib__ chapi_integer_option_value_t
    : public chapi_cfg_option_value_t {
public:

    //--------------------------------------------------------------------------

    chapi_integer_option_value_t(chapi_cfg_option_t *option, int opt_val_idx,
        void *opt_buffer);
    virtual ~chapi_integer_option_value_t();

    //--------------------------------------------------------------------------

    operator int ();

    bool set(int val);

protected:
private:

};


//
// Option and value for boolean
//
class __chapi_lib__ chapi_bool_option_value_t;
class __chapi_lib__ chapi_bool_option_t : public chapi_cfg_option_t {
public:

    //--------------------------------------------------------------------------

    chapi_bool_option_t(const struct __chapi_in * ci, const char *opt_name,
        int opt_vals_count);
    chapi_bool_option_t(const chapi_bool_option_t &option);
    virtual ~chapi_bool_option_t();

    //----------------------------------------------------------------------

    chapi_bool_option_value_t & operator[] (int idx);

protected:
private:
};

class __chapi_lib__ chapi_bool_option_value_t
    : public chapi_cfg_option_value_t {
public:

    //----------------------------------------------------------------------

    chapi_bool_option_value_t(chapi_cfg_option_t *option, int opt_val_idx,
        void *opt_buffer);
    virtual ~chapi_bool_option_value_t();

    //----------------------------------------------------------------------

    operator bool ();

    bool set(bool val);

protected:
private:

};


//
// Option and value for strings
//
class __chapi_lib__ chapi_string_option_value_t;
class __chapi_lib__ chapi_string_option_t : public chapi_cfg_option_t {
public:

    //----------------------------------------------------------------------

    chapi_string_option_t(const struct __chapi_in * ci, const char *opt_name,
        int opt_vals_count, int opt_size);
    chapi_string_option_t(const chapi_string_option_t &option);

    virtual ~chapi_string_option_t();

    //----------------------------------------------------------------------

    chapi_string_option_value_t & operator[] (int idx);

protected:
private:
};

class __chapi_lib__ chapi_string_option_value_t
    : public chapi_cfg_option_value_t {
public:

    //----------------------------------------------------------------------
    chapi_string_option_value_t(chapi_cfg_option_t *option, int opt_val_idx,
        void *opt_buffer);
    virtual ~chapi_string_option_value_t();

    //-------------------------------------------------------------------------

    operator char* ();

    bool set(const char * val);

protected:
private:
};


//-----------------------------------------------------------------------------
// Some general functions
//

//
// Abstract:
//
//   Atomically replace value and return previous.
//
// Description:
//
// Parameters:
//
// Returns:
//

#if defined(_WIN32)
// warning C4035: 'interlocked_xchg' : no return value
#pragma warning (disable:4035)
#endif // defined(_WIN32)

#if defined(_MSC_VER) && defined(_M_IX86)
// warning C4311: 'type cast' : pointer truncation from 'void *' to 'long'
// warning C4312: 'type cast' : conversion from 'LONG' to 'void *' of greater
size
#pragma warning (disable:4311 4312)
#endif

inline int interlocked_xchg(int *a, int b)
{
#if defined(_MSC_VER)

    return (int)InterlockedExchange((long volatile *)a, (long)b);

#elif defined(__DECCXX)

#if (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)
#pragma __required_pointer_size __save
#pragma __required_pointer_size __long
#endif // (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)

    return (dword_t)
      __ATOMIC_EXCH_LONG
          ((void volatile *)a, (long)b);
```

```
#if (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)
#pragma __required_pointer_size __restore
#endif // (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)

#else // if !defined(__DECCXX)
#error interlocked_xchg(dword_lp_t, dword_t) not implemented
#endif // defined(__DECCXX)
};

inline
void interlocked_or
    (udword_lp_t a, udword_t b)
{
#if defined(_MSC_VER) && defined(_M_IX86)

    __asm {
      mov         ebx, a
      mov         eax, b
      lock or           [ebx], eax
    }
    // there is no InterlockedOr() procedure available

#elif defined(_MSC_VER) && defined(_M_AMD64)

    InterlockedOr
      ((long volatile *)a, (long)b);

#elif defined(__DECCXX)

#if (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)
#pragma __required_pointer_size __save
#pragma __required_pointer_size __long
#endif // (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)

    __ATOMIC_OR_LONG
      ((void volatile *)a, (long)b);

#if (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)
#pragma __required_pointer_size __restore
#endif // (defined(__INITIAL_POINTER_SIZE) && __INITIAL_POINTER_SIZE)

#else // if !defined(__DECCXX)
#error "interlocked_or(udword_lp_t, udword_t) not implemented"
#endif // defined(__DECCXX)
};

inline int log2(unsigned int x)
{
#if defined(_M_IX86) || defined(_M_AMD64)

    int v = 0;
    if (!(x & 0x0000FFFF)) {
        v += 16;
    }
    if (!(x & 0x00FF00FF)) {
        v += 8;
    }
    if (!(x & 0x0F0F0F0F)) {
        v += 4;
    }
    if (!(x & 0x33333333)) {
        v += 2;
    }
    if (!(x & 0x55555555)) {
        v += 1;
    }
    return v;

#else // unknown architecture

    // least_bit_set_decoder_32[(2 ** i) % 37] = i; i = 0..31
    const int least_bit_set_decoder_32[37] = {
      -1,   //  0: there is no i for which (2 ** i) % 37 = 0 is true
       0,   //  1: (2 ** 0) % 37 = 1, therefore 0
       1,   //  2: (2 ** 1) % 37 = 2, therefore 1
      26,   //  3: (2 ** 26) % 37 = 3, therefore 26
       2,   //  4: (2 ** 2) % 37 = 4, therefore 2
      23,   //  5: (2 ** 23) % 37 = 5, therefore 23
      27,   //  6: (2 ** 27) % 37 = 6, therefore 27
      -1,   //  7: there is no i for which (2 ** i) % 37 = 7 is true
       3,   //  8: (2 ** 3) % 37 = 8, therefore 3
      16,   //  9: (2 ** 16) % 37 = 9, therefore 16
      24,   // 10: (2 ** 24) % 37 = 10, therefore 24
      30,   // 11: (2 ** 30) % 37 = 11, therefore 30
      28,   // 12: (2 ** 28) % 37 = 12, therefore 28
      11,   // 13: (2 ** 11) % 37 = 13, therefore 11
      -1,   // 14: there is no i for which (2 ** i) % 37 = 14 is true
      13,   // 15: (2 ** 13) % 37 = 15, therefore 13
       4,   // 16: (2 ** 4) % 37 = 16, therefore 4
       7,   // 17: (2 ** 7) % 37 = 17, therefore 7
      17,   // 18: (2 ** 17) % 37 = 18, therefore 17
      -1,   // 19: there is no i for which (2 ** i) % 37 = 19 is true
      25,   // 20: (2 ** 25) % 37 = 20, therefore 25
      22,   // 21: (2 ** 22) % 37 = 21, therefore 22
      31,   // 22: (2 ** 31) % 37 = 22, therefore 31
      15,   // 23: (2 ** 15) % 37 = 23, therefore 15
      29,   // 24: (2 ** 29) % 37 = 24, therefore 29
      10,   // 25: (2 ** 10) % 37 = 25, therefore 10
      12,   // 26: (2 ** 12) % 37 = 26, therefore 12
       6,   // 27: (2 ** 6) % 37 = 27, therefore 6
      -1,   // 28: there is no i for which (2 ** i) % 37 = 28 is true
      21,   // 29: (2 ** 21) % 37 = 29, therefore 21
      14,   // 30: (2 ** 14) % 37 = 30, therefore 14
       9,   // 31: (2 ** 9) % 37 = 31, therefore 9
       5,   // 32: (2 ** 5) % 37 = 32, therefore 5
      20,   // 33: (2 ** 20) % 37 = 33, therefore 20
       8,   // 34: (2 ** 8) % 37 = 34, therefore 8
      19,   // 35: (2 ** 19) % 37 = 35, therefore 19
      18    // 36: (2 ** 18) % 37 = 36, therefore 18
    };

    return least_bit_set_decoder_32[(x) % 37];

#endif // unknown architecture
};

//
//
// Abstract:
```

```
//
//   Returns the number of the least "1" in the bit pattern.
//
// Description:
//
// Parameters:
//
//   x [in]
//
//     - a bit pattern;
//
// Returns:
//
//   As specified in Abstract clause the returning value is in the range
//   0..31 or 0..63 respectively.
//

inline int least_bit_set(unsigned int x)
{
    //
    // Generic implementation of log2() needs the argument be a power of
    // two, therefore we perform some bit manipulations.
    //
    return log2(((x & (x - 1)) ^ x));
};


//--------------------------------------------------------//
// CHAPI Task                                             //
//--------------------------------------------------------//

class __chapi_lib__ chapi_task;

typedef unsigned int  chapi_timeout_t;
typedef unsigned long chapi_proc_t;
typedef HANDLE        chapi_handle_t;
typedef HANDLE        chapi_event_t;

const chapi_handle_t chapi_invalid_handle = INVALID_HANDLE_VALUE;

//--------------------------------------------------------//
// Task utility defition                                  //
//--------------------------------------------------------//
class __chapi_lib__ chapi_task
{
protected:

    //--------------------------------------------------------//
    // Constructots and Destructors                           //
    //--------------------------------------------------------//

    chapi_task();

    virtual ~chapi_task();

public:

    //--------------------------------------------------------//
    // Task control functions                                 //
    //--------------------------------------------------------//
    //
    // Create a task
    //
    static chapi_handle_t create(
        chapi_proc_t (__stdcall * task_body)(void * task_arg),
        void * task_arg);

    //
    // Provides a delay for the caller. To be removed.
    //
    static void sleep(chapi_timeout_t delay_msec);

    //
    // Waits for the task to complete, and returns the task's completion
    // status.
    //
    static chapi_proc_t wait_for_result(chapi_handle_t the_task);
};


//--------------------------------------------------------//
// Task starter utility declaration                       //
//--------------------------------------------------------//
template <class T>
class chapi_task_starter
{
private:

    T *instance;

    chapi_proc_t (T::* task_body)(void * task_arg);

    void *task_arg;

protected:

    chapi_task_starter(T * inst, chapi_proc_t (T::* tb)(void * arg), void * arg)
        : instance(inst), task_body(tb), task_arg(arg)
    {
        return;
    }

    chapi_proc_t run()
    {
        chapi_proc_t result;

        result = (instance->*task_body)(task_arg);

        return result;
    }

    static chapi_proc_t __stdcall task_starter_func(void * arg)
    {
        chapi_proc_t result;
        chapi_task_starter<T> *starter;

        starter = (chapi_task_starter<T> *)arg;
        result = starter->run();
        return result;
    }

public:

    static chapi_handle_t run_task(T * instance,
        chapi_proc_t (T::* task_body)(void * task_arg),
        void * task_arg = 0)
    {
        chapi_task_starter<T> *starter;
        chapi_handle_t task_handle;

        starter = new chapi_task_starter<T>(instance, task_body, task_arg);
        task_handle =
chapi_task::create(&chapi_task_starter<T>::task_starter_func, starter);

        return task_handle;
    }
};


//--------------------------------------------------------//
// CHAPI critical section object                          //
//--------------------------------------------------------//

//--------------------------------------------------------//
// Abstract critical section object                       //
//--------------------------------------------------------//
class __chapi_lib__ chapi_critical_section_interface
{
public:

    //
    // Abstract:
    //
    //   Try to enter the critical section.
    //
    // Returns:
    //
    //         true if is successfully enetred to critical section
    //         false if another thread already enetred to critical section
    //
    virtual bool try_enter() = 0;

    //
    // Abstract:
    //
    //   Enter the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void enter() = 0;

    //
    // Abstract:
    //
    //   Leave the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void leave() = 0;
};


//--------------------------------------------------------//
// Empty critical section object                          //
//--------------------------------------------------------//
class __chapi_lib__ chapi_stub_critical_section : public
chapi_critical_section_interface
{
public:
    //
    // Abstract:
    //
    //   Try to enter the critical section.
    //
    // Returns:
    //
    //         true if is successfully enetred to critical section
    //         false if another thread already enetred to critical section
    //
    virtual bool try_enter();

    //
    // Abstract:
    //
    //   Enter the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void enter();

    //
    // Abstract:
    //
    //   Leave the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void leave();
};

//--------------------------------------------------------//
// Mutex object                                           //
//--------------------------------------------------------//
class __chapi_lib__ chapi_mutex : public chapi_critical_section_interface
{
protected:

    CRITICAL_SECTION section;

public:
    chapi_mutex();

    ~chapi_mutex();

    //
    // Abstract:
    //
    //   Try to enter the critical section.
    //
    // Returns:
    //
    //         true if is successfully enetred to critical section
    //         false if another thread already enetred to critical section
    //
    virtual bool try_enter();

    //
    // Abstract:
    //
    //   Enter the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void enter();

    //
    // Abstract:
    //
    //   Leave the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void leave();
};

//--------------------------------------------------------//
// Spinlock object                                        //
//--------------------------------------------------------//
class __chapi_lib__ chapi_spinlock : public chapi_critical_section_interface
{
protected:

    CRITICAL_SECTION section;

public:

    chapi_spinlock();

    ~chapi_spinlock();

    //
    // Abstract:
    //
    //   Try to enter the critical section.
    //
    // Returns:
    //
    //         true if is successfully enetred to critical section
    //         false if another thread already enetred to critical section
    //
    virtual bool try_enter();

    //
    // Abstract:
    //
    //   Enter the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void enter();

    //
    // Abstract:
    //
    //   Leave the critical section.
    //
    // Returns:
    //
    //   None.
    //
    virtual void leave();
};


//--------------------------------------------------------//
// CHAPI Containers                                       //
//--------------------------------------------------------//

//--------------------------------------------------------//
// Ring buffer                                            //
//--------------------------------------------------------//
template<class Elem, udword_t max_buffer_size>
class chapi_ring_buffer
{
protected:

    Elem items[max_buffer_size];

    uword_t head;

    uword_t tail;

    uword_t buffer_size;

public:

    chapi_ring_buffer()
    {
        head = 0;
        tail = 0;
        buffer_size = 0;
        return;
    }
```

```
chapi_ring_buffer(const chapi_ring_buffer& rb)
{
    head = rb.head;
    tail = rb.tail;
    buffer_size = rb.buffer_size;
    std::copy(rb.items[0], rb.items[size-1], items[0]);
    return;
}

~chapi_ring_buffer()
{
    return;
}

const chapi_ring_buffer& operator =(const chapi_ring_buffer& rb)
{
    std::copy(rb.items[0], rb.items[size-1], items[0]);
    return *this;
}

void clean()
{
    head = 0;
    tail = 0;
    buffer_size = 0;
}

bool is_empty() const
{
    bool retval;

    if (buffer_size > 0)
    {
        retval = false;
    }
    else
    {
        retval = true;
    }

    return retval;
}

bool is_full() const
{
    bool retval;

    if (buffer_size < max_buffer_size)
    {
        retval = false;
    }
    else
    {
        retval = true;
    }

    return retval;
}
```

```
    uword_t size() const
    {
        return buffer_size;
    }

    void push(const Elem& elem)
    {
        if (buffer_size < max_buffer_size)
        {
            items[head] = elem;
            head = (head + 1) % max_buffer_size;
            buffer_size += 1;
        }
        return;
    }

    Elem top()
    {
        return items[tail];
    }

    void pop()
    {
        if (buffer_size > 0)
        {
            tail = (tail + 1) % max_buffer_size;
            buffer_size -= 1;
        }

        return;
    }
};


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

1.1.3. CHAPI_SERIAL_DEVICE_IFACE.H

This appendix contains listing of the **chapi_serial_device_iface.h** file containing basic definitions to be used in a new CHAPI serial line controller development.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_SERIAL_DEVICE_INTERFACE_H__)
#define __CHAPI_SERIAL_DEVICE_INTERFACE_H__

#include "chapi_serial_line_iface.h"
```

```
#include "chapi.h"


#define _MAX_SERIAL_LINE_NUMBER_ 256

// Base class for all CHAPI serial line controllers.
class chapi_serial_device_interface
{
public:

    //CHAPI start()
    virtual void start() = 0;

    //CHAPI stop()
    virtual void stop() = 0;

    //CHAPI start
    virtual void reset() = 0;

    //CHAPI read()
    virtual int read(unsigned int addr, bool is_byte) = 0;

    //CHAPI write()
    virtual void write(unsigned int addr, int val, bool is_byte) = 0;

    //CHAPI set_configuration()
    virtual int set_configuration(const char * parameters) = 0;

    //CHAPI set_configuration_ex()
    virtual int set_configuration_ex() = 0;

    // Pointer to CHARON-supplied structure
    const chapi_in *ci;

protected:

    // CHAPI serial line interface class
    // Base class for all CHAPI serial lines
    friend class chapi_serial_line_interface;

    // Array of CHAPI serial lines pointers
    chapi_serial_line_interface * line[_MAX_SERIAL_LINE_NUMBER_];


    // Callback function, serial line must call serial line controller`s
callback
    // function rx_done(...) to notify controller about received data.
    //
    //  Arguments:
    //      unsigned int len      - number of received data
    //      unsigned char line_id - line number
    //
    //  Return : 0 - No errors
    virtual int rx_done(unsigned int len, unsigned char line_id) = 0;


    // Callback function, serial line must call serial line controller`s
callback
    // function tx_done(...) to notify controller about completion of send
operation.
```

```
    //
    //  Arguments:
    //      unsigned int len      - number of sent data
    //      unsigned char line_id - line number
    //
    //  Return : 0 - No errors
    virtual int tx_done(unsigned int len, unsigned char line_id) = 0;


    // Callback function, serial line must call serial line controller`s
callback
    // function get_tx_char(...) to get data for transmission from serial line
    // controller`s TX buffer.
    //
    //  Arguments:
    //      unsigned char & from_buf - place to put char for transmission
    //      unsigned char line_id - line number
    //
    //  Return : 1 - No errors, you have got 1 char for transmission
    //          -1 - No char for trasmission, serial line controller`s TX buffer
is empty
    //
    virtual int get_tx_char(unsigned char & from_buf, unsigned char line_id) =
0;


    // Callback function, serial line must call serial line controller`s
callback
    // function input_signal(...) to notify controller about new input signals
(modem).
    //
    //  Arguments:
    //      unsigned char in_signal - input signals
    //      unsigned char line_id   - line number
    //
    //  Return : 0 - No errors
    //
    virtual int input_signal(unsigned char in_signal, unsigned char line_id) =
0;


    // Callback function, serial line must call serial line controller`s
callback
    // function error_tx(...) to notify controller about transmission errors.
    //
    //  Arguments:
    //      unsigned char error   - transmission error
    //      unsigned char line_id - line number
    //
    //  Return : 0 - No errors
    //
    virtual int error_tx(unsigned char error, unsigned char line_id) = 0;


