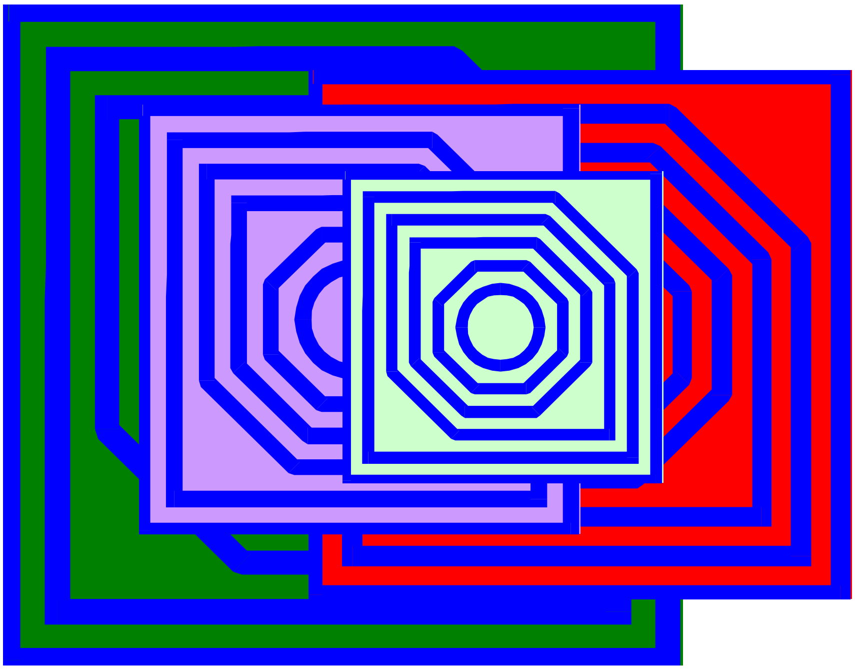
CHARON-11

Application Interface Manual (CHAPI)



SRI
Software Resources
International

CHARON-11

Application Interface Manual (CHAPI)

Document 10-01-040

www.softresint.com

We greatly acknowledge the development of this manual and the practical application of the CHARON-11 Application Interface by Eduardo Marcelo Serrat of Galmes y Casale S.R.L., Argentina.

Document number: 10-01-040

20 February 2002

Printed in Switzerland

Contents

Preface

The PDP-11 processor family designed by Digital Equipment Corporation during the 1970’s and 1980’s became a landmark in the history of computing. In decades of development and sales these systems found their way into virtually every application of computer technology.

It is a tribute to the original PDP-11 designers that their products survive the company the company who developed them by so many years, in an industry where product life cycles are now measured in months. Even a well-designed PDP-11 will eventually stop working due to mechanical or electrical hardware failure, but its design lives on in CHARON-11, a PDP-11 system emulator developed by Software Resources International.

Although Charon-11 provides a wide range of PDP-11 emulated peripherals, it has the capability to add support for peripheral devices that are not part of its standard component library. One way is to connect an existing Qbus or UNIBUS back plane to the emulator via a bus adapter. Another method is to emulate a peripheral on the host system in software and connect it to the emulated peripheral bus. This feature of CHARON-11 is achieved through its Application Programming Interface named CHAPI.

This manual explains you how to use CHAPI for developing additional CHARON-11 peripheral support. It provides a number of examples and explains the individual software calls. To be successful in developing a peripheral using CHAPI you need to be familiar with the PDP-11 architecture, have access to the technical documentation of the peripheral you want to emulate and have the capability to write real-time applications on the Windows platform.

Chapter 2 describes the application of CHAPI by showing in detail the code implementation of a KW11-L line clock device.

Appendix A contains detailed functions reference.

Related information:

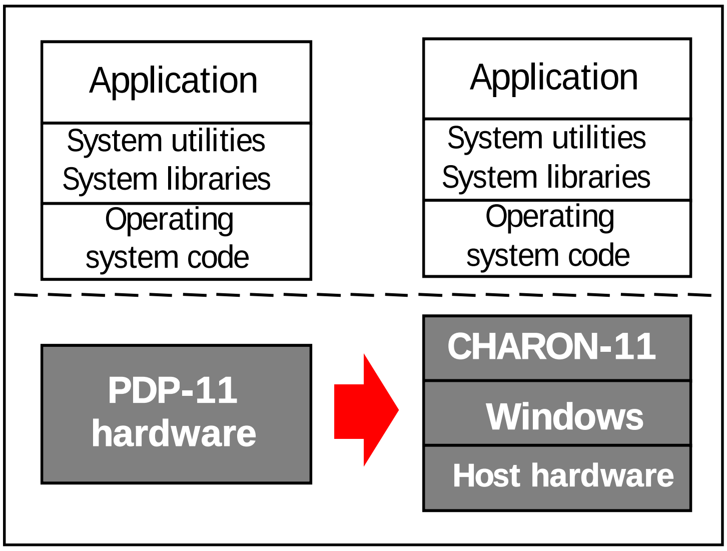
http://www.montagar.com/~patj/dec/hcps.htm points to several PDP-11 technical manuals.

Conventions

Throughout this manual these conventions are followed:

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

Chapter 1 Overview



CHARON-11 emulates the hardware of a PDP-11 system. It provides a software model of a PDP-11 CPU, memory, disks, tapes, serial lines, other controllers and its system bus (Qbus or UNIBUS).

CHAPI, the CHARON-11 Application Interface is a method to connect a user written emulated peripheral controller to the emulated PDP-11 bus. The user written device is implemented as a Visual C++ WIN32 DLL. This DLL can be loaded with the CHARON-11 command console or in the configuration file.