    // Callback function, serial line must call serial line controller`s
callback
    // function error_rx(...) to notify controller about receive errors.
    //
    //  Arguments:
    //      unsigned char error   - receive error
    //      unsigned char line_id - line number
    //
    //  Return : 0 - No errors
    //
    virtual int error_rx(unsigned char error, unsigned char line_id) = 0;


    // Callback function, serial line must call serial line controller`s
callback
    // function extra(...) for extra features.
    //
    //  Arguments:
    //      void * extra          - void pointer
    //      int arg               - integer argument
    //      unsigned char line_id - line number
    //
    //  Return : 0 - No errors
    //
    virtual int extra(void * extra, int arg, unsigned char line_id) = 0;


    // System error logging,
    // also callback function, serial line must call serial line controller`s
callback
    // get_sys_err(...) for system error logging.
    //
    //  Arguments:
    //     unsigned long err - system error number
    //
    //
    virtual void get_sys_err( unsigned long err ) = 0;


    // Message logging,
    // also callback function, serial line must call serial line controller`s
callback
    // log_msg(...) for message logging.
    //
    //  Arguments:
    //     log_message_type_t msg_type  - type of meesage
    //     log_message_id_t msg_code    - message code
    //     const char *file             - file from where the message is logged
    //     int line                     - line from where the message is logged
    //     const char *str              - message
    //
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *str)
    {
        if(ci != 0 && ci->log_message_ex != 0) {
            ci->log_message_ex(ci, msg_type, file, line, msg_code, str);
        }
    };


    // Trace logging,
    // also callback function, serial line must call serial line controller`s
callback
    // debug_trace(...) for trace logging.
    //
    //  Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    //
    void debug_trace(unsigned char debug_level, const char *str)
    {
        if(ci != 0 && ci->debug_trace != 0) {
            ci->debug_trace(ci, debug_level, str);
        }
    };

};

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

1.1.4. CHAPI_SERIAL_LINE_IFACE.H

This appendix contains listing of the **chapi_serial_line_iface.h** file which contains basic definitions to be used in development of new CHAPI serial line port.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_SERIAL_LINE_INTERFACE_H__)
#define __CHAPI_SERIAL_LINE_INTERFACE_H__


#if defined(IN_CHAPI_SERIAL_DLL)
#define __chapi_serial_lib__ __declspec(dllexport)
#endif // defined(IN_CHAPI_SERIAL_DLL)

#if defined(USE_CHAPI_SERIAL_DLL)
#define __chapi_serial_lib__ __declspec(dllimport)
#endif // defined(USE_CHAPI_SERIAL_DLL)

#if !defined(__chapi_serial_lib__)
#define __chapi_serial_lib__
#endif // !defined(__chapi_serial_lib__)

// Modem signals definitions
#define R_T_S (0x01 << 0)
#define D_T_R (0x01 << 1)
#define D_C_D (0x01 << 2)
#define C_T_S (0x01 << 3)
#define D_S_R (0x01 << 4)
#define R_I_N_G (0x01 << 5)

// RX errors definitions
#define LINE_ERROR_BREAK (0x01 << 0)
```

```
#define LINE_ERROR_OVER (0x01 << 1)
#define LINE_ERROR_FRAMING (0x01 << 2)
#define LINE_ERROR_PARITY (0x01 << 3)

// Break controls definitions
#define START_BREAK   1
#define STOP_BREAK    0

// Flow control definitions
#define NONE_FLOW_CONTROL 0x00
#define XON_XOFF_FLOW_CONTROL (0x01 << 0)
#define DTR_DSR_FLOW_CONTROL (0x01 << 1)
#define RTS_CTS_FLOW_CONTROL (0x01 << 2)

// Parity control definitions
#define PARITY_CONTROL_NONE 0x00
#define PARITY_CONTROL_EVEN (0x01 << 0)
#define PARITY_CONTROL_ODD (0x01 << 1)
#define PARITY_CONTROL_MARK (0x01 << 2)
#define PARITY_CONTROL_SPACE (0x01 << 3)

// Extra features definition
#define EXTRA_LINE_SETUP (0x01 << 0)

// Logging definitions
#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE


 // Message logging ...
#define _ERR_MSG_ error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ info_msg_type, __FILE__, __LINE__

#define LOGMSG(x) log_msg x;


// Debug trace ...
#define L(n) n
#define TRACE(x) debug_trace x;


#include "chapi_serial_device_iface.h"
#include <stdio.h>
#include <stdarg.h>


// CHAPI serial line controller interface
class chapi_serial_device_interface;

// CHAPI serial line interface
class __chapi_serial_lib__ chapi_serial_line_interface
{
public:

    // Constructor of CHAPI serial line instance
```

```
//
// Arguments :
//      void * ctrl           - serial line controller instance (pointer)
//      unsigned char line_id - current serial line instance number
//
chapi_serial_line_interface(void * ctrl, unsigned char line_id)
{
    this->ctrl = (chapi_serial_device_interface *) ctrl;
    this->line_id = line_id;
}


// Start serial line.
//
//  Return : 0 - No errors
//
virtual int start() = 0;


// Stop serial line.
//
//  Return :
//      0 - No errors
//
virtual int stop() = 0;


// Setup serial line baud rate
//
//  Arguments :
//      unsigned int speed - baud rate
//
//  Return :
//      0 - No errors
//
virtual int setup_speed(unsigned int speed) = 0;


// Setup serial line char bits length
//
//  Arguments :
//      unsigned char char_len - character bits length
//
//  Return :
//      0 - No errors
//
virtual int setup_char_len(unsigned char char_len) = 0;


// Setup serial line stop bits length
//
//  Arguments :
//      unsigned char stop_len -  stop bits length
//
//  Return :
//      0 - No errors
//
virtual int setup_stop_len(unsigned char stop_len) = 0;
```

```
// Setup serial line parity control
//
//  Arguments :
//      unsigned char parity - parity control
//
//  Return :
//      0 - No errors
//
virtual int setup_parity(unsigned char parity) = 0;


// Setup serial line flow control
//
//  Arguments :
//      unsigned char flow_ctrl - flow control
//
//  Return :
//      0 - No errors
//
virtual int setup_flow_ctrl(unsigned char flow_ctrl) = 0;


// Send special control character, w/o respond, ahead of any pending
// data in the output buffer
//
//  Arguments :
//      unsigned char ctrl_char - control character
//
//  Return :
//      0 - No errors
//
virtual int send_ctrl_char(unsigned char ctrl_char) = 0;


// Setup serial line output signals (modems)
//
//  Arguments :
//      unsigned char out_signal - signals
//
//  Return :
//      0 - No errors
//
virtual int setup_output_signal(unsigned char out_signal) = 0;


// Setup serial line break transmission
//
//  Arguments :
//      unsigned char break_signal - start/stop break
//
//  Return :
//      0 - No errors
//
virtual int setup_break(unsigned char break_signal) = 0;


// Initiate transmission
//
//  Arguments :
//      unsigned int len - number of bytes for transmission,
//                          or 0 for transmission all bytes from
//                          serial line controller TX buffer
//
//  Return :
//      0 - No errors
//
virtual int do_tx(unsigned int len) = 0;



// Enable transmission
//
//  Return :
//      0 - No errors
//
virtual int set_tx_enable() = 0;



// Disable transmission
//
//  Return :
//      0 - No errors
//
virtual int set_tx_disable() = 0;



// Enable receive
//
//  Return :
//      0 - No errors
//
virtual int set_rx_enable() = 0;


// Disable transmission
//
//  Return :
//      0 - No errors
//
virtual int set_rx_disable() = 0;


// Set serial line`s extra features.
//
//  Arguments:
//      void * extra          - void pointer
//      int arg               - integer argument
//
//  Return : 0 - No errors
//
virtual int set_extra_command(void * extra_command, int arg) = 0;


// Get received data from serial line RX buffer.
//
//  Arguments:
//      unsigned char & from_buf - place to put received char
//
```

```
    //  Return : 1 - No errors, you have got 1 received char
    //          -1 - No received char, serial line RX buffer is empty
    //
    virtual int get_rx_char(unsigned char & from_buf) = 0;

protected:

    //-----------------------------------------------------------------------
    // Message logging / debugging routines
    //

    //Instance of serial line controller (pointer)
    chapi_serial_device_interface * ctrl;

    // Serial line number
    unsigned char line_id;


    // Wrapper for callback function, serial line must call function
    // ctrl_rx_done(...) to notify controller about received data.
    //
    //  Arguments:
    //      unsigned int len      - number of received data
    //
    //  Return : 0 - No errors
    int ctrl_rx_done(unsigned int len);

    // Wrapper for callback function, serial line must call function
    // ctrl_tx_done(...) to notify controller about completion of send
operation.
    //
    //  Arguments:
    //      unsigned int len      - number of sent data
    //
    //  Return : 0 - No errors
    int ctrl_tx_done(unsigned int len);


    // Wrapper for callback function, serial line must call function
    // ctrl_get_tx_char(...) to get data for transmission from serial line
    // controller`s TX buffer.
    //
    //  Arguments:
    //      unsigned char & from_buf - place to put char for transmission
    //
    //  Return : 1 - No errors, you have got 1 char for transmission
    //          -1 - No char for trasmission, serial line controller`s TX buffer
is empty
    //
    int ctrl_get_tx_char(unsigned char & from_buf);


    // Wrapper for callback function, serial line must call function
    // ctrl_input_signal(...) to notify controller about new input signals
(modem).
    //
    //  Arguments:
    //      unsigned char in_signal - input signals
    //
    //  Return : 0 - No errors
```

```
//
int ctrl_input_signal(unsigned char in_signal);


// Wrapper for callback function, serial line must call function
// ctrl_error_tx(...) to notify controller about transmission errors.
//
//  Arguments:
//      unsigned char error   - transmission error
//
//  Return : 0 - No errors
//
int ctrl_error_tx(unsigned char error);


// Wrapper for callback function, serial line must call function
// ctrl_error_rx(...) to notify controller about receive errors.
//
//  Arguments:
//      unsigned char error   - receive error
//      unsigned char line_id - line number
//
//  Return : 0 - No errors
//
int ctrl_error_rx(unsigned char error);


// Wrapper for callback function, serial line must call function
// ctrl_extra(...) for serial line controller`s extra features.
//
//  Arguments:
//      void * extra          - void pointer
//      int arg               - integer argument
//
//  Return : 0 - No errors
//
int ctrl_extra(void * extra, int arg);


// Message logging,
//
//  Arguments:
//     log_message_type_t msg_type  - type of meesage
//     const char *file            - the file from where message is logged
//     int line                    - the line from where mesage is logged
//     log_message_id_t msg_code    - message code
//     const char *str              - message
//
void log_msg(log_message_type_t msg_type, const char *file, int line,
    log_message_id_t msg_code, const char *fmt, ...);



// Trace logging,
//
//  Arguments:
//      unsigned char debug_level - debug level
//      const char *str - trace message
//
void debug_trace(unsigned char debug_level, const char *fmt, ...);
```

```
    // System error logging,
    //
    //  Arguments:
    //     unsigned long err - system error number
    //
    //
    void get_sys_err( unsigned long err );

};



typedef bool (*pf_init_line_t)(void * ctrl,
    chapi_serial_line_interface ** p_serial_l_i, unsigned char line_id);

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

1.1.5. CHAPI_ADAPTER.H

This appendix contains source code listing of the **chapi_adapter.h** file which contains basic definitions to be used in development of new CHAPI adapter for hardware replacement boards.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_ADAPTERS_H__)
#define __CHAPI_ADAPTERS_H__

#if defined(IN_CHAPI_HW_DLL)
#define __chapi_adapters__ __declspec(dllexport)
#endif // defined(IN_CHAPI_HW_DLL)

#if defined(USE_CHAPI_HW_DLL)
#define __chapi_adapters__ __declspec(dllimport)
#endif // defined(USE_CHAPI_HW_DLL)

#if !defined(__chapi_adapters__)
#define __chapi_adapters__
#endif // !defined(__chapi_adapters__)


// PCI local bus register set definitions
#include "pci_reg.h"

// CHAPI protocol definitions
#include "chapi.h"
```

```
// template l2lists<>
#include "chapi_lib.h"


// Host page size relative stuff
#if defined(_M_IX86) || defined(_M_AMD64)
#define HOST_PAGE_SIZE        (4*1024)
#endif // defined(_M_IX86) || defined(_M_AMD64)

#define HOST_PAGE_NO(o)       ((o)/HOST_PAGE_SIZE)
#define HOST_PAGE_OFFSET(o)   ((o)%HOST_PAGE_SIZE)
#define ROUND_TO_HOST_PAGE(s) (((s)+HOST_PAGE_SIZE-1)&(~(HOST_PAGE_SIZE-1)))
#define TRUNCATE_TO_HOST_PAGE(s) ((s)&(~(HOST_PAGE_SIZE-1)))


// CPU description structure - used for IRQ processing
typedef struct _cpu_dsc_t {
    unsigned long volatile *attention_object;
    unsigned long           attention_value;

    unsigned long volatile *brq_object;
    unsigned long           brq_mask;
} cpu_dsc_t;


// Bus request description structure
#define BUS_SERVER_MAP_LENGTH   (8 * sizeof(unsigned int))
typedef struct _brq_dsc_t {
    unsigned long   level;
             long   cpu_no;
    cpu_dsc_t       cpu_dsc[BUS_SERVER_MAP_LENGTH];
} brq_dsc_t;


// Adapter description structure
typedef struct _adapter_hdr_t {
    int             spin_lock;
    unsigned int    brq_num;
} adapter_hdr_t;


//---------------------------------------------------------------------------
// Register access macroses to be used in adapter's implementation
//
#if defined(_X86_)
#define READ_REGISTER_UCHAR(r)      (*(volatile UCHAR *)(r))
#define READ_REGISTER_USHORT(r)     (*(volatile USHORT *)(r))
#define READ_REGISTER_LONG(r)       (*(volatile LONG *)(r))
#define READ_REGISTER_ULONG(r)      (*(volatile ULONG *)(r))
#define WRITE_REGISTER_UCHAR(r, v)  (*(volatile UCHAR *)(r) = (v))
#define WRITE_REGISTER_USHORT(r, v) (*(volatile USHORT *)(r) = (v))
#define WRITE_REGISTER_ULONG(r, v)  (*(volatile ULONG *)(r) = (v))
#endif


// Base class for hardware adapter's support
class __chapi_adapters__ chapi_adapter_t {
public:

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

```
chapi_adapter_t(const chapi_in *_ci);
virtual ~chapi_adapter_t();

//-------------------------------------------------------------------------

//
// General interface to any kind of hardware adapter connected to PCI bus
// of the host system.
//

//
// Abstract:
//
//  Check if adapter is valid. We assume that adapter is valid when it is
//  connected to device driver and has valid descriptor.
//
// Returns:
//
//  true if adapter is valid, false otherwise.
//
virtual bool valid() {
    return (h != INVALID_HANDLE_VALUE) && (adapter_dsc != 0);
}

//
// Abstract:
//
//  Connect adapter with specified instance number. Map required
//  register spaces and do other pre-run work here. This method is adapter
//  resource allocator.
//
// Arguments:
//
//  adapter_no     -   adapter instance number to use.
//
//  \\.\\<adapter_name><adapter_no> is a physical device name to open for
//  communication with device driver.
//
// Returns:
//
//  Operation status.
//
virtual bool assign(int adapter_num);

//
// Abstract:
//
//  Free allocated resources (unmap register spaces, etc..) and close
//  device handle.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  None.
//
virtual void release();
```

```
    //
    // Abstract:
    //
    //  This is a place to initialize adapter (tune registers content,
    //  attach IRQ, etc...).
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool attach();

    //
    // Abstract:
    //
    //  Stop adapter activity (detach IRQ, etc...).
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    virtual void detach();

    //
    // Abstract:
    //
    //  Reset adapter hardware.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Operation result.
    //
    virtual bool reset() = 0;

    //
    // Abstract:
    //
    //  Configure adapter hardware. This method is called by device controlling
    //  hardware adapter when it is necessary to pass configuration string to
us.
    //  It is called each time, parameters are changed in configuration file.
    //
    // Arguments:
    //
    //  options - adapter options string (can contain any text adapter is
    //            waiting for).
    //
```

```
// Returns:
//
//  None.
//
virtual void configure(const char *options) = 0;

//
// Abstract:
//
//  Translate specified offset within the mapped virtual memory into
//  the host physical address.
//
// Arguments:
//
//  addr    -   bus address to get ram one for;
//  len     -   length in bytes to translate;
//  buf     -   reference to store translated address there.
//
// Returns:
//
//  The size of contigous ram region.
//
virtual unsigned int translate_for_dma(udword_t addr,
    unsigned int len, char *& buf);

//
// Abstract:
//
//  Translate specified offset within the mapped virtual memory into
//  the host physical address.
//
// Arguments:
//
//  addr    -   virtual address to get physical one for;
//  phys_addr   -   reference where host physical address will be stored.
//
// Returns:
//
//  The size of contigous physical region.
//
virtual unsigned int get_phys_chunk(char *addr, unsigned int& phys_addr);

//
// Abstract:
//
//  Translate specified ram address to the host physical address.
//
// Arguments:
//
//  addr        -   primary ram address;
//  phys_addr   -   reference where host physical address will be stored.
//
// Returns:
//
//  The size of contigous physical region.
//
virtual unsigned int get_phys_chunk(unsigned int addr,
    unsigned int& phys_addr);

//
```

```
// Abstract:
//
//  Map specified memory segment for DMA. It is actual for any kind of DMA
//  replacement hardware.
//
// Arguments:
//
//  addr        -   start ram address of segment;
//  seg_base    -   base virtual address of segment;
//  seg_size    -   size of segment to map.
//
// Returns:
//
//  Operation status.
//
virtual bool map_ram_segment_for_dma(unsigned int addr,
    char *seg_base, unsigned int seg_size);

//
// Abstract:
//
//  Unmap specified memory segment mapped for DMA. It is actual for any kind
//  of DMA replacement hardware.
//
// Arguments:
//
//  addr    -   start ram address of segment.
//
// Returns:
//
//  None.
//
virtual void unmap_ram_segment(unsigned int addr);

//
// Abstract:
//
//  Map the whole emulator memory for DMA. It is actual for any kind of DMA
//  replacement hardware.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  Operation status.
//
virtual bool map_emulator_memory_for_dma();

//
// Abstract:
//
//  Unmap emulator memory mapped for DMA. It is actual for any kind of DMA
//  replacement hardware.
//
// Arguments:
//
//  None.
//
```

```
    // Returns:
    //
    //  Operation status.
    //
    virtual void unmap_emulator_memory();

    //
    // Abstract:
    //
    //  Setup BRQ descriptor to be passed to the driver with the information
    //  about CPUs and BRQs. This information will be used by the driver in
order to
    //  interrupt CPU efficiently.
    //
    // Arguments:
    //
    //  brq_dsc -   BRQ descriptor to setup;
    //  brq     -   chapi BRQ object which parameters should be described;
    //  level   -   BRQ level
    //
    // Returns:
    //
    //  None.
    //
    virtual void setup_brq_dsc(brq_dsc_t &brq_dsc, chapi_brq_t &brq, int level);

    //
    // Abstract:
    //
    //  Setup affinity mask for specified BRQ level.
    //
    // Arguments:
    //
    //  level       -   BRQ level;
    //  cpu_no      -   CPU affinity mask.
    //
    // Returns:
    //
    //  None.
    //
    virtual void setup_brq_dsc_affinity(int level, int cpu_no) = 0;

    //
    // Abstract:
    //
    //  Check if adapter allows interrupting.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Check result.
    //
    virtual bool is_interrupt_enabled() = 0;

    //
    // Abstract:
    //
    //  Set specified bus request active.
    //
    // Arguments:
    //
    //  brq_no  -   BRQ number to activate;
    //
    // Returns:
    //
    //  None.
    //
    virtual void put_brq(word_t brq_no);

    //
    // Abstract:
    //
    //  Get pointer to the brq_dsc_t structure describing specified BRQ.
    //
    // Arguments:
    //
    //  brq_no  -   BRQ number to get descriptor for;
    //
    // Returns:
    //
    //  Pointer to the brq_dsc_t describing specified BRQ.
    //
    virtual brq_dsc_t *get_brq_dsc(word_t brq_no = 0) = 0;

    //
    // Abstract:
    //
    //  Setup adapter descriptor. This virtual function is called before BRQ
    //  mapping in order to setup descriptor to be used in device driver.
    //  Descriptor is hardware specific but it always has predefined header.
    //  This routine should be implemented for particular adapter.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  None.
    //
    virtual void setup_adapter_dsc() = 0;


protected:


    // Definition for DMA mappings.
    class adapter_dma_dsc_t : public l2list<adapter_dma_dsc_t> {
    public:

        //--------------------------------------------------------------------

        adapter_dma_dsc_t(chapi_adapter_t *_adapter, adapter_dma_dsc_t
**entrance);
        ~adapter_dma_dsc_t();

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

```
//
// Abstract:
//
//  Check if specified virtual address is mapped by this descriptor.
//
// Arguments:
//
//  addr   -   virtual address to check for.
//
// Returns:
//
//  true    -   if specified address is mapped by this descriptor,
//  false   -   otherwise.
//
bool contains(char *addr);

//
// Abstract:
//
//  Check if specified ram address is mapped by this descriptor.
//
// Arguments:
//
//  addr   -   ram address to check for.
//
// Returns:
//
//  true    -   if specified address is mapped by this descriptor,
//  false   -   otherwise.
//
bool contains(unsigned int addr);

//
// Abstract:
//
//  Map specified piece of virtual memory to host physical memory using
//  device driver. Mapping is done only in the case if there is no
//  existent mapping in the list yet.
//
// Arguments:
//
//  addr        -   start ram address to map;
//  mem_start   -   start virtual address to map;
//  mem_size    -   the size of block in bytes.
//
// Returns:
//
//  Operation status.
//
bool map_for_dma(unsigned int addr, char *mem_start,
    unsigned int mem_size);

//
// Abstract:
//
//  Unmap specified piece of virtual memory.
//
// Arguments:
//
```

```
    //  None.
    //
    // Returns:
    //
    //  Operation status.
    //
    bool unmap_for_dma();

    //
    // Abstract:
    //
    //  Get physical address for specified virtual one and the size of
    //  contigous host physical memory area.
    //
    // Arguments:
    //
    //  addr        -   virtual address to get physical one for;
    //  phys_addr   -   location where to store physical address.
    //
    // Returns:
    //
    //  0   -   operation failure;
    //  > 0 -   the size of contigous host physical memory area.
    //
    size_t get_phys_chunk(char *addr, unsigned int & phys_addr);

    //
    // Abstract:
    //
    //  Get physical address for specified bus address and the size of
    //  contigous host physical memory area.
    //
    // Arguments:
    //
    //  addr        -   ram address to get physical one for;
    //  phys_addr   -   location where to store physical address.
    //
    // Returns:
    //
    //  0   -   operation failure;
    //  > 0 -   the size of contigous host physical memory area.
    //
    size_t get_phys_chunk(unsigned int addr, unsigned int & phys_addr);


    //---------------------------------------------------------------------


protected:

    // chapi_adapter_t instance we are belong to ...
    chapi_adapter_t *adapter;

    // chapi_in retreived from adapter for logging facility
    const chapi_in *ci;

    // Start ram address of the mapped block
    unsigned int    addr_start;

    // Start and length of the mapped block (in terms of virtual address)
    char            *mem_start;
    unsigned int    mem_size;

    //
    // This is a host physical addresses mapping for the mentioned above
    // virtual memory block. This mapping is always done via device driver
    // this adapter is wotking with.
    //
    unsigned int    *pci_address_map;
    size_t          pci_address_map_entries;

};


//---------------------------------------------------------------------------
// This is a standard interface to device driver.The set of predefined IOCTL
// device codes is used here to perform described operations. It is assumed
// that used device driver supports required IOCTLs.
//
// Additional calls to support hardware device driver specific IOCTLs should
// be defined in derived classes.
//

//
// Abstract:
//
//  Ask device driver about hardware attributes.
//
// Arguments:
//
//  va  -   pointer to device driver specific block of information;
//  len -   the size of supplied block.
//
// Returns:
//
//  Operation status.
//
virtual bool get_attributes(void *va, size_t len);

//
// Abstract:
//
//  Ask device driver to set hardware attributes.
//
// Arguments:
//
//  va  -   pointer to device driver specific block of information;
//  len -   the size of supplied block.
//
// Returns:
//
//  Operation status.
//
virtual bool set_attributes(void *va, size_t len);

//
// Abstract:
//
//  Ask device driver to map specified registers address space.
//
```

```
// Arguments:
//
//  n   -   the number of space to map (0 - PLX, device specific 1,
//              device specific 2, etc ...)
//  pa  -   location where to store mapped region address;
//  pl  -   location where to store mapped region size.
//
// Returns:
//
//  Operation status.
//
virtual bool map_register_space(int n, void **pa, size_t *pl);

//
// Abstract:
//
//  Ask device driver to unmap specified registers address space.
//
// Arguments:
//
//  n   -   the number of space to map (0 - PLX, device specific 1,
//              device specific 2, etc ...)
//  a   -   mapped registers address space;
//  l   -   mapped registers address space length.
//
// Returns:
//
//  Operation status.
//
virtual bool unmap_register_space(unsigned int n, void *a, size_t len);

//
// Abstract:
//
//  Map specified number of bus requests for device.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  Operation status.
//
virtual bool map_adapter_brq();

//
// Abstract:
//
//  Unmap specified number of device's bus requests.
//
// Arguments:
//
//  None.
//
// Returns:
//
//  Operation status.
//
virtual bool unmap_adapter_brq();
```

```
    //
    // Abstract:
    //
    //  Map specified memory region into the physical memory.
    //  Mapping is described by pci_address_map and consists
    //  od a number of contigous regions.
    //
    // Arguments:
    //
    //  a   -   virtual addres where mapped region is started;
    //  len -   length in bytes of mapped region;
    //  map -   pointer to the array where created mapping can be stored;
    //  map_len -   the number of entries in the specified above map.
    //
    // Returns:
    //
    //  Success indicator.
    //
    virtual bool map_virtual_memory_chunk(void *a, size_t len, void *map,
        size_t map_len);

    //
    // Abstract:
    //
    //  Unmap specified memory region, previously mapped by
    //  map_virtual_memory_chunk().
    //
    // Arguments:
    //
    //  a   -   virtual addres where mapped region is started;
    //
    // Returns:
    //
    //  Success indicator.
    //
    virtual bool unmap_virtual_memory_chunk(void *a);

    //
    // Abstract:
    //
    //  This method asks device driver to map common buffer to be used by driver
    //  itself and device emulation. Details of buffer structure and its usage
    //  are known by particular device driver and device emulation uses this
driver.
    //
    // Arguments:
    //
    //  dcb   -   pointer to device communication buffer to be used for driver<-
>
    //            emulation communication;
    //
    // Returns:
    //
    //  Success indicator.
    //
    virtual bool map_common_buffer(void *dcb);

    //
    // Abstract:
    //
    //  This method asks device driver to unmap the memory used for common
buffer.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Success indicator.
    //
    virtual bool unmap_common_buffer();


    //
    // Abstract:
    //
    //  This method should be implemented in derived classes and return adapter
name
    //  which is cerated in Windows by device driver in order to access
particular
    //  hardware adapter.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Windows device name for adapter.
    //
    virtual char * get_adapter_name() = 0;

    //------------------------------------------------------------------------

    // Handle to communicate with device driver
    HANDLE      h;

    // Mapped PCI local bus register space
    char      *plx_space;

    // ... and its size
    size_t      plx_size;

    //
    // CHAPI input context is used to communicate the bus, for message logging
    // and other needs in CHARON kernel.
    //
    // FIXME: it will be nice to replace this with pointer to some
chapi_device_t
    // implementing basic functionality of CHAPI device. This is in future
plans.
    //
    const chapi_in      *ci;

    // This stuff is used to map memory for DMA via used adpater driver
    adapter_dma_dsc_t   *adapter_dma_map;
    // This is a pointer to adapter descriptor
    adapter_hdr_t       *adapter_dsc;

    //
    // HOST memory system characteristics used during the mapping of virtual
memory
    // to the physical one. They are retreived from the system during the
adapter
    // instance creation and used by adapter_dma_dsc_t class.
    //
    DWORD               host_page_size;
    DWORD               host_page_bits;
    DWORD               host_page_mask;

private:

};

typedef bool (*pf_init_adapter_t)(void * ctrl, chapi_adapter_t **
p_chapi_adapter,
    const chapi_in *_ci, chapi_brq_t *hw_brq);


#endif  //  __CHAPI_ADAPTERS_H__
```

1.1.6. CHAPI_BUS_ADAPTER.H

This appendix contains source code listing of the **chapi_bus_adapter.h** file which contains basic definitions to be used in development of new CHAPI bus adapter.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_BUS_ADAPTER_H__)
#define __CHAPI_BUS_ADAPTER_H__


// class chapi_adapter_t
#include "chapi_adapter.h"


//
// This class adds bus adapter specific protocol methods to general chapi
adapter
// described by the class chapi_adapter_t
//
class __chapi_adapters__ chapi_bus_adapter_t
    : public chapi_adapter_t {
public:

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

```
chapi_bus_adapter_t(const chapi_in *_ci)
    : chapi_adapter_t(_ci)
    , memory_fault(false)
{}

virtual ~chapi_bus_adapter_t()
{}

//-------------------------------------------------------------------------
// chapi_adapter_t interface
//

// chapi_adapter_t::assign()
virtual bool assign(int adapter_num);

// chapi_adapter_t::release()
virtual void release();

// chapi_adapter_t::attach()
virtual bool attach();

// chapi_adapter_t::detach()
virtual void detach();

// chapi_adapter_t::reset()
virtual bool reset() = 0;

// chapi_adapter_t::configure()
virtual void configure(const char *options) = 0;

// chapi_adapter_t::translate_fpr_dma()
virtual unsigned int translate_for_dma(udword_t addr, unsigned int len,
    char *& buf);

// chapi_adapter_t::get_phys_chunk()
virtual unsigned int get_phys_chunk(char *addr, unsigned int& phys_addr);

// chapi_adapter_t::get_phys_chunk()
virtual unsigned int get_phys_chunk(unsigned int bus_address,
    unsigned int& phys_addr);

// chapi_adapter_t::map_ram_segment_for_dma()
virtual bool map_ram_segment_for_dma(unsigned int addr,
    char *seg_base, unsigned int seg_size);

// chapi_adapter_t::unmap_ram_segment()
virtual void unmap_ram_segment(unsigned int addr);

// chapi_adapter_t::map_emulator_memory_for_dma()
virtual bool map_emulator_memory_for_dma();

// chapi_adapter_t::unmap_emulator_memory()
virtual void unmap_emulator_memory();

// chapi_adapter_t::setup_brq_dsc()
virtual void setup_brq_dsc(brq_dsc_t &brq_dsc, chapi_brq_t &brq, int level);

// chapi_adapter_t::setup_brq_dsc_affinity()
virtual void setup_brq_dsc_affinity(int level, int cpu_no) = 0;
```

```
    // chapi_adapter_t::is_interrupt_enabled()
    virtual bool is_interrupt_enabled() = 0;

    // chapi_adapter_t::put_brq()
    virtual void put_brq(word_t brq_no = 0);

    // chapi_adapter_t::get_brq_dsc()
    virtual brq_dsc_t *get_brq_dsc(word_t brq_no) = 0;

    // chapi_adapter_t::setup_adapter_dsc()
    virtual void setup_adapter_dsc() = 0;


    //------------------------------------------------------------------------
    // BUS adapter specific interface.
    //

    virtual int run_interactive_command(const char *command_verb, char
*parameters) = 0;

    virtual bool has_memory_fault() {
        return memory_fault;
    }

    //
    // Abstract:
    //
    //  Read the byte from the bus at specified address.
    //
    // Arguments:
    //
    //  addr    -   address to read byte from;
    //  val     -   output parameter to store read byte in.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool read_byte(unsigned int addr, int *val) = 0;

    //
    // Abstract:
    //
    //  Read the word from the bus at specified address.
    //
    // Arguments:
    //
    //  addr    -   address to read word from;
    //  val     -   output parameter to store read word in.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool read_word(unsigned int addr, int *val) = 0;

    //
    // Abstract:
    //
    //  Write the byte to the bus at specified address.
    //
    // Arguments:
    //
    //  addr    -   address to write byte to;
    //  val     -   value to write to the bus.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool write_byte(unsigned int addr, int val) = 0;

    //
    // Abstract:
    //
    //  Write the word to the bus at specified address.
    //
    // Arguments:
    //
    //  addr    -   address to write word to;
    //  val     -   value to write to the bus.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool write_word(unsigned int addr, int val) = 0;

    //
    // Abstract:
    //
    //  Write value to Scatter/Gather register.
    //
    // Arguments:
    //
    //  reg_no  -   register number;
    //  val     -   value to write to the register.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool write_sgmr(int reg_no, int val) = 0;

    //
    // Abstract:
    //
    //  Write value to UMR.
    //
    // Arguments:
    //
    //  reg_no  -   UMR register number;
    //  val     -   value to write to the register.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool write_umr(int reg_no, int val) = 0;
    //
    // Abstract:
    //
    //  Write value to MCSR.
    //
    // Arguments:
    //
    //  val     -   value to write to the register.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool write_mcsr(int val) = 0;

    //
    // Abstract:
    //
    //  Build static SGMR mapping for emulators memory of specified size.
    //
    // Arguments:
    //
    //  memory_size -   the size of emulated system memory to build SGMR mapping
for;
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool build_sgmr_mapping(unsigned int memory_size) = 0;

    //
    // Abstract:
    //
    //  Read IRQ vector during the interrupt sequence.
    //
    // Arguments:
    //
    //  brq_level   -   BRQ level.
    //
    // Returns:
    //
    //  Interrupt vector value.
    //
    virtual int read_irq_vector(int brq_level) = 0;

    //
    // Abstract:
    //
    //  Enable DMA operations.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual bool enable_dma() = 0;

    //
    // Abstract:
    //
    //  Disable DMA operations.
    //
    // Arguments:
    //
    //  None.
    //
    // Returns:
    //
    //  Operation status.
    //
    virtual void disable_dma() = 0;


protected:

    // chapi_adapter_t::get_adapter_name()
    virtual char * get_adapter_name() = 0;

    // Memory fault flag
    bool memory_fault;

private:

};


#endif  //  __CHAPI_BUS_ADAPTER_H__
```

1.1.7. CHAPI_DISK_DEVICE_IFACE.H

This appendix contains listing of the **chapi_disk_device_iface.h** file containing basic definitions to be used in a new CHAPI disk controller development.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_DISK_DEVICE_IFACE_H__)
#define __CHAPI_DISK_DEVICE_IFACE_H__


// structure definition to contain disk geometry
struct CHAPI_DISK_DRIVE_GEOM {
    unsigned long cylinders_per_disk;
    unsigned long tracks_per_cylinder;
    unsigned long sectors_per_track;
    unsigned long bytes_per_sector;
```

```
};

// structure definition to contain current heads position
struct CHAPI_DISK_DRIVE_POS {
    unsigned long volatile cylinder_num;
    unsigned long volatile track_num;
    unsigned long volatile sector_num;
};


#include "chapi_disk_drive_iface.h"

#include "chapi.h"

#define _MAX_DISK_DRIVE_NUMBER_ 16


// Base class for all CHAPI disk drives
class chapi_disk_drive_iface;


// Base class for all CHAPI disk controllers.
class chapi_disk_device_iface
{

public:


    // Abstract:
    //      Callback function, disk drive must calls controller`s callback
    //      read_done(...) to notify controller about read command completion.
    //
    // Arguments:
    //      int disk_number        - disk drive instance number
    //      unsigned int status    - disk drive status after read command
completion (can be mixed)
    //                               see status code definition (DRIVE_STS_...).
    //      unsigned int n_of_byte - number of bytes ACTUALLY transferred by
READ operations;
    //
    // Return:
    //      0 - Success
    //
    virtual int read_done(int disk_number, unsigned int status, unsigned int
n_of_byte) = 0;


    // Abstract:
    //      Callback function, disk drive must calls controller`s callback
    //      write_done(...) to notify controller about write command completion.
    //
    // Arguments:
    //      int disk_number        - disk drive instance number
    //      unsigned int status    - disk drive status after read command
completion (can be mixed),
    //                               see status code definition (DRIVE_STS_...).
    //      unsigned int n_of_byte - number of bytes ACTUALLY transferred by
WRITE operations;
    //
    // Return:
```

```
    //      0 - Success
    //
    virtual int write_done(int disk_number, unsigned int status, unsigned int
n_of_byte) = 0;


    // Abstract:
    //      System error logging,
    //      also callback function, tape transport must call controller`s
callback
    //      get_sys_err(...) for system error logging.
    //
    // Arguments:
    //      unsigned long err - system error number
    //
    virtual void get_sys_err( unsigned long err ) = 0;



    // Abstract:
    //      Message logging,
    //      also callback function, tape transport must call controller`s
callback
    //      log_msg(...) for message logging.
    //
    //  Arguments:
    //      log_message_type_t msg_type - type of meesage
    //      const char *file            - the file from where message is logged
    //      int line                    - the line from where mesage is logged
    //      log_message_id_t msg_code   - meesage code
    //      const char *str             - message
    //
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *str)
    {
        if(ci != 0 && ci->log_message_ex != 0) {
            ci->log_message_ex(ci, msg_type, file, line, msg_code, str);
        }
    };



    // Abstract:
    //      Trace logging,
    //      also callback function, tape transport must call controller`s
callback
    //      debug_trace(...) for trace logging.
    //
    // Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    //
    void debug_trace(unsigned char debug_level, const char *str)
    {
        if(ci != 0 && ci->debug_trace != 0) {
            ci->debug_trace(ci, debug_level, str);
        }
    };
```

```
protected:


    //CHAPI start()
    virtual void start() = 0;

    //CHAPI stop()
    virtual void stop() = 0;

    //CHAPI start
    virtual void reset() = 0;

    //CHAPI read()
    virtual int read(unsigned int addr, bool is_byte) = 0;

    //CHAPI write()
    virtual void write(unsigned int addr, int val, bool is_byte) = 0;

    //CHAPI set_configuration()
    virtual int set_configuration(const char * parameters) = 0;

    //CHAPI set_configuration_ex()
    virtual int set_configuration_ex() = 0;

    // Pointer to CHARON-supplied structure
    const chapi_in *ci;

    // Array of CHAPI DISK DRIVE pointers
    chapi_disk_drive_iface * HDD[_MAX_DISK_DRIVE_NUMBER_];

    // Array of disk drive geometry.
    CHAPI_DISK_DRIVE_GEOM disk_geom[_MAX_DISK_DRIVE_NUMBER_];

    // Array of disk drive head's position.
    CHAPI_DISK_DRIVE_POS disk_pos[_MAX_DISK_DRIVE_NUMBER_];
};

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

1.1.8. CHAPI_DISK_DRIVE_IFACE.H

This appendix contains listing of the **chapi_disk_drive_iface.h** file containing basic definitions to be used in a new CHAPI disk drive development.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//
```

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


#if defined(IN_CHAPI_STORAGE_DLL)
#define __chapi_storage_lib__ __declspec(dllexport)
#endif // defined(IN_CHAPI_STORAGE_DLL)

#if defined(USE_CHAPI_STORAGE_DLL)
#define __chapi_storage_lib__ __declspec(dllimport)
#endif // defined(USE_CHAPI_STORAGE_DLL)

#if !defined(__chapi_storage_lib__)
#define __chapi_storage_lib__
#endif // !defined(__chapi_storage_lib__)


// Logging definitions
#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE


 // Message logging ...
#define _ERR_MSG_ error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ info_msg_type, __FILE__, __LINE__

#define LOGMSG(x) log_msg x;


// Debug trace ...
#define L(n) n
#define TRACE(x) debug_trace x;

#include "assert.h"
#include <stdio.h>
#include <Windows.h>
#include "chapi_disk_device_iface.h"


// Disk drive status after READ/WRITE command completion
enum
{
    // Good status
    DRIVE_STS_OK      =  0x0001 << 0,

    // Bad/fatal disk drive error
    DRIVE_STS_BDE     =  0x0001 << 1,

    // Disk drive SEEK error
    DRIVE_STS_SE      =  0x0001 << 2,

    // Disk drive READ error
    DRIVE_STS_RE      =  0x0001 << 3,

    // Disk drive WRITE error
    DRIVE_STS_WE      =  0x0001 << 4,

    // Disk drive HEAD POSITION error
    DRIVE_STS_HPE     =  0x0001 << 5,

    // Disk drive READ/WRITE length error
    DRIVE_STS_LE      =  0x0001 << 6
};


// Base class for all CHAPI disk devices/controllers
class chapi_disk_device_iface;


// Base class for all CHAPI disk drives
class __chapi_storage_lib__ chapi_disk_drive_iface
{
public:

    // Abstract:
    //      Setup parameters for disk drive, MUST be called before start().
    //
    // Arguments:
    //      void * param - pointer to parameter
    //      int arg      - integer parameter
    //
    virtual void setup(void * param, int arg) = 0;



    // Abstract:
    //      Obtain disk drive capacity, return value valid after at once
setup(...) calling only.
    //
    // Arguments:
    //
    // Returns:
    //      Disk drive capacity.
    //
    virtual unsigned _int64 capacity() = 0;



    // Abstract:
    //      Setup disk drive geometry, MUST be called before start().
    //
    // Arguments:
    //      CHAPI_DISK_DRIVE_GEOM geom - disk drive geometry
    //
    virtual void setup_geometry(CHAPI_DISK_DRIVE_GEOM geom) = 0;


    // Abstract:
    //      Start disk drive, MUST be called after any setup functions.
    //
    // Arguments:
    //
    virtual void start() = 0;
    // Abstract:
    //      Stop disk drive.
    //
    // Arguments:
    //
    virtual void stop() = 0;



    // Abstract:
    //      Check if disk drive is online.
    //
    // Arguments:
    //
    // Returns:
    //      true  - disk drive is online
    //      false - disk drive is offline
    //
    virtual bool is_online() = 0;



    // Abstract:
    //      Check if disk is write protected.
    //
    // Arguments:
    //
    // Returns:
    //      true  - disk is write protected
    //      false - disk is not write protected
    //
    virtual bool is_write_prot() = 0;



    // Abstract:
    //      Initiate READ operation on disk drive.
    //
    // Arguments:
    //      CHAPI_DISK_DRIVE_POS disk_pos - disk drive head's position to start
read from.
    //      char * io_buf_p               - buffer to read to (pointer).
    //      unsigned int n_of_byte        - number of bytes to read.
    //
    virtual void do_read(CHAPI_DISK_DRIVE_POS disk_pos, char * io_buf_p,
unsigned int n_of_byte) = 0;



    // Abstract:
    //      Initiate WRITE operation on disk drive.
    //
    // Arguments:
    //      CHAPI_DISK_DRIVE_POS disk_pos - disk drive head's position to start
write from.
    //      char * io_buf_p               - buffer to write from (pointer).
    //      unsigned int n_of_byte        - number of bytes to write.
    //
    virtual void do_write(CHAPI_DISK_DRIVE_POS disk_pos, char * io_buf_p,
unsigned int n_of_byte) = 0;



protected:

    // Abstract:
    //      Constructor of CHAPI disk drive instance
    //
    // Arguments:
    //      void * ctrl                 - disk drive controller instance (pointer)
    //      unsigned char disk_number - current disk drive instance number
    //
    // Returns:
    //
    chapi_disk_drive_iface(void * ctrl, unsigned char disk_number)
    {
        this->ctrl = (chapi_disk_device_iface *) ctrl;
        this->disk_number = disk_number;
    };



    // Abstract:
    // Message logging,
    //
    //  Arguments:
    //     log_message_type_t msg_type  - type of meesage
    //     const char *file             - message source file name
    //     int line                     - message source file line
    //     log_message_id_t msg_code    - meesage code
    //     const char *str              - message
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *fmt, ...);



    // Abstract:
    //      Trace logging,
    //
    // Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    void debug_trace(unsigned char debug_level, const char *fmt, ...);



    // Abstract:
    //      System error logging,
    //
    // Arguments:
    //      unsigned long err - system error number
    //
    //
    void get_sys_err( unsigned long err );
    // Instance of disk drive controller (pointer).
    chapi_disk_device_iface * ctrl;


    // Disk drive instance number.
    unsigned int disk_number;

    // Current disk drive geometry.
    CHAPI_DISK_DRIVE_GEOM disk_geom;

    // Current head's position.
    CHAPI_DISK_DRIVE_POS head_pos;

};


// Type definition of function for creating disk drive instance from DLL
typedef bool (*pf_init_disk_t)(void * ctrl,
    chapi_disk_drive_iface ** p_disk_i, unsigned char disk_number);



#endif // #if !defined(__CHAPI_DISK_DRIVE_IFACE_H__)
```

1.1.9. CHAPI_TAPE_DEVICE_IFACE.H

This appendix contains listing of the **chapi_tape_device_iface.h** file containing basic definitions to be used in a new CHAPI tape controller development.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

#if !defined(__CHAPI_TAPE_DEVICE_IFACE_H__)
#define __CHAPI_TAPE_DEVICE_IFACE_H__

#include "chapi_tape_transport_iface.h"

#include "chapi.h"

#define _MAX_TAPE_NUMBER_ 16


// Base class for all CHAPI tape transports
class chapi_tape_transport_iface;


// Base class for all CHAPI tape controllers.
class chapi_tape_device_iface
{
```

```
public:


    // Abstract:
    //      The only callback function, tape transport must calls controller`s
callback
    //      cmd_done(...) to notify controller about requested command
completion.
    //
    // Arguments:
    //      int transport_number   - tape transport instance number
    //      int cmd                - requested command
    //      unsigned int status    - tape transport status after requested
command completion (can be mixed)
    //      unsigned int n_of_data - depends on operartion:
    //                               number of bytes ACTUALLY transferred by
READ/WRITE operations;
    //                               number of items NOT skipped by SKIP
operations;
    //
    // Return:
    //      0 - Success
    //
    virtual int cmd_done(int transport_number, int cmd, unsigned int status,
unsigned int n_of_data) = 0;



    // Abstract:
    //      System error logging,
    //      also callback function, tape transport must call controller`s
callback
    //      get_sys_err(...) for system error logging.
    //
    // Arguments:
    //      unsigned long err - system error number
    //
    virtual void get_sys_err( unsigned long err ) = 0;



    // Abstract:
    //      Message logging,
    //      also callback function, tape transport must call controller`s
callback
    //      log_msg(...) for message logging.
    //
    //  Arguments:
    //      log_message_type_t msg_type - type of meesage
    //      const char *file            - source file name
    //      inr line                    - source file line
    //      const char *str             - message
    //
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *str)
    {
        if(ci != 0 && ci->log_message_ex != 0) {
            ci->log_message_ex(ci, msg_type, file, line, msg_code, str);
        }
    };



    // Abstract:
    //      Trace logging,
    //      also callback function, tape transport must call controller`s
callback
    //      debug_trace(...) for trace logging.
    //
    // Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    //
    void debug_trace(unsigned char debug_level, const char *str)
    {
        if(ci != 0 && ci->debug_trace != 0) {
            ci->debug_trace(ci, debug_level, str);
        }
    };



protected:



    //CHAPI start()
    virtual void start() = 0;

    //CHAPI stop()
    virtual void stop() = 0;

    //CHAPI start
    virtual void reset() = 0;

    //CHAPI read()
    virtual int read(unsigned int addr, bool is_byte) = 0;

    //CHAPI write()
    virtual void write(unsigned int addr, int val, bool is_byte) = 0;

    //CHAPI set_configuration()
    virtual int set_configuration(const char * parameters) = 0;

    //CHAPI set_configuration_ex()
    virtual int set_configuration_ex() = 0;

    // Pointer to CHARON-supplied structure
    const chapi_in *ci;

    // Array of CHAPI TAPE TRANSPORT pointers
    chapi_tape_transport_iface * tape[_MAX_TAPE_NUMBER_];
};


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

1.1.10.CHAPI_TAPE_TRANSPORT_IFACE.H

This appendix contains listing of the **chapi_tape_transport_iface.h** file containing basic definitions to be used in a new CHAPI tape transport development.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//


#if !defined(__CHAPI_TAPE_TRANSPORT_IFACE_H__)
#define __CHAPI_TAPE_TRANSPORT_IFACE_H__

#if defined(IN_CHAPI_STORAGE_DLL)
#define __chapi_storage_lib__ __declspec(dllexport)
#endif // defined(IN_CHAPI_STORAGE_DLL)

#if defined(USE_CHAPI_STORAGE_DLL)
#define __chapi_storage_lib__ __declspec(dllimport)
#endif // defined(USE_CHAPI_STORAGE_DLL)

#if !defined(__chapi_storage_lib__)
#define __chapi_storage_lib__
#endif // !defined(__chapi_storage_lib__)


// Logging definitions
#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE


 // Message logging ...
#define _ERR_MSG_ error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ info_msg_type, __FILE__, __LINE__

#define LOGMSG(x) log_msg x;


// Debug trace ...
#define L(n) n
#define TRACE(x) debug_trace x;


#include "Winsock2.h"
#include "assert.h"
#include <stdio.h>
#include <Windows.h>
```

```
#include "chapi_tape_device_iface.h"



// Tape transport commands.
enum
{
    // Erase tape
    TAPE_CMD_ERASE,

    // Write tape mark
    TAPE_CMD_WRITE_MARK,

    // Rewrite tape mark
    TAPE_CMD_REWRITE_MARK,

    // Rewind tape
    TAPE_CMD_REWIND,

    // Skip some tape records in forward direction
    TAPE_CMD_SKIP_RECORD,

    // Skip some tape marks in forward direction
    TAPE_CMD_SKIP_MARK,

    // Skip some tape records in backward direction
    TAPE_CMD_SKIP_RECORD_REV,

    // Skip some tape marks in backward direction
    TAPE_CMD_SKIP_MARK_REV,

    // Read one tape record in forward direction
    TAPE_CMD_READ,

    // Read one tape record in backward direction
    TAPE_CMD_READ_REV,

    // Reread one next tape record
    TAPE_CMD_REREAD_NEXT,

    // Reread one previous tape record
    TAPE_CMD_REREAD_PREV,

    // Write one tape record
    TAPE_CMD_WRITE,

    // Rewrite one tape record
    TAPE_CMD_REWRITE,

    // Set tape transport offline
    // Depends on implementation
    TAPE_CMD_OFFLINE,

    // Set tape transport online
    // Depends on implementation
    TAPE_CMD_ONLINE,

    // Obtain tape transport status
    // Depends on implementation
    TAPE_CMD_STATUS
```

```
};


// Tape transport status after command completion
enum
{
    // Good status
    TAPE_STS_OK      =  0x0001 << 0,

    // Bad/fatal tape error, position lost
    TAPE_STS_BTE     =  0x0001 << 1,

    // Tape mark detected
    TAPE_STS_MARK    =  0x0001 << 2,

    // Physical end of tape detected
    TAPE_STS_PEOT    =  0x0001 << 3,

    // Logical end of tape detected
    // Usually double tape mark, position between marks
    TAPE_STS_LEOT    =  0x0001 << 4,

    //Beginning of tape detected
    TAPE_STS_BOT     =  0x0001 << 5,

    //End of recorded data detected, position lost
    TAPE_STS_EOD     =  0x0001 << 6,
};



// Base class for all CHAPI tape devices/controllers
class chapi_tape_device_iface;


// Base class for all CHAPI tape transports
class __chapi_storage_lib__ chapi_tape_transport_iface
{
public:

    // Abstract:
    //      Setup parameters for tape transport, MUST be called before start().
    //
    // Arguments:
    //      void * param - pointer to parameter
    //      int arg      - integer parameter
    //
    virtual void setup(void * param, int arg) = 0;



    // Abstract:
    //      Start tape transport, MUST be called after setup(...).
    //
    // Arguments:
    //
    virtual void start() = 0;
    // Abstract:
    //      Stop tape transport.
    //
    // Arguments:
    //
    virtual void stop() = 0;



    // Abstract:
    //      Check if tape transport is online.
    //
    // Arguments:
    //
    // Returns:
    //      true  - tape transport is online
    //      false - tape transport is offline
    //
    virtual bool is_online() = 0;



    // Abstract:
    //      Check if tape is write protected.
    //
    // Arguments:
    //
    // Returns:
    //      true  - tape is write protected
    //      false - tape is not write protected
    //
    virtual bool is_write_prot() = 0;



    // Abstract:
    //      Initiate tape transport operation.
    //
    // Arguments:
    //      unsigned int cmd       - tape transport operation
    //      char * io_buf_p        - IO buffer for READ/WRITE operations,
    //      unsigned int n_of_data - depends on operation:
    //                               number of bytes to write for WRITE
operations;
    //                               size of IO buffer for READ operations
    //                               number of skip items  for SKIP operations;
    //
    // Returns:
    //
    virtual void do_command(unsigned int cmd, char * io_buf_p, unsigned int
n_of_data) = 0;



protected:

    // Abstract:
    //      Constructor of CHAPI tape transport instance
    //
    // Arguments:
```

```
    //      void * ctrl                    - tape transport controller instance
(pointer)
    //      unsigned char transport_number - current tape transport instance
number
    //
    // Returns:
    //
    chapi_tape_transport_iface(void * ctrl, unsigned char transport_number)
    {
        this->ctrl = (chapi_tape_device_iface *) ctrl;
        this->transport_number = transport_number;
    };



    // Abstract:
    // Message logging,
    //
    //  Arguments:
    //     log_message_type_t msg_type   - type of meesage
    //     const char *file            - source file name
    //     int line                    - source file line
    //     log_message_id_t msg_code     - meesage code
    //     const char *str               - message
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *fmt, ...);



    // Abstract:
    //      Trace logging,
    //
    // Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    void debug_trace(unsigned char debug_level, const char *fmt, ...);



    // Abstract:
    //      System error logging,
    //
    // Arguments:
    //      unsigned long err - system error number
    //
    //
    void get_sys_err( unsigned long err );


    // Instance of tape transport controller (pointer).
    chapi_tape_device_iface * ctrl;


    // Tape transport instance number.
    unsigned int transport_number;

};
```

```
// Type definition of function for creating tape transport instance from DLL
typedef bool (*pf_init_tape_t)(void * ctrl,
    chapi_tape_transport_iface ** p_tape_i, unsigned char transport_number);



#endif // #if !defined(__CHAPI_TAPE_TRANSPORT_IFACE_H__)
```

1.1.11.CHAPI_PARALLEL_CTLR_INTERFACE.H

This appendix contains listing of the **chapi_parallel_device_iface.h** file containing basic definitions to be used in a new CHAPI parallel line controller development.

```
//
// Copyright (C) 1999-2006 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_PARALLEL_CTLR_INTERFACE_H__)
#define __CHAPI_PARALLEL_CTLR_INTERFACE_H__

#include "chapi_parallel_port_interface.h"

#include "chapi.h"

// Base class for all CHAPI parallel IO controllers.
class __chapi_parallel_lib__ chapi_parallel_controller_interface
{
public:
    chapi_parallel_controller_interface();
    ~chapi_parallel_controller_interface();

    //CHAPI start()
    virtual void start() = 0;

    //CHAPI stop()
    virtual void stop() = 0;

    //CHAPI start
    virtual void reset() = 0;

    //CHAPI read()
    virtual int read(unsigned int addr, bool is_byte) = 0;

    //CHAPI write()
    virtual void write(unsigned int addr, int val, bool is_byte) = 0;

    //CHAPI set_configuration()
    virtual int set_configuration(const char * parameters) = 0;
    //CHAPI set_configuration_ex()
    virtual int set_configuration_ex() = 0;

    // Callback function, port must call controller`s callback
    // function rx_done(...) when it receives a new data.
    //
    //  Arguments:
    //
    //  Return : 0 - No errors
    //
    virtual int rx_done() = 0;

    // Callback function, port must call controller`s callback
    // function extra(...) for extra features.
    //
    //  Arguments:
    //      void * extra          - void pointer
    //      int arg               - integer argument
    //
    //  Return : 0 - No errors
    //
    virtual int extra(void * extra, int arg) = 0;


    // Callback function, port must call controller`s callback
    // function op_done(...) when it completes reset, write or read operation.
    //
    //  Arguments:
    //
    //  Return : 0 - No errors
    //
    virtual int op_done(void * p, int arg) = 0;

    virtual int dma_done(void * p, int arg) = 0;

    virtual int get_dma_ptr(void * p, int arg) = 0;

    virtual int fill_dma_buf(void * p, int arg) = 0;


protected:

    // Pointer to CHARON-supplied structure
    const chapi_in *ci;

    // CHAPI parallel port interface class
    // Base class for all CHAPI parallel ports
    friend class chapi_parallel_port_interface;

    // Each controller can be connected just to one port
    chapi_parallel_port_interface * port;


    // System error logging,
    // also callback function, port must call controller`s callback
    // get_sys_err(...) for system error logging.
    //
    //  Arguments:
    //     unsigned long err - system error number
    //
    //
    virtual void get_sys_err( unsigned long err ) = 0;


    // Message logging,
    // also callback function, port must call controller`s callback
    // log_msg(...) for message logging.
    //
    //  Arguments:
    //     log_message_type_t msg_type  - type of meesage
    //     const char *file             - file from where the message is logged
    //     int line                     - line from where the message is logged
    //     log_message_id_t msg_code    - message code
    //     const char *str              - message
    //
    //
    void log_msg(log_message_type_t msg_type, const char *file, int line,
        log_message_id_t msg_code, const char *str)
    {
        if(ci != 0 && ci->log_message_ex != 0) {
            ci->log_message_ex(ci, msg_type, file, line, msg_code, str);
        }
    };


    // Trace logging,
    // also callback function, port must call controller`s callback
    // debug_trace(...) for trace logging.
    //
    //  Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    //
    void debug_trace(unsigned char debug_level, const char *str)
    {
        if(ci != 0 && ci->debug_trace != 0) {
            ci->debug_trace(ci, debug_level, str);
        }
    };

};



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

1.1.12.CHAPI_PARALLEL_PORT_INTERFACE.H

This appendix contains listing of the **chapi_parallel_port_iface.h** file which contains basic definitions to be used in development of new CHAPI parallel port.

```
//
// Copyright (C) 1999-2006 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_PARALLEL_PORT_INTERFACE_H__)
#define __CHAPI_PARALLEL_PORT_INTERFACE_H__


#if defined(IN_CHAPI_PARALLEL_DLL)
#define __chapi_parallel_lib__ __declspec(dllexport)
#endif // defined(IN_CHAPI_PARALLEL_DLL)

#if defined(USE_CHAPI_PARALLEL_DLL)
#define __chapi_parallel_lib__ __declspec(dllimport)
#endif // defined(USE_CHAPI_PARALLEL_DLL)

#if !defined(__chapi_parallel_lib__)
#define __chapi_parallel_lib__
#endif // !defined(__chapi_parallel_lib__)

#include "chapi.h"

#define WRITE_CYCLE_SOFT 0x00000001
#define ATTN_SIGN        0x00000100
#define LINK_MODE        0x00010000

// Logging definitions
#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE


 // Message logging ...
#define _ERR_MSG_ error_msg_type
#define _WARN_MSG_ warning_msg_type
#define _INFO_MSG_ info_msg_type
#define LOGMSG(x) log_msg x;


// Debug trace ...
#define L(n) n
#define TRACE(x) debug_trace x;

#if defined(_WIN32)
typedef char            byte_t;
typedef unsigned char   ubyte_t;

typedef short           word_t;
typedef unsigned short  uword_t;

typedef int       dword_t;
typedef unsigned int    udword_t;
#endif // defined(_WIN32)

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

```
// CHAPI parallel IO controller interface
class chapi_parallel_controller_interface;

// CHAPI serial line interface
class __chapi_parallel_lib__ chapi_parallel_port_interface
{
public:
    enum wait_state { abort = 0, error = 1, state_changed = 2 };

    unsigned char volatile pulse_len;

    unsigned short volatile bus_no, device_no, function_no;

public:

    // Constructor of CHAPI parallel port instance
    //
    // Arguments :
    //      void * ctlr           - parallel controller instance (pointer)
    //
    chapi_parallel_port_interface(void * ctlr)
    {
        this->ctlr = (chapi_parallel_controller_interface *) ctlr;
    }

    virtual ~chapi_parallel_port_interface() {}

    // Start parallel port.
    //
    //  Return : 0 - No errors
    //
    virtual int start() = 0;


    // Stop parallel port.
    //
    //  Return :
    //      0 - No errors
    //
    virtual int stop() = 0;

    // Set controller BRQ descriptor.
    //
    //  Return :
    //      0 - No errors
    //
    virtual int set_dsc(void * p_dsc, udword_t len) = 0;

    // Reset parallel port.
    //
    //  Return :
    //      0 - No errors
    //
    virtual int reset() = 0;

    virtual dword_t read(udword_t addr, dword_t size) = 0;

    virtual void write(udword_t addr, dword_t size, dword_t val) = 0;
    virtual udword_t conn_out_read_in_ctrl() = 0;

    virtual udword_t conn_out_write_out_ctrl(udword_t val) = 0;

    virtual udword_t conn_out_read_out_ctrl() = 0;

    virtual udword_t conn_out_read_out_data() = 0;

    virtual udword_t conn_out_write_out_data(udword_t val) = 0;


    virtual udword_t conn_in_read_in_ctrl() = 0;

    virtual udword_t conn_in_write_out_ctrl(udword_t val) = 0;

    virtual udword_t conn_in_read_out_ctrl() = 0;

    virtual udword_t conn_in_read_in_data() = 0;

    virtual udword_t conn_in_write_in_data(udword_t val) = 0;


    virtual udword_t write_extra(void * p, udword_t val) = 0;

    virtual udword_t read_extra(void * p, udword_t val) = 0;

    virtual udword_t extra(void * p, udword_t val) = 0;

    virtual udword_t start_dma(udword_t op, void * buf, udword_t size) = 0;




protected:

    //----------------------------------------------------------------------
    // Message logging / debugging routines
    //


    // Message logging,
    //
    //  Arguments:
    //     log_message_type_t msg_type  - type of meesage
    //     const char *file             - file from where the message is logged
    //     int line                     - line from where the message is logged
    //     log_message_id_t msg_code    - meesage code
    //     const char *str              - message
    //
    virtual void log_msg(log_message_type_t msg_type, const char *file, int
line,
        log_message_id_t msg_code, const char *fmt, ...);



    // Trace logging,
    //
    //  Arguments:
    //      unsigned char debug_level - debug level
    //      const char *str - trace message
    //
    virtual void debug_trace(unsigned char debug_level, const char *fmt, ...);


    // System error logging,
    //
    //  Arguments:
    //     unsigned long err - system error number
    //
    //
    virtual void get_sys_err( unsigned long err );

    //Instance of parallel controller (pointer)
    chapi_parallel_controller_interface* ctlr;

    // Wrapper for callback function, port must call function
    // ctrl_extra(...) for controller`s extra features.
    //
    //  Arguments:
    //      void * extra          - void pointer
    //      int arg               - integer argument
    //
    //  Return : 0 - No errors
    //
    int ctrl_extra(void * extra, int arg);



};


typedef bool (*pf_init_parallel_port_t)(void * ctrl,
    chapi_parallel_port_interface ** p_parallel_port);


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

1.1.13.PCI_REG.H

This appendix contains **pci_reg.h** header file which contains some basic definitions to work with PCI bus hardware.

```
//
// Copyright 2006 Software Resources International
//
// These header files describe access via a standard API to CHARON-VAX,
// which is a proprietary VAX emulator product of Software Resources
// International. The use of CHARON-VAX, and the development, distribution
// and use with CHARON-VAX of any software interconnection based on this
// API is authorized only pursuant to a valid CHARON-VAX license from
// Software Resources International.
//

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

```
// A few PCI local configuration registers (used by BCI-2X0X).
#define PLX_DMCA        0x00000024

#define PLX_DMCR        0x00000028

#define PLX_DMCR_DMAE       0x00000001
#define PLX_DMCR_DIAE       0x00000002
#define PLX_DMCR_LLIE       0x00000004
#define PLX_DMCR_DPRS       0x00000008
#define PLX_DMCR_DMPR       0x00000010


// A few PCI shared runtime registers (used by BCI-2X0X)
#define PLX_ICSR        0x00000068

#define PLX_ICSR_PIE        0x00000100
#define PLX_ICSR_PAIE       0x00000400
#define PLX_ICSR_PLIE       0x00000800
#define PLX_ICSR_PAI        0x00004000
#define PLX_ICSR_LI         0x00008000
#define PLX_ICSR_LIE        0x00010000
#define PLX_ICSR_L0IE       0x00040000
#define PLX_ICSR_L1IE       0x00080000


#define PLX_EPIR        0x0000006C

#define PLX_EPIR_ASR    0x40000000


// A few PCI operation registers (used by DCI-1100)
#define AMCC_OMR_1            0x00000000
#define AMCC_MWAR       0x00000024
#define AMCC_MWTCR            0x00000028
#define AMCC_MRAR       0x0000002C
#define AMCC_MRTCR            0x00000030
#define AMCC_MBES       0x00000034

#define AMCC_ICSR       0x00000038

#define AMCC_OBOXI            (1 << 16)
#define AMCC_MABRTI           (1 << 20)
#define AMCC_TABRTI           (1 << 21)
#define AMCC_INTON            (1 << 23)

#define AMCC_BMCSR            0x0000003C


#endif  //  __PCI_REG_H__
```

1.2. CHAPI device code

1.2.1. DLV11

chapi_dlv11.h

```
//
```

```
// Copyright (C) 1999-2006 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_DLV11_H__)
#define __CHAPI_DLV11_H__


#define _WIN32_WINNT 0x0400

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#include <WinSock2.h>

//
// Default bus address & interrupt vector
// Used if not specified in .cfg file
//
#define DEFAULT_ADDRESS 017760010
#define DEFAULT_VECTOR  0310

#include "chapi_lib.h"

#include "chapi_serial_device_iface.h"


class chapi_dlv11:public chapi_serial_device_interface
{
public:

    chapi_dlv11(const chapi_in *_ci, const char *instance_name);

    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);

    int set_configuration_ex();

    void setup_bus_requests();

protected:

    // context of DLV11 device
    struct {
```

```
    int baud;
    char char_len;
    char stop_len;
    char parity_en;
    bool breaken;
    bool baudsel;
    bool dtr;
    bool rts;
    bool erroren;
    bool is_terminal;
    char mode;
} context;



//  dlv11 register structure definition, see dlv11 manual
union RCSR_REGISTER_UNION {
    struct {
        UWORD rdr_enb      : 1;
        UWORD dtr          : 1;
        UWORD rts          : 1;
        UWORD sec_xmit     : 1;
        UWORD undef_1      : 1;
        UWORD dset_int_enb : 1;
        UWORD rcvr_int_enb : 1;
        UWORD rcvr_done    : 1;
        UWORD undef_2      : 2;
        UWORD sec_rec      : 1;
        UWORD rcvr_act     : 1;
        UWORD dcd          : 1;
        UWORD cts          : 1;
        UWORD ring         : 1;
        UWORD data_set_int : 1;
    };
    UWORD value;
} RCSR;


//  dlv11 register structure definition, see dlv11 manual
union RBUF_REGISTER_UNION {
    struct {
        UWORD character   : 8;
        UWORD undef       : 4;
        UWORD parity_err  : 1;
        UWORD framing_err : 1;
        UWORD overrun_err : 1;
        UWORD error       : 1;
    };
    UWORD value;
} RBUF;


//  dlv11 register structure definition, see dlv11 manual
union XCSR_REGISTER_UNION {
    struct {
        UWORD start_break  : 1;
        UWORD undef_1      : 1;
        UWORD maint        : 1;
        UWORD undef_2      : 3;
        UWORD xmit_int_enb : 1;
        UWORD xmit_rdy     : 1;
        UWORD undef_3      : 3;
        UWORD pbr_sel_enb  : 1;
        UWORD speed        : 4;
    };
    UWORD value;
} XCSR;


//  dlv11 register structure definition, see dlv11 manual
union XBUF_REGISTER_UNION {
    struct {
        UWORD character   : 8;
        UWORD undef       : 8;
    };
    UWORD value;
} XBUF;


// Masks for RCSR register
enum {
    rcsr_data_set_int_mask  = 0x0001 << 15,
    rcsr_ring_mask          = 0x0001 << 14,
    rcsr_cts_mask           = 0x0001 << 13,
    rcsr_dcd_mask           = 0x0001 << 12,
    rcsr_rcvr_act_mask      = 0x0001 << 11,
    rcsr_sec_rec_mask       = 0x0001 << 10,
    rcsr_rcvr_done_mask     = 0x0001 << 7,
    rcsr_rcvr_int_enb_mask  = 0x0001 << 6,
    rcsr_dset_int_enb_mask  = 0x0001 << 5,
    rcsr_sec_xmit_mask      = 0x0001 << 3,
    rcsr_rts_mask           = 0x0001 << 2,
    rcsr_dtr_mask           = 0x0001 << 1,
    rcsr_rdr_enb_mask       = 0x0001
};

// Masks for XCSR register
enum {
    xcsr_pbr_sel_mask       = 0x000f << 12,
    xcsr_pbr_enb_mask       = 0x0001 << 11,
    xcsr_xmit_rdy_mask      = 0x0001 << 7,
    xcsr_xmit_int_enb_mask  = 0x0001 << 6,
    xcsr_maint_mask         = 0x0001 << 2,
    xcsr_start_break_mask   = 0x0001
};

// Length of all ring buffers - must be power of 2
enum {
    ring_buf_size = 1 << 10
};


int rx_done(unsigned int len, unsigned char line_id);

int tx_done(unsigned int len, unsigned char line_id);

static void tx_irq_requestor(void * arg1, int arg2);

static void rx_irq_requestor(void *arg1, int arg2);
static void signal_irq_requestor(void *arg1, int arg2);

static void put_errors_in(void * arg1, int arg2);

int get_tx_char(unsigned char & from_buf, unsigned char line_id);

int input_signal(unsigned char in_signal, unsigned char line_id);

int error_tx(unsigned char error, unsigned char line_id);

int error_rx(unsigned char error, unsigned char line_id);

int extra(void * extra, int arg, unsigned char line_id);

static int tx_brq_ack(chapi_dlv11 *the_dlv11);

static int rx_brq_ack(chapi_dlv11 *the_dlv11);

// RS line is TX ready if we have some space to store outgoing data
bool tx_ready() {
    return (tx_ring_end + 1 - tx_ring_start) % ring_buf_size != 0;
}

// RS line is RX ready if we have some space to store data
bool rx_ready() {
    return (rx_ring_end + 1 - rx_ring_start) % ring_buf_size != 0;
}

// Determine free space in RX ring buffer
unsigned int rx_free_space() {
    return((rx_ring_start - rx_ring_end) % ring_buf_size);
}

void get_sys_err( unsigned long err );


//configuration options
chapi_string_option_t line_dll;
chapi_string_option_t line_cfg;
chapi_string_option_t line_param;
chapi_bool_option_t line_is_terminal;
chapi_string_option_t mode;
chapi_integer_option_t char_len;
chapi_integer_option_t stop_len;
chapi_integer_option_t baud;
chapi_string_option_t parity;
chapi_bool_option_t baudselen;
chapi_bool_option_t breaken;
chapi_bool_option_t dtr;
chapi_bool_option_t erroren;
chapi_bool_option_t rts;


//----------------------------------------------------------------------------

//
// chapi_dlv11 uses only 1 I/O region and 2 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

    // Bus request
    chapi_brq_t tx_brq;
    chapi_brq_t rx_brq;


    //
    // RX ring buffer.
    // dlv11 device use this buffer to store received data.
    // When reads from RBUF register it reads from rx_ring_buf[].
    //
    RBUF_REGISTER_UNION volatile rx_ring_buf[ring_buf_size];

    //
    // TX ring buffer.
    // In order to speed up completion of write() procedure,
    // data written to XBUF register is just stored here.
    // Working TX/RX thread will do the real job in the background.
    //
    unsigned char volatile tx_ring_buf[ring_buf_size];

    // Pointers on TX ring buffers parts for the line
    unsigned int volatile tx_ring_start;
    unsigned int volatile tx_ring_done;
    unsigned int volatile tx_ring_end;

    // Pointers on RX ring buffers parts for the line
    unsigned int volatile rx_ring_start;
    unsigned int volatile rx_ring_done;
    unsigned int volatile rx_ring_end;

    //state flags
    bool volatile flag_maint_mode;
    bool volatile flag_flow_control;


    // interrupts enable/disable flags
    bool rx_int_en;
    bool tx_int_en;
    bool dset_int_en;

    //for load dll file
    HINSTANCE hinst;

};

#endif // #if !defined(__CHAPI_DLV11_H__)
```

chapi_dlv11.cxx

```
//
// Copyright (C) 1999-2006 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.
//


#include "chapi_dlv11.h"

// Error codes for the CHAPI_DLV11.DLL
#include "chapi_dlv11_msgid.h"

#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE
//
// Debug/trace and message logging.
// Format is as follows:
//
// LOGMSG((_ERR_MSG_, <message_code>, <format string>, ...))
// LOGMSG((_WARN_MSG_, <message_code>, <format string>, ...))
// LOGMSG((_INFO_MSG_, <message_code>, <format string>, ...))
//
// TRACE((L(k), <format string>, ...)),
// where k in [0, 10] defines trace level.
//

// Message logging ...
#define _ERR_MSG_ p->ci, error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ p->ci, warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ p->ci, info_msg_type, __FILE__, __LINE__

#define LOGMSG(x)\
    if(p->ci != 0 && p->ci->log_message_ex != 0) { \
        p->ci->log_message_ex x; \
    }

// Debug trace ...
#define L(n) p->ci, n
#define TRACE(x)\
    if(p->ci != 0 && p->ci->debug_trace != 0) { \
        p->ci->debug_trace x; \
    }



void chapi_dlv11::tx_irq_requestor(void * arg1, int arg2)
{
    chapi_dlv11 *p = (chapi_dlv11 *)(arg1);

    TRACE((L(5),"*********** TX_IRQ_REQUESTOR -> SET XCSR.xmit_rdy "));
    p->XCSR.xmit_rdy = 1;

    // And request interrupt, if enabled.
    if(p->XCSR.xmit_int_enb) {
        TRACE((L(5),"*********** TX_IRQ_REQUESTOR -> PUT TX IRQ"));
        p->tx_brq.set();
    }
}



void chapi_dlv11::rx_irq_requestor(void *arg1, int arg2)
{
    unsigned char from_buf;

    chapi_dlv11 *p = (chapi_dlv11 *)(arg1);

    p->rx_ring_start = p->rx_ring_done;

    if((p->rx_ring_end - p->rx_ring_done) != 0) {
        TRACE((L(5),"*********** RX_IRQ_REQUESTOR -> SET RCSR.rcvr_done "));
        p->RCSR.rcvr_done = 1;

        // And request interrupt, if enabled.
        if(p->RCSR.rcvr_int_enb) {
            TRACE((L(5),"*********** RX_IRQ_REQUESTOR -> PUT RX IRQ"));
            p->rx_brq.set();
        }
        return;
    }

    if(!p->rx_ready()){
            LOGMSG((_ERR_MSG_, RX_BUFFER_FULL, "RX BUFFER IS FULL !!!"));
    }
    else {
        if(p->line[0]){
            if(p->line[0]->get_rx_char(from_buf) != -1) {
                p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].value = 0;
                p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].character =
from_buf;
                p->rx_ring_end++;
            }
        }
    }


    if((p->rx_ring_end - p->rx_ring_done) != 0) {
        TRACE((L(5),"*********** RX_IRQ_REQUESTOR -> SET RCSR.rcvr_done "));
        p->RCSR.rcvr_done = 1;

        // And request interrupt, if enabled.
        if(p->RCSR.rcvr_int_enb) {
            TRACE((L(5),"*********** RX_IRQ_REQUESTOR -> PUT RX IRQ"));
            p->rx_brq.set();
        }
    }
}


void chapi_dlv11::signal_irq_requestor(void *arg1, int arg2)
{
    chapi_dlv11 *p = (chapi_dlv11 *)(arg1);

    if(((unsigned char)(arg2)) & C_T_S) {
        p->RCSR.cts = 1;
    }
    else {
        p->RCSR.cts = 0;
    }

    if(((unsigned char)(arg2)) & D_C_D) {
        p->RCSR.dcd = 1;
    }
    else {
        p->RCSR.dcd = 0;
    }

    if(((unsigned char)(arg2)) & R_I_N_G) {
        p->RCSR.ring = 1;
    }
    else {
        p->RCSR.ring = 0;
    }


    TRACE((L(5),"*********** SIGNAL_IRQ_REQUESTOR -> SET RCSR.data_set_int "));
    p->RCSR.data_set_int = 1;

    // And request interrupt, if enabled.
    if(p->RCSR.dset_int_enb) {
        TRACE((L(5),"*********** SIGNAL_IRQ_REQUESTOR -> PUT RX IRQ"));
        p->rx_brq.set();
    }
}


void chapi_dlv11::put_errors_in(void * arg1, int arg2)
{
    chapi_dlv11 *p = (chapi_dlv11 *)(arg1);
    unsigned char error = (unsigned char) arg2;

    p->rx_ring_start = p->rx_ring_done;

    if(!p->rx_ready()){
        LOGMSG((_ERR_MSG_, RX_BUFFER_FULL, "RX BUFFER IS FULL !!!"));
        return;
    }

    if(error & LINE_ERROR_BREAK){
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].value = 0;
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].character = 0x00;
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].error = 1;
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].framing_err;
        p->rx_ring_end++;
        error &=~((unsigned char)(LINE_ERROR_BREAK));
    }

    if(!p->rx_ready()){
        LOGMSG((_ERR_MSG_, RX_BUFFER_FULL, "RX BUFFER IS FULL !!!"));
        return;
    }

    if(error){
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].value = 0;
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].character = 0xff;
        p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].error = 1;

        if(error & LINE_ERROR_FRAMING){
            p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].framing_err = 1;
        }
        if(error & LINE_ERROR_PARITY){
            p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].parity_err = 1;
        }
        if(error & LINE_ERROR_OVER){
            p->rx_ring_buf[p->rx_ring_end % p->ring_buf_size].overrun_err = 1;
        }

        p->rx_ring_end++;
    }

    TRACE((L(5),"*********** PUT_ERRORS_IN -> SET RCSR.rcvr_done "));
    p->RCSR.rcvr_done = 1;

    // And request interrupt, if enabled.
    if(p->RCSR.rcvr_int_enb) {
        TRACE((L(5),"*********** PUT_ERRORS_IN -> PUT RX IRQ"));
        p->rx_brq.set();
    }
}



int chapi_dlv11::tx_brq_ack(chapi_dlv11 *p)
{
    TRACE((L(5),"p->tx_brq.clear()"));
    p->tx_brq.clear();
    return (p) ? p->i_vector+4 : -1;
}


int chapi_dlv11::rx_brq_ack(chapi_dlv11 *p)
{
    TRACE((L(5),"p->rx_brq.clear()"));
    p->rx_brq.clear();
    return (p) ? p->i_vector : -1;
}




#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE
//
// Debug/trace and message logging.
// Format is as follows:
//
// LOGMSG((_ERR_MSG_, <message_code>, <format string>, ...))
// LOGMSG((_WARN_MSG_, <message_code>, <format string>, ...))
// LOGMSG((_INFO_MSG_, <message_code>, <format string>, ...))
//
// TRACE((L(k), <format string>, ...)),
// where k in [0, 10] defines trace level.
//

// Message logging ...
#define _ERR_MSG_ ci, error_msg_type, __FILE__, __LINE__
#define _WARN_MSG_ ci, warning_msg_type, __FILE__, __LINE__
#define _INFO_MSG_ ci, info_msg_type, __FILE__, __LINE__

#define LOGMSG(x)\
    if(ci != 0 && ci->log_message_ex != 0) { \
        ci->log_message_ex x; \
    }

// Debug trace ...
#define L(n) ci, n
#define TRACE(x)\
    if(ci != 0 && ci->debug_trace != 0) { \
        ci->debug_trace x; \
    }



chapi_dlv11::chapi_dlv11(const chapi_in *_ci, const char *instance_name)
:rx_brq(0x04, (__chapi_brq_acknowledge_p)rx_brq_ack, (void*)this),
tx_brq(0x04, (__chapi_brq_acknowledge_p)tx_brq_ack, (void*)this),
line_dll(_ci, "line_dll", 1, _MAX_PATH),
line_cfg(_ci, "line_cfg", 1, 32),
line_param(_ci, "line_param", 1, 2000),
line_is_terminal(_ci, "line_is_terminal", 1),
mode(_ci, "mode", 1, 5),
char_len(_ci, "char_len",1),
stop_len(_ci, "stop_len",1),
baud(_ci, "baud_rate",1),
baudselen(_ci, "baudselen", 1),
breaken(_ci, "breaken", 1),
parity(_ci, "parity", 1, 10),
dtr(_ci, "dtr", 1),
erroren(_ci, "erroren", 1),
rts(_ci, "rts", 1)
{
    ci = _ci;
    memset(&context, 0, sizeof(context));

    context.mode = 'e';
    context.char_len = 8;
    context.stop_len = 1;
    context.parity_en = 0;
    context.baud = 9600;
    context.baudsel = false;
    context.breaken = false;
    context.dtr = false;
    context.erroren = false;
    context.rts = false;
    context.is_terminal = false;

    tx_brq.clear();
    rx_brq.clear();
```

```
}



void chapi_dlv11::start()
{
    b_address = ci->base_b_address;
    i_vector = ci->base_i_vector;

    tx_brq.enable();
    rx_brq.enable();


    TRACE((L(1),"CHAPI dlv11 START"));

    RCSR.value = 0;
    XCSR.value = 0;
    RBUF.value = 0;
    XBUF.value = 0;

    XCSR.xmit_rdy = 1;

    rx_int_en = false;
    tx_int_en = false;
    dset_int_en = false;

    ::memset(line, 0, sizeof(line));

    tx_ring_start = tx_ring_done = tx_ring_end = 0;
    rx_ring_start = rx_ring_done = rx_ring_end = 0;

    flag_maint_mode = false;

    if(!line_param[0].is_specified()){
            LOGMSG((_ERR_MSG_, LINE_INSTANCE_CRE_ERROR,
                "UNABLE CREATE LINE, line_param NOT specified"));
            return;
    }

    if(!line_dll[0].is_specified()) {
        // Set default value to "chapi_serial"
        line_dll[0].set("chapi_serial");
        line_dll[0].commit();
        line_dll[0].change_ack();
    }

    if(!line_cfg[0].is_specified()) {
        // Set default value to "tcpip"
        line_cfg[0].set("tcpip");
        line_cfg[0].commit();
        line_cfg[0].change_ack();
    }

    // create line instance
    TRACE((L(1),"CHAPI dlv11 STARTING UNIT"));

    // create line instance
    TRACE((L(1),"CHAPI dhv11 STARTING line"));

    char init_name[50];
    ::memset(init_name, 0, sizeof(init_name));

    hinst=::LoadLibrary(TEXT((char*)line_dll[0]));

    if(hinst != NULL){
        TRACE((L(1),"DONE LOAD DLL FILE <%s> ", (char*)line_dll[0]));
        ::strcat(init_name, "init_");
        ::strcat(init_name, (char*)line_cfg[0]);
        ::strcat(init_name, "_line");

        pf_init_line_t init_line = (pf_init_line_t)::GetProcAddress(hinst,
init_name);

        if(init_line != NULL){
            TRACE((L(1),"<%s> FOUND", init_name));
            init_line(this, &line[0], 0);
            if(line[0]){
                line[0]->set_extra_command((char*)line_param[0],
EXTRA_LINE_SETUP);
                line[0]->start();
                if((bool)line_is_terminal[0]){
                    line[0]->setup_flow_ctrl(XON_XOFF_FLOW_CONTROL);
                }
                line[0]->setup_char_len(context.char_len);
                line[0]->setup_stop_len(context.stop_len);
                line[0]->setup_parity(context.parity_en);
                line[0]->setup_speed(context.baud);
                unsigned char signals =0;
                if(context.dtr){
                    signals |= D_T_R;
                }
                if(context.rts){
                    signals |= R_T_S;
                }
                line[0]->setup_output_signal(signals);
            }
            else{
                LOGMSG((_ERR_MSG_, LINE_INSTANCE_CRE_ERROR,
                    "UNABLE CREATE LINE[%d] <%s> INSTANCE
",(char*)line_cfg[0]));
            }
        }
        else {
            LOGMSG((_ERR_MSG_, DLL_SYMBOL_NOT_FOUND,
                "UNABLE FIND <%s> IN DLL", init_name));
            get_sys_err(::GetLastError());
        }
    }
    else {
        LOGMSG((_ERR_MSG_, DLL_LOAD_ERROR, "UNABLE LOAD DLL FILE",
(char*)line_dll[0]));
        get_sys_err(::GetLastError());
    }
}
```

```
void chapi_dlv11::stop()
{
    if(line[0]) {
        TRACE((L(1),"CHAPI dhv11 STOPING line"));
        line[0]->stop();

        TRACE((L(1),"DO FREE LINE"));
        delete line[0];
        line[0] = 0;
    }

    if(hinst){
        ::FreeLibrary(hinst);
    }
}



void chapi_dlv11::reset()
{

}



int chapi_dlv11::read(unsigned int addr, bool is_byte)
{
    TRACE((L(5),"CHAPI dlv11 READ SMTH"));

    UWORD temp = 0;

    if(!is_byte){
        switch(addr - b_address) {
        case 0:
            TRACE((L(5),"CHAPI dlv11 READ RCRS"));

            temp = RCSR.value;
            RCSR.data_set_int = 0;
            return temp;

        case 2:
            TRACE((L(5),"CHAPI dlv11 READ RBUF"));

            rx_brq.clear();
            RCSR.rcvr_done = 0;

            if((rx_ring_end - rx_ring_done) != 0) {
                RBUF.value = rx_ring_buf[(rx_ring_done++) %
ring_buf_size].value;
                ci->put_ast(ci, 0, rx_irq_requestor, this, 0);
            }
            return RBUF.value;

        case 4:
            TRACE((L(5),"CHAPI dlv11 READ XCSR=%04X",XCSR.value));

            return XCSR.value;

        case 6:
            TRACE((L(5),"CHAPI dlv11 READ XBUF"));
            break;

        }
    }
    else {
        switch(addr - b_address) {
        case 0:
            TRACE((L(5),"CHAPI dlv11 READ RCSR LO"));

                  temp = RCSR.value;
            RCSR.data_set_int = 0;
            return (temp & 0x00ff);

        case 1:
            TRACE((L(5),"CHAPI dlv11 READ RCSR HI"));

            temp = RCSR.value;
            RCSR.data_set_int = 0;
            return ((temp >> 8) & 0x00ff);

        case 2:
            TRACE((L(5),"CHAPI dlv11 READ RBUF LO"));

            RCSR.rcvr_done = 0;
            rx_brq.clear();

            if((rx_ring_end - rx_ring_done) != 0) {
                RBUF.value = rx_ring_buf[(rx_ring_done++) %
ring_buf_size].value;
                ci->put_ast(ci, 0, rx_irq_requestor, this, 0);
            }
            return (RBUF.value & 0x00ff);

        case 3:
            RCSR.rcvr_done = 0;

            TRACE((L(5),"CHAPI dlv11 READ RBUF HI"));

            return ((RBUF.value >> 8) & 0x00ff);

        case 4:
            TRACE((L(5),"CHAPI dlv11 READ XCSR LO"));

            return (XCSR.value & 0x00ff);

        case 5:
            TRACE((L(5),"CHAPI dlv11 READ XCSR HI"));

            return ((XCSR.value >> 8) & 0x00ff);

        case 6:
            TRACE((L(5),"CHAPI dlv11 READ XBUF LO"));

            break;

        case 7:
            TRACE((L(5),"CHAPI dlv11 READ XBUF HI"));
            break;
        }
    }

    return 0;
}




void chapi_dlv11::write(unsigned int addr, int val, bool is_byte)
{
    const unsigned int def_speed[16]={50,   75,   110,  134,
                                      150,  300,  600,  1200,
                                      1800, 2000, 2400, 3600,
                                      4800, 7200, 9600, 19200};
    unsigned char signal_mask;

   if(!is_byte){
        val &= 0x0000ffff;
        switch(addr - b_address) {
        case 0:
            TRACE((L(5),"CHAPI dlv11 write RCSR, val= %04X",val));

            RCSR.rcvr_int_enb = (val & rcsr_rcvr_int_enb_mask) >> 6;
            RCSR.dset_int_enb = (val & rcsr_dset_int_enb_mask) >> 5;
            RCSR.sec_xmit = (val & rcsr_sec_xmit_mask) >> 3;
            RCSR.rts = (val & rcsr_rts_mask) >> 2;
            RCSR.dtr = (val & rcsr_dtr_mask) >> 1;
            RCSR.rdr_enb = val & rcsr_rdr_enb_mask;

            if(RCSR.rcvr_int_enb && RCSR.rcvr_done && (rx_int_en == false)){
                TRACE((L(5),"TRY SETUP RX IRQ FROM WRITE"));
                rx_brq.set();
            }

            if(RCSR.rcvr_int_enb){
                rx_int_en = true;
            }
            else {
                rx_int_en = false;
            }


            if((context.mode != 'e') ||
                (context.dtr && context.rts)) {
                    break;
            }

            if(RCSR.dset_int_enb && RCSR.data_set_int && (dset_int_en ==
false)){
                TRACE((L(5),"TRY SETUP RX IRQ FROM WRITE (SIGNALS)"));
                rx_brq.set();
            }

            if(RCSR.dset_int_enb){
                dset_int_en = true;
            }
            else {
```

```
            dset_int_en = false;
        }

        if(context.dtr) {
            RCSR.dtr = 1;
        }
        if(context.rts) {
            RCSR.rts = 1;
        }

        signal_mask = 0;

        if (RCSR.dtr) {
            signal_mask |= D_T_R;
        }
        if (RCSR.rts) {
            signal_mask |= R_T_S;
        }

        if(line[0]){
            line[0]->setup_output_signal(signal_mask);
        }

        break;

    case 2:
        TRACE((L(5),"CHAPI dlv11 write RBUF, val= %04X",val));

        break;

    case 4:
        TRACE((L(5),"CHAPI dlv11 write XCSR, val= %04X ",val));


        XCSR.speed = (val & xcsr_pbr_sel_mask) >> 12;
        XCSR.pbr_sel_enb = (val & xcsr_pbr_enb_mask) >> 11;
        XCSR.xmit_int_enb = (val & xcsr_xmit_int_enb_mask) >> 6;
        XCSR.maint = (val & xcsr_maint_mask) >> 2;
        XCSR.start_break = val & xcsr_start_break_mask;

        if(line[0]){
            if(XCSR.pbr_sel_enb && context.baudsel){
                line[0]->setup_speed(def_speed[XCSR.speed]);
            }
            if(context.breaken){
                line[0]->setup_break(XCSR.start_break);
            }
        }


        if(XCSR.maint){
            flag_maint_mode = true;
            if(line[0]){
                line[0]->set_rx_disable();
            }
        }
        else {
            flag_maint_mode = false;
```

```
                if(line[0]){
                    line[0]->set_rx_enable();
                }
            }

            if(XCSR.xmit_int_enb){
                if(XCSR.xmit_rdy && (tx_int_en == false)){
                    TRACE((L(5),"TRY SETUP TX IRQ FROM WRITE"));
                    tx_brq.set();
                }
                tx_int_en = true;
            }
            else {
                tx_int_en = false;
                tx_brq.clear();
            }

            break;

        case 6:
            TRACE((L(5),"CHAPI dlv11 write XBUF, val= %04X ",val));

            XCSR.xmit_rdy = 0;

            tx_brq.clear();

            XBUF.character = val & 0x00ff;

            if(flag_maint_mode){
                if(!rx_ready()){
                    LOGMSG((_ERR_MSG_, RX_BUFFER_FULL, "RX BUFFER IS FULL !!!"
));
                }
                else{
                    rx_ring_buf[rx_ring_end % ring_buf_size].value = 0;
                    rx_ring_buf[rx_ring_end % ring_buf_size].character =
XBUF.character;
                    rx_ring_end++;

                    ci->put_ast(ci, 990, rx_irq_requestor, this, 1);
                    ci->put_ast(ci, 1000, tx_irq_requestor, this, 1);
                }

                break;
            }

            if(tx_ready()) {
                tx_ring_buf[(tx_ring_end++) % ring_buf_size] = XBUF.character;

                if(XBUF.character == 023){
                    TRACE((L(4),"XBUF XOFF"));
                }
                if(XBUF.character == 021){
                    TRACE((L(4),"XBUF XON"));
                }

                if(line[0]){
                    line[0]->do_tx(0);
                }
            }
            else {
                LOGMSG((_ERR_MSG_, TX_BUFFER_FULL, "WRITE:   !!! TX BUFFER IS
FULL !!!"));
            }

            break;
        }
    }
    else {
        val &= 0x000000ff;
        switch(addr - b_address) {
        case 0:
            TRACE((L(5),"CHAPI dlv11 write RCSR LO, val= %04X",val));

            RCSR.rcvr_int_enb = (val & rcsr_rcvr_int_enb_mask) >> 6 ;
            RCSR.dset_int_enb = (val & rcsr_dset_int_enb_mask) >> 5;
            RCSR.sec_xmit = (val & rcsr_sec_xmit_mask) >> 3;
            RCSR.rts = (val & rcsr_rts_mask) >> 2;
            RCSR.dtr = (val & rcsr_dtr_mask) >> 1;
            RCSR.rdr_enb = val & rcsr_rdr_enb_mask;

            if(RCSR.rcvr_int_enb && RCSR.rcvr_done && (rx_int_en == false)){
                TRACE((L(5),"TRY SETUP RX IRQ FROM WRITE"));
                rx_brq.set();
            }

            if(RCSR.rcvr_int_enb){
                rx_int_en = true;
            }
            else {
                rx_int_en = false;
            }

            if((context.mode != 'e') ||
                (context.dtr && context.rts))
            {
                break;
            }

            if(RCSR.dset_int_enb && RCSR.data_set_int && (dset_int_en ==
false)){
                TRACE((L(5),"TRY SETUP RX IRQ FROM WRITE (SIGNAL)"));
                rx_brq.set();
            }

            if(RCSR.dset_int_enb){
                dset_int_en = true;
            }
            else {
                dset_int_en = false;
            }


            if(context.dtr) {
                RCSR.dtr = 1;
            }
            if(context.rts) {
                RCSR.rts = 1;
            }

            signal_mask = 0;

            if (RCSR.dtr) {
                signal_mask |= D_T_R;
            }
            if (RCSR.rts) {
                signal_mask |= R_T_S;
            }

            if(line[0]){
                line[0]->setup_output_signal(signal_mask);
            }


            break;

        case 1:
            TRACE((L(5),"CHAPI dlv11 write RCSR HI, val= %04X",val));
            break;

        case 2:
            TRACE((L(5),"CHAPI dlv11 write RBUF LO, val= %04X",val));
            break;

        case 3:
            TRACE((L(5),"CHAPI dlv11 write RBUF HI, val= %04X ",val));
            break;

        case 4:
            TRACE((L(5),"CHAPI dlv11 write XCSR LO, val= %04X",val));


            XCSR.xmit_int_enb = (val & xcsr_xmit_int_enb_mask) >> 6;
            XCSR.maint = (val & xcsr_maint_mask) >> 2;
            XCSR.start_break = val & xcsr_start_break_mask;

            if(line[0]){
                if(context.breaken){
                    line[0]->setup_break(XCSR.start_break);
                }
            }

            if(XCSR.maint){
                flag_maint_mode = true;
                if(line[0]){
                    line[0]->set_rx_disable();
                }
            }
            else {
                flag_maint_mode = false;
                if(line[0]){
                    line[0]->set_rx_enable();
                }
            }

            if(XCSR.xmit_int_enb){
                if(XCSR.xmit_rdy && (tx_int_en == false)){
                    TRACE((L(5),"TRY SETUP TX IRQ FROM WRITE"));
                    tx_brq.set();
                }
                tx_int_en = true;
            }
            else {
                tx_int_en = false;
                tx_brq.clear();
            }

            break;

        case 5:
            TRACE((L(5),"CHAPI dlv11 write XCSR HI, val= %04X",val));


            XCSR.speed = ((val << 8) & xcsr_pbr_sel_mask) >> 12;
            XCSR.pbr_sel_enb = ((val << 8) & xcsr_pbr_enb_mask) >> 11;

            if (XCSR.pbr_sel_enb && context.baudsel){
                if(line[0]){
                    line[0]->setup_speed(def_speed[XCSR.speed]);
                }
            }

            break;

        case 6:
            TRACE((L(5),"CHAPI dlv11 write XBUF LO, val= %04X",val));


            XCSR.xmit_rdy = 0;

            tx_brq.clear();

            XBUF.character = val;

            if(flag_maint_mode){
                if(!rx_ready()){
                    LOGMSG((_ERR_MSG_, RX_BUFFER_FULL, "RX BUFFER IS FULL
!!!"));
                }
                else{
                    rx_ring_buf[rx_ring_end % ring_buf_size].value = 0;
                    rx_ring_buf[rx_ring_end % ring_buf_size].character =
XBUF.character;
                    rx_ring_end++;

                    ci->put_ast(ci, 990, rx_irq_requestor, this, 1);
                    ci->put_ast(ci, 1000, tx_irq_requestor, this, 1);
                }

                break;
            }

            if(tx_ready()) {
                tx_ring_buf[(tx_ring_end++) % ring_buf_size] = XBUF.character;

                if(XBUF.character == 023){
                    TRACE((L(4),"XBUF XOFF"));
                }
                if(XBUF.character == 021){
                    TRACE((L(4),"XBUF XON"));
                }

                if(line[0]){
                    line[0]->do_tx(0);
                }
            }
            else {
                LOGMSG((_ERR_MSG_, TX_BUFFER_FULL, "WRITE:   !!! TX BUFFER IS
FULL !!!"));
            }

            break;

        case 7:
            TRACE((L(5),"CHAPI dlv11 write XBUF HI, val= %04X",val));

            break;
        }
    }
}


int chapi_dlv11::set_configuration(const char * parameters)
{
        return 0;
}

int chapi_dlv11::set_configuration_ex()
{

    if(line_cfg[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        line_cfg[0].commit();
        line_cfg[0].change_ack();

        TRACE((L(1), "Configuration parameter \"line_cfg\" is set with value
\"%s\"",
            (char*)line_cfg[0]));
    }

    if(line_dll[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        line_dll[0].commit();
        line_dll[0].change_ack();

        TRACE((L(1), "Configuration parameter \"line_dll\" is set with value
\"%s\"",
            (char*)line_dll[0]));
    }
    if(line_param[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        line_param[0].commit();
        line_param[0].change_ack();

        TRACE((L(1), "Configuration parameter \"line_param\" is set with value
\"%s\"",
            (char*)line_param[0]));
    }

    if(line_is_terminal[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        line_is_terminal[0].commit();
        line_is_terminal[0].change_ack();

        TRACE((L(1), "Configuration parameter \"line_is_terminal\" is set with
value \"%d\"",
            (bool)line_is_terminal[0]));
    }


    if(mode[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        mode[0].commit();
        mode[0].change_ack();


        context.mode = *((char*)mode[0]);
        if(context.mode == 'E'){
            context.mode = 'e';
        }

        TRACE((L(1), "Configuration parameter \"mode\" is set with value
\"%c\"",
            *(char*)mode[0]));
    }



    if(char_len[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        char_len[0].commit();
        char_len[0].change_ack();

        context.char_len = (int)char_len[0];
        TRACE((L(1), "Configuration parameter \"char_len\" is set with value
\"%d\"",
            (int)char_len[0]));
    }



    if(stop_len[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        stop_len[0].commit();
        stop_len[0].change_ack();

        context.stop_len = (int)stop_len[0];

        TRACE((L(1), "Configuration parameter \"stop_len\" is set with value
\"%d\"",
            (int)stop_len[0]));
    }



    if(baud[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        baud[0].commit();
        baud[0].change_ack();

        context.baud = (int)baud[0];

        TRACE((L(1), "Configuration parameter \"baud\" is set with value
\"%d\"",
            (int)baud[0]));
    }



    if(baudselen[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        baudselen[0].commit();
        baudselen[0].change_ack();

        context.baudsel = (bool)baudselen[0];

        TRACE((L(1), "Configuration parameter \"baudselen\" is set with value
\"%d\"",
            (bool)baudselen[0]));
    }
    if(breaken[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        breaken[0].commit();
        breaken[0].change_ack();

        context.breaken = (bool)breaken[0];

        TRACE((L(1), "Configuration parameter \"breaken\" is set with value
\"%d\"",
            (bool)breaken[0]));
    }




    if(parity[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        parity[0].commit();
        parity[0].change_ack();

        char temp[50];
        ::memset(temp, 0, sizeof(temp));
        ::strcpy(temp, (char*)parity[0]);
        int len = (int)strlen(temp);
        temp[len++] = ' ';
        ::_strlwr(temp);


        if(::strstr(temp,"none")){
            context.parity_en = PARITY_CONTROL_NONE;
        }

        if(::strstr(temp,"even")){
            context.parity_en = PARITY_CONTROL_EVEN;
        }

        if(::strstr(temp,"odd")){
            context.parity_en = PARITY_CONTROL_ODD;
        }

        if(::strstr(temp,"mark")){
            context.parity_en = PARITY_CONTROL_MARK;
        }

        if(::strstr(temp,"space")){
            context.parity_en = PARITY_CONTROL_SPACE;
        }


        TRACE((L(1), "Configuration parameter \"parity\" is set with value
\"%s\"",
            (char*)parity[0]));
    }



    if(dtr[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        dtr[0].commit();
        dtr[0].change_ack();

        context.dtr = (bool)dtr[0];

        TRACE((L(1), "Configuration parameter \"dtr\" is set with value \"%d\"",
            (bool)dtr[0]));
    }




    if(erroren[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        erroren[0].commit();
        erroren[0].change_ack();

        context.erroren = (bool)erroren[0];

        TRACE((L(1), "Configuration parameter \"erroren\" is set with value
\"%d\"",
            (bool)erroren[0]));
    }



    if(rts[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        rts[0].commit();
        rts[0].change_ack();

        context.rts = (bool)rts[0];

        TRACE((L(1), "Configuration parameter \"rts\" is set with value \"%d\"",
            (bool)rts[0]));
    }

    return 0;
}

int chapi_dlv11::rx_done(unsigned int len, unsigned char line_id)
{

    TRACE((L(5),"rx_done -> put_ast"));
    ci->put_ast(ci, 990, rx_irq_requestor, this, len);
    return 0;
}


int chapi_dlv11::tx_done(unsigned int len, unsigned char line_id)
{
    TRACE((L(5),"tx_done -> put_ast"));
    ci->put_ast(ci, 1000, tx_irq_requestor, this, len);
    return 0;
}


int chapi_dlv11::get_tx_char(unsigned char & from_buf, unsigned char line_id)
{
    if(((tx_ring_end - tx_ring_done) % ring_buf_size) == 0){
        return -1;
    }

    from_buf = tx_ring_buf[tx_ring_done % ring_buf_size];

    tx_ring_done++;
    tx_ring_start++;

    return 1;
}



int chapi_dlv11::input_signal(unsigned char in_signal, unsigned char line_id)
{
    if(context.mode != 'e'){
        return 0;
    }

    ci->put_ast(ci, 1, signal_irq_requestor, this, in_signal);
    return 0;
}


int chapi_dlv11::error_tx(unsigned char error, unsigned char line_id)
    {
        return 0;
    }


int chapi_dlv11::error_rx(unsigned char error, unsigned char line_id)
{
    if(!context.erroren){
        return 0;
    }

    ci->put_ast(ci, 0, put_errors_in, this, error);
    return 0;
}



int chapi_dlv11::extra(void * extra, int arg, unsigned char line_id)
```

```
    {
        return 0;
    }



void chapi_dlv11::get_sys_err( DWORD dwErr )
{
    char Message[2000];
    //TRACE((L(1), "GetSystemErrorMessage(%d)", dwErr));

    LPTSTR lpszTemp = NULL;
    DWORD dwRet = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER|
        FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_ARGUMENT_ARRAY,
        NULL, dwErr, LANG_NEUTRAL, (LPTSTR)&lpszTemp, 0, NULL);

    if(!dwRet) {
        LOGMSG((_ERR_MSG_, PROBLEM_DETECT_ERROR, "Unable to detect the
problem"));
        sprintf( Message, "0x%X\n", dwErr );
    }
    else {
        // Remove CR and LF character
        lpszTemp[strlen(lpszTemp)-2] = '\0';
        sprintf( Message, "Error: %s (0x%X)\n", lpszTemp, dwErr );
    }

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

    LOGMSG((_ERR_MSG_, SYSTEM_ERROR, Message));

    return;
}


//
// This method is a part of newly developed in CHAPI v. 2.0 concept of
// CHAPI device IRQ processing. This is a place where populated BRQs
// should be connected to the bus.
//
void chapi_dlv11::setup_bus_requests()
{
    TRACE((L(1), "setup_bus_requests() RX: ci->base_i_vector = 0%o",
        ci->base_i_vector));

    TRACE((L(1), "setup_bus_requests() TX: ci->base_i_vector = 0%o",
        ci->base_i_vector+4));

    if(!rx_brq.connect(ci, ci->base_i_vector)) {
        LOGMSG((_ERR_MSG_, BRQ_CONNECT_ERROR,
            "Unable to connect RX BRQ to the bus."));
    }

    if(!tx_brq.connect(ci, ci->base_i_vector+4)) {
        LOGMSG((_ERR_MSG_, BRQ_CONNECT_ERROR,
            "Unable to connect TX BRQ to the bus."));
    }
}
```

```
// ------------------------------------------------------------------
//
// A number of wrappers to be passed to CHARON
//
#define CALL(co, func, arglist) ((chapi_dlv11 *)(co->context))->func arglist

static void CHAPI chapi_dlv11_start(const struct __chapi_out * co) {
    CALL(co, start, ());
}

static void CHAPI chapi_dlv11_stop(const struct __chapi_out * co) {
    CALL(co, stop, ());
}

static void CHAPI chapi_dlv11_reset(const struct __chapi_out * co) {
    CALL(co, reset, ());
}

static int CHAPI chapi_dlv11_read(const struct __chapi_out * co,
    unsigned int addr, bool is_byte)
{
    return CALL(co, read, (addr, is_byte));
}

static void CHAPI chapi_dlv11_write(const struct __chapi_out * co,
    unsigned int addr, int val, bool is_byte)
{
    CALL(co, write, (addr, val, is_byte));
}

static int CHAPI chapi_dlv11_set_configuration(const struct __chapi_out * co,
    const char * parameters)
{
    return CALL(co, set_configuration, (parameters));
}

static int CHAPI chapi_dlv11_set_configuration_ex(const struct __chapi_out * co)
{
    return CALL(co, set_configuration_ex, ());
}

static void CHAPI chapi_dlv11_setup_bus_requests(const struct __chapi_out * co)
{
    return CALL(co, setup_bus_requests, ());
}



// ------------------------------------------------------------------

//
// Initialization routine
//
CHAPI_INIT(CHAPI_DLV11)(const chapi_in *ci, chapi_out *co,
    const char *instance_name)
{
    if(ci == NULL) {
        // We have no possibility to log anything here...
        return 0;
    }

    if((ci->put_irq  == 0) || (ci->read_mem == 0) || (ci->put_ast  == 0) ||
        (ci->put_sst  == 0))
    {
        LOGMSG((_ERR_MSG_, INPUT_CONTEXT_INVALID,
            "CHARON didn't provide us one of the following procedures:\n"
                  "\tread_mem\n"
                  "\tput_ast\n"
                  "\tput_sst"
                  "Giving up"
        ));

        return 0;
    }

    chapi_dlv11 *dev = new chapi_dlv11(ci, instance_name);

    if(dev == NULL) {
        LOGMSG((_ERR_MSG_, DEV_CREATION_ERROR,
            "cannot create chapi_dlv11 instance. Giving up"));
        if(dev != NULL) {
            delete dev;
        }
        return 0;
    }


    co->context = dev;
    co->base_b_address = DEFAULT_ADDRESS;
    co->b_address_range = 8;
    co->base_i_vector = DEFAULT_VECTOR;
    co->n_of_i_vector = 2;
    co->i_priority = 4;
    co->supported_buses = QBUS | UNIBUS;
    co->start = chapi_dlv11_start;
    co->stop = chapi_dlv11_stop;
    co->reset = chapi_dlv11_reset;
    co->read = chapi_dlv11_read;
    co->write = chapi_dlv11_write;
    co->set_configuration = chapi_dlv11_set_configuration;
    co->set_configuration_ex = chapi_dlv11_set_configuration_ex;
    co->setup_bus_requests = chapi_dlv11_setup_bus_requests;

    return dev;
}

#undef _ERR_MSG_
#undef _WARN_MSG_
#undef _INFO_MSG_
#undef LOGMSG
#undef L
#undef TRACE
```

1.2.2. LPV11

```
//
// Copyright (C) 1999-2006 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(VAXPRINT)

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

#endif

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>

// + CHAPI protocol
// + messaging
// + class chapi_brq_t
#include <chapi_lib.h>

// Error codes for LPV11.DLL
#include "lpv11_msgid.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"


#if !defined(_DEVELOPMENT)

//
// Undefine this and rebuild in order to avoid license key serial printings
// each 25 lines printed.
//
#define _PRINT_PRODUCT_INFO_STR

#endif // defined(_DEVELOPMENT)

#if defined(_PRINT_PRODUCT_INFO_STR)
```

```
    // License serial number string to print at the beginning of the first page.
    static char product_info_str[255];
    static unsigned long lines_counter = 0;
#endif


//
// 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 {
public:

    //------------------------------------------------------------------------

    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_ex();

    void setup_bus_requests();


    //------------------------------------------------------------------------
    // Additional options for LPV11 device

    // Application to start as data consumer
    chapi_string_option_t m_AppCmd;

    // TCP/IP connection to be used for data spooling
    chapi_string_option_t hostname;
    chapi_integer_option_t portnum;

    // Output file/device name - checked if TCP/IP connection isn't specified
    chapi_string_option_t outfile_name;

protected:

    void GetSystemErrorMessage( DWORD dwErr );

    //------------------------------------------------------------------------

    // start VAXPrint using LPV11 command parameters
    bool bStartApp;

#if defined(VAXPRINT)
```

```
    // HOSTPrint startup support
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    // HOSTPrint connection
    SOCKET sock;
    SOCKET connection_sock;
#endif

    char Message[256];

    // TCPIP usage flag
    bool UseTCPIP;

private:

    // Bus request
    chapi_brq_t brq;

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

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

    // 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);

    static int brq_ack(lpv11 *the_lpv11);

    //-----------------------------------------------------------------------------

    const chapi_in *ci;     // pointer to CHARON-supplied structure

    //
    // 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

    // LPCS & LPDB bits definitions
    enum {
        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;

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

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

    enum {
        // Working thread is waiting for this event to process data
        wrk_event = 0,

        // Auxiliary event used by the working thread
        io_event,

        // Auxiliary event used to inform working thread that it should cleanup
I/O
        cancel_event,

        // This event is used to terminate working thread
        termination_event,

        // Events count
        total_events
    };

    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
};
```

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

lpv11::lpv11(const chapi_in *_ci, const char *instance_name)
: brq(0x04, (__chapi_brq_acknowledge_p)brq_ack, (void*)this)
, m_AppCmd(_ci, "application", 1, 1024)
, hostname(_ci, "host", 1, 256)
, portnum(_ci, "port", 1)
, outfile_name(_ci, "file", 1, _MAX_PATH)
{
    // Remember communication context
    ci = _ci;

    TRACE((L(1), "lpv11(%x, %s)", _ci, instance_name));

    // Initialize the state of device here
    UseTCPIP = false;
    bStartApp = false;

    int_enabled = false;
    error = false;
    sst_pending = false;
    ring_start = ring_done = ring_end = 0;

    //
    // In order do not keep instance_name somewhere else - just set it as
    // default value for outfile_name option.
    //
    outfile_name[0].set(instance_name);
    outfile_name[0].commit();
    outfile_name[0].change_ack();
}

// ---------------------------------------------------------------------------

lpv11::~lpv11()
{
    TRACE((L(1), "~lpv11()"));

    // Stop device if it is running ...
    if(valid()) {
        stop();
    }
}

// ---------------------------------------------------------------------------

bool lpv11::valid()
{
    // Make sure that required object created
    // ... Events
    for(int i = 0; i < total_events; i++) {
        if(events[i] == NULL)
            return false;
    }

    // ... working thread
    if(wrk_thread == NULL) {
        return false;
    }

    // ... and output file
    //if(outfile_name == NULL) {
    //    return false;
    //}

    return true;
}

// ---------------------------------------------------------------------------

void lpv11::start()
{
    TRACE((L(1), "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;

    // Enable bus request here - brq.connect() will not enable request
    // automatically
    brq.enable();

    // Create all synchronization events
    for(int i = 0; i < total_events; i++) {
        events[i] = CreateEvent(NULL, FALSE, FALSE, NULL);
        if(events[i] == NULL) {
            LOGMSG((_ERR_MSG_, CREATE_EVENT_ERROR,
                "Cannot create event: error = %d",
                GetLastError()));
        }
    }

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

    //
    // Create local output file if TCP/IP connection to HOSTPrint application
    // isn't set.
    //
    if(!UseTCPIP ) {
        TRACE((L(3), "TCP/IP connection isn't set, use local file."));

        if(strlen(outfile_name[0])) {
            TRACE((L(3), "%s output file will be used by LPV11",
(char*)outfile_name[0]));

            outfile = CreateFile(outfile_name[0], 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 {
                LOGMSG((_ERR_MSG_, FILE_OPEN_ERROR,
                    "Cannot open file : error = %d",
                    GetLastError()));
            }
        }

    }

    // This is the right place to start working thread
    wrk_thread = CreateThread(NULL, 0, wrk_thread_proc, this, 0, NULL );
    if(wrk_thread == NULL) {
        LOGMSG((_ERR_MSG_, CREATE_THREAD_ERROR,
            "Cannot create thread: error = %d",
            GetLastError()));
    }
}

// ---------------------------------------------------------------------------

void lpv11::stop()
{
    TRACE((L(1), "stop()"));

    if(UseTCPIP) {
#if defined(VAXPRINT)
        TRACE((L(3), "Try to terminate HOSTprint application"));

        // We have to terminate HOSTPrint application
        if(pi.hProcess) {
            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();
#endif
    }
    else {
        // We have to close output file ...
        if(strlen(outfile_name[0])) {
            TRACE((L(3), "Close output file %s", outfile_name));

            DeleteFile( outfile_name[0] );
            //outfile_name[0] = '\0';
        }

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

    // request working thread to cancel I/O
    SetEvent(events[cancel_event]);
    // And close the output
    if( !UseTCPIP ) {
        if(outfile != INVALID_HANDLE_VALUE) {
            CloseHandle(outfile);
            outfile = INVALID_HANDLE_VALUE;
        }
    }

    // Terminate working thread if any
    if(wrk_thread != NULL) {
        TRACE((L(3), "Try to terminate working thread..."));

        SetEvent(events[termination_event]);
        WaitForSingleObject(wrk_thread, INFINITE);
        wrk_thread = NULL;

        TRACE((L(3), "Working thread successfully terminated..."));
    }

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

// ----------------------------------------------------------------------------

void lpv11::reset()
{
    TRACE((L(1), "reset()"));

    int_enabled = false;
    error = false;
}

// ----------------------------------------------------------------------------

int lpv11::read(unsigned int addr, bool is_byte)
{
    TRACE((L(2), "read(%x, %s)",
        addr, is_byte ? "true" : "false"));

    // Construct value of LPCS register
    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)
{
    TRACE((L(2), "write(%x, %x, %s)",
        addr, val, is_byte ? "true" : "false"));

    // Do nothing if attempting to write to high-order byte of LPCS/LPDB
    switch(addr - b_address) {

    // LPCS: only bother about interrupt enable/disable
    case 0:
        if(val & lpcs_int) {
            TRACE((L(3), "Interrupts enabled."));
            int_enabled = true;
        } else {
            TRACE((L(3), "Interrupts disabled."));
            int_enabled = false;
        }
        break;

    // LPDB: use lower 7 bits only. If not ready, do nothing.
    case 2:
        if(ready()) {
            ring_buf[ring_end++ % ring_buf_size] = val & lpdb_data;

            // Ring buffer is full => kicj working thread immediately
            if(!ready()) {
                kick_wrk_thread();
            }
            // Else, kick it a bit later... if not already directed so
            else if(!sst_pending) {
                sst_pending = true;
                ci->put_sst(ci, kick_delay, kick_wrk_thread, this, 0);
            }
        }
        break;
    }
}


// ----------------------------------------------------------------------------


//
// This method is a part of newly developed in CHAPI v. 2.0 concept of
// CHAPI device configuration. See details in _INIT method for LPV11 device.
//
int lpv11::set_configuration_ex()
{
    TRACE((L(1), "set_configuration_ex()"));


    // Check for the 'host' parameter changes
    if(hostname[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        hostname[0].commit();
        hostname[0].change_ack();
        TRACE((L(3), "Configuration parameter \"host\" is set with value
\"%s\"",
            (char*)hostname[0]));
    }

    // Check for the 'port' parameter changes
    if(portnum[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        portnum[0].commit();
        portnum[0].change_ack();

        // If port is specified - we will use TCP/IP
        UseTCPIP = true;

        TRACE((L(3), "Configuration parameter \"port\" is set with value %d",
            (int)portnum[0]));
    }

    // Check for the 'application' parameter changes
    if(m_AppCmd[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        m_AppCmd[0].commit();
        m_AppCmd[0].change_ack();

        if( strlen(m_AppCmd[0]) ) {
            bStartApp = true;
        }

        TRACE((L(3), "Configuration parameter \"application\" is set with value
\"%s\"",
            (char*)m_AppCmd[0]));
    }

    // Check for the 'application' parameter changes
    if(outfile_name[0].is_changed()) {
        //
        // Ok, we have to commit/acknowledge change option in order to have
        // valid value here...
        //
        outfile_name[0].commit();
        outfile_name[0].change_ack();

        TRACE((L(3), "Configuration parameter \"outfile\" is set with value
\"%s\"",
            (char*)outfile_name[0]));
    }

    return 0;
}

//
// This method is a part of newly developed in CHAPI v. 2.0 concept of
// CHAPI device IRQ processing. This is a place where populated BRQs
// should be connected to the bus.
```

```
//
void lpv11::setup_bus_requests()
{
    TRACE((L(1), "setup_bus_requests() : ci->base_i_vector = 0%o",
        ci->base_i_vector));

    if(!brq.connect(ci, ci->base_i_vector)) {
        LOGMSG((_ERR_MSG_, BRQ_CONNECT_ERROR, "Unable to connect BRQ to the
bus."));
    }
}

// -----------------------------------------------------------------

void lpv11::GetSystemErrorMessage( DWORD dwErr )
{
    TRACE((L(1), "GetSystemErrorMessage(%d)", dwErr));

    LPTSTR lpszTemp = NULL;
    DWORD dwRet = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER|
        FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_ARGUMENT_ARRAY,
        NULL, dwErr, LANG_NEUTRAL, (LPTSTR)&lpszTemp, 0, NULL);

    if(!dwRet) {
        LOGMSG((_ERR_MSG_, PROBLEM_DETECT_ERROR, "Unable to detect the problem"
));
        sprintf( Message, "0x%X\n", dwErr );
    }
    else {
        // Remove CR and LF character
        lpszTemp[strlen(lpszTemp)-2] = '\0';
        sprintf( Message, "Communication error: %s (0x%X)\n", lpszTemp, dwErr );
    }

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

    LOGMSG((_ERR_MSG_, SYSTEM_ERROR, Message));

    return;
}

// ---------------------------------------------------------------------------

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

// ---------------------------------------------------------------------------

DWORD lpv11::wrk_thread_proc()
{
    TRACE((L(1), "wrk_thread_proc()"));

    if( UseTCPIP ) {
```

```
#if defined(VAXPRINT)
        int rc = 0;
        int so_true = 1;
        WSADATA info;

        TRACE((L(3), "Try to start HOSTprint and connect to its socket"));

        // Initialize windows sockets
        if(WSAStartup(MAKEWORD(2,0), &info)) {
            LOGMSG((_ERR_MSG_, WINSOCK_INIT_ERROR, "Socket: WSAStartup error :
%d",
                ::WSAGetLastError()));
            return( FALSE );
        }

        // Create socket and set its parameters
        sock = ::WSASocket( AF_INET, SOCK_STREAM, 0, 0, 0, WSA_FLAG_OVERLAPPED
);
        if( sock == INVALID_SOCKET ) {
            rc = ::WSAGetLastError();
            LOGMSG((_ERR_MSG_, CREATE_SOCKET_ERROR,
                "Socket: Can't create socket : %d", rc ));
            return( rc );
        }

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

        sockaddr_in sa;
        memset( &sa, 0, sizeof(sa) );
        sa.sin_family = PF_INET;
        sa.sin_port = htons((short)portnum[0]);

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

        if(listen(sock, SOMAXCONN) == SOCKET_ERROR ) {
            rc = ::WSAGetLastError();
            closesocket( sock );
            LOGMSG((_ERR_MSG_, SOCKET_LISTEN_ERROR, "Socket: Can't listen: %d",
rc));
            return( rc );
        }

        // Start HOSTprint
        ZeroMemory( &si, sizeof(si) );
        si.cb = sizeof(si);
        ZeroMemory( &pi, sizeof(pi) );

        if(bStartApp) {
            if(!CreateProcess( NULL, m_AppCmd[0], NULL, NULL, FALSE, 0, NULL,
NULL,
                &si, &pi ) )
            {
                rc = GetLastError();
                LOGMSG((_ERR_MSG_, APP_START_ERROR,
                    "HOSTPrint start failed: %d", rc));
                closesocket( sock );
                return rc;
            }
        }

        connection_sock = accept( sock, NULL, NULL );
        if( connection_sock == INVALID_SOCKET ) {
            rc = ::WSAGetLastError();
            if( rc != WSAEWOULDBLOCK ) {
                closesocket( sock );
                LOGMSG((_ERR_MSG_, SOCKET_ACCEPT_ERROR,
                    "Socket: Can't accept: %d", rc));
                return( rc );
            }
        }
#endif
    }

    // We don't want more then 1 I/O operation at a time
    bool io_pending = false;

    // Number of bytes written by the last operation
    DWORD written;

    TRACE((L(3), "Start event driven working loop."));

    for(;;) {
        DWORD signaled_object = WaitForMultipleObjects(total_events, events,
            FALSE, INFINITE);
        switch(signaled_object - WAIT_OBJECT_0) {

        // Cancel pending I/O
        case cancel_event:
            TRACE((L(4), "Cancel pending I/O."));

            CancelIo(outfile);

            // Cleanup ring buffer
            ring_done = ring_end;
            ci->put_ast(ci, 0, set_ring_start, this, ring_end);
            continue;

        // I/O completed
        case io_event:
            TRACE((L(4), "I/O completed."));

            ovl.Offset += written;
            io_pending = false;
            // fall through

        // We have something to work on...
        case wrk_event:
            TRACE((L(4), "We have something to work on..."));

            if(io_pending) {
                continue;
            }

            unsigned int num_to_print = ring_end - ring_done;
            size_t ext_size = 0;

            TRACE((L(4), "We have %d bytes for processing in the ring buffer.",
                num_to_print));

            if(!num_to_print) {
                continue;
            }

#if !defined(_PRINT_PRODUCT_INFO_STR)
            // 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];
            }
#else
            //
            // We will print license information string each 50 lines
            // Allocate contiguous buffer to use in WriteFile
            //
            char *outbuf = new char [num_to_print + 1024/25 *
                strlen(product_info_str) + 1];

            //
            // Copy data from the ring buffer, insert license text
            // each 25 lines...
            //
            size_t ob_pos = 0;
            if(lines_counter == 0 && num_to_print != 0) {
                strcpy(outbuf, product_info_str);
                ob_pos += strlen(product_info_str);
            }

            for(unsigned int i = 0; i < num_to_print; i++) {
                char c = ring_buf[(ring_done + i) % ring_buf_size];
                outbuf[ob_pos++] = c;
                if(c == '\n') {
                    if((++lines_counter %  25) == 0) {
                        strcpy(&outbuf[ob_pos], product_info_str);
                        ob_pos += strlen(product_info_str);
                    }
                }
            }
            ext_size = ob_pos - num_to_print;

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

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

                    connection_sock = accept( sock, NULL, NULL );
                    if( connection_sock == INVALID_SOCKET ) {
                        LOGMSG((_ERR_MSG_, SOCKET_LISTEN_ERROR,
                            "Socket: Can't listen: %d", ::WSAGetLastError()));
                        error = true;
                    }
                }
#endif
            }
            else {
                if(! WriteFile(outfile, outbuf, num_to_print + ext_size,
                    &written, &ovl))
                {
                    if(GetLastError() != ERROR_IO_PENDING) {
                        LOGMSG((_ERR_MSG_, WRITE_FILE_ERROR,
                            "WriteFile failed: %d", GetLastError()));

                        // IRQ will be requested a few lines later anyway
                        error = true;
                    }
                    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;
    }

    TRACE((L(1), "wrk_thread_proc() termination."));

    return 0;
}

// --------------------------------------------------------------------------

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

// ---------------------------------------------------------------------------

void lpv11::kick_wrk_thread()
{
    TRACE((L(1), "kick_wrk_thread()"));

    // Wake up working thread
    SetEvent(events[wrk_event]);
}

// ---------------------------------------------------------------------------

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

// ---------------------------------------------------------------------------

void lpv11::irq_requestor(unsigned int delta)
{
    TRACE((L(1), "irq_requestor(%d)", 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) {
        TRACE((L(3), "Set BRQ"));
        brq.set();
    }
}

// ---------------------------------------------------------------------------

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

// ---------------------------------------------------------------------------

void lpv11::set_ring_start(unsigned int new_ring_start)
{
    TRACE((L(1), "set_ring_start(%d)",
        new_ring_start));

    ring_start = new_ring_start;
}

int lpv11::brq_ack(lpv11 * the_lpv11) {
    the_lpv11->brq.clear();
    return (the_lpv11) ? the_lpv11->i_vector : -1;
}

// ---------------------------------------------------------------------------
//
// 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));
}

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_ex(const struct __chapi_out * co)
{
    return CALL(co, set_configuration_ex, ());
}

static void CHAPI lpv11_setup_bus_requests(const struct __chapi_out * co)
{
    return CALL(co, setup_bus_requests, ());
}

// ---------------------------------------------------------------------------
//
// Initialization routine
//
CHAPI_INIT(LPV11)(const chapi_in *ci, chapi_out *co, const char *instance_name)
{
    // Check context
    if(ci == NULL) {
        //
        // It's even not possible to log message in this case, let CHARON kernel
        // to log some error code because of our creation failure ...
        //
        return 0;
    }
    //
    // Compose license string which should be printed at the beginning of
    // each page here.
    //
#if defined(_PRINT_PRODUCT_INFO_STR)

    unsigned int hl_serial;
    if(ci->get_license_no &&
       ci->get_product_ident &&
       ci->get_hardware_name &&
       ci->get_hardware_model &&
       ci->get_product_major_version &&
       ci->get_product_minor_version &&
       ci->get_product_build_version &&
       ci->get_product_copyright &&
       ci->get_product_custom_string &&
       ci->get_chapi_major_version &&
       ci->get_chapi_minor_version)
    {
        ci->get_license_no(ci, &hl_serial);
        sprintf(product_info_str, "%s (%s [%s]), V %d.%d B %d / %d / CHAPI %d.%d
(%s/%s)\n\n",
            ci->get_product_ident(ci),
            ci->get_hardware_name(ci),
            ci->get_hardware_model(ci),
            ci->get_product_major_version(ci),
            ci->get_product_minor_version(ci),
            ci->get_product_build_version(ci),
            hl_serial,
            ci->get_chapi_major_version(ci),
            ci->get_chapi_minor_version(ci),
            ci->get_product_copyright(ci),
            ci->get_product_custom_string(ci));
    }
    else {
        sprintf(product_info_str, "PROBLEMS WITH CHAPI_IN ENCOUNTERED!!!\n\n");
    }

#endif

    TRACE((L(1), "LPV11_INIT(%x, %x, %s)",
        ci, co, instance_name));

#if defined(_PRINT_PRODUCT_INFO_STR)
    TRACE((L(1), "product_info_str = %s",
        product_info_str));
#endif

    if((ci->put_irq == 0) || (ci->put_ast == 0) || (ci->put_sst == 0)) {
        LOGMSG((_ERR_MSG_, INPUT_CONTEXT_INVALID,
            "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) {
        LOGMSG((_ERR_MSG_, DEV_CREATION_ERROR,
            "Cannot create LPV11 instance. Giving up"));
        if(dev != NULL) {
            delete dev;
        }
        return 0;
    }

    // Prepare output context with specified callbacks finally ...
    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->supported_buses = QBUS|UNIBUS;
    co->start = lpv11_start;
    co->stop = lpv11_stop;
    co->reset = lpv11_reset;
    co->read = lpv11_read;
    co->write = lpv11_write;
    co->set_configuration_ex = lpv11_set_configuration_ex;
    co->setup_bus_requests = lpv11_setup_bus_requests;

    // ... and return created device instance.
    return dev;
}
```

Reader's Comments

We appreciate your comments, suggestions, criticism and updates of this manual. You can Email us your comments at:

vaxinfo@vaxemulator.com

Please mention the document reference number: 30-09-006

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