Every PDP-11 controller has a specific bus interface through which it communicates with the rest of the PDP-11. The controller functionality is typically described in its User Guide. The User Guide shows the CSRs addresses and bit field descriptions, Interrupt VECTORS, DMA features (if present) and describes how to program the device.

In order to be recognizable by the PDP-11 software, the user written controller emulation must present the same CSRs, VECTORs and DMA features of its hardware equivalent. A full understanding of the device operation should be acquired before proceeding to implement an emulation of it using CHAPI.

Chapter 2 CHAPI structure

A user written emulated device is implemented as a Visual C++ WIN32 DLL. At the core of CHAPI, there are two important structures, chapi_in and chapi_out, defining pointers to functions. These structures are shared by CHARON-11 and the user written DLL.

2.1 The chapi_in structure

Chapi_in contains pointers to emulator filled-in internal functions and values for the CSR and VEC addresses:

```
typedef struct {
   size_t  reg_addr;
   size_t  reg_win_size;
   int       vector;
   void    (CHAPI* put_ast)(ast_handler func, void* parm);
   void    (CHAPI* put_sst)(long delay, int tag,
                          sst_fun fun, void* arg1, int arg2);
   void    (CHAPI* put_irq)    (int vec, int pri, int delay_cycles,
                                    irq_fun fun, int arg);
   void    (CHAPI* clear_irq) (int vec);
   size_t  (CHAPI* read_mem)          (size_t adr, size_t len, char* buf);
   size_t  (CHAPI* write_mem)         (size_t adr, size_t len, const char* buf);
} chapi_in;
```

The user written DLL calls the functions provided by this structure.

2.2 The chapi_out structure

Chapi_out contains pointers to DLL's filled-in functions that are called by CHARON-11 emulator when accessing the emulated device:

```
typedef struct {
   void    (CHAPI* stop) (void* instance);
   void    (CHAPI* start)(void* instance);
   void    (CHAPI* reset)(void* instance);
   void    (CHAPI* write)(void* instance, size_t addr, int value,
                               int is_byte_access);
int       (CHAPI* read) (void* instance, size_t addr, int
          is_byte_access);
int          (CHAPI* set_configuration)(void* instance, char*
                       parameters);
} chapi_out;
```

The user written DLL must export one symbol, the name of the Initialization function that will receive the chapi_in structure and send out the chapi_out structure properly filled with the corresponding function pointers.

The DLL you create runs in a thread separate from Charon-11. Depending on the device you are trying to emulate, you can create a single thread or a multithreaded DLL for it.

There are two kinds of emulated devices: those that do not require mapping and those that map an emulated device functionality onto a host resource.

The first ones are very simple devices like the KW11-L line clock and are written using single threaded DLLs. The second ones are more complex and are typically implemented as multithreaded DLLs. Disk subsystems, Communication devices like Ethernet Adapters, Sync Adapters, Serial Multiplexers, etc. fall into this category.

Use of CHAPI is illustrated in the following chapter by showing in detail the code implementation of a KW11-L line clock device.

2.3 CHAPI performance considerations.

As the CHARON CPU thread uses up to 100% of the host CPU cycles normally the CPU threads run on a lower priority than device priority. This enables device threads to handle near to real-time activity.

In some circumstances as CHARON-11 runs as a NORMAL_PRIORITY_CLASS process with the main thread at THREAD_PRIORITY_NORMAL this can affect CHARON-11 performance when used along with CHAPI modules, particularly when CHARON-11 is not the foreground process. The actual foreground process receives a priority boost that affects CHARON-11 as a background process. Thus CHAPI Modules threads that execute I/O, may also receive Priorities Boost that surpasses CHARON's. This can cause problems.

To avoid such problems consider changing the CHARON-11 Main Thread Priority to THREAD_PRIORITY_HIGHEST. With this priority, foreground priority boosting and CHAPI Threads running at THREAD_PRIORITY_NORMAL do not affect CHARON11.

With this approach, CHARON-11 and the CHAPI DZ11 modules work well.

Chapter 3 KW11-L emulation with CHAPI

The KW11-L line clock uses a signal generated from the AC Power Supply input line voltage and converts it in a square wave identical in frequency. The KW11-L sets one bit on its CSR for each cycle. A program can make use of this event and count unit time intervals of 20ms or 16 2/3ms depending on the input frequency, 50HZ or 60HZ correspondingly.

This device is simple to implement with CHAPI. The KW11-L has only one register and one interrupt vector (see figure 3.) In its register it only uses CSR's bits 6 and 7. For each cycle, bit 7 is set. A program can "poll" the CSR to count the unit time intervals. If the program is polling the CSR, it is responsible for resetting bit 7 to 0. If bit 6 is on, an interrupt is delivered for each cycle. The routine hooked at interrupt vector 100 increments a time interval counter.

3.1 The KW11-L CSR description

```
                  Bit Fields
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
                         |  |
                         |  | Interrupt Enable R/W)
                         |___Interrupt Monitor(R/W)
CSR: 777546
VEC: 100
```

The KW11-L CHAPI counterpart is produced from KW11.C coded as a WIN32 DLL using Microsoft Visual C++ 6.0 (see further on). In order to produce the DLL, KW11.C and CHAPI.H should be located preferably at the following path:

`C:\Program Files\Charon-11\kw11`

The Microsoft C++ command line compiler is used with the following syntax to compile KW11. The /LD option indicates the compile to produce a dll as output.

```
cl /LD kw11.c
```

The output from the previous command is KW11.DLL. It is loaded from Charon-11 configuration file inserting the instructions shown as follows:

```
# - This symbol means that the text is comment
set CPU model="PDP 11/44"
set MMU memory_size=1024.
# The following line is commented to load the CHAPI KW11 timer
# instead of the CHARON-11 internal KW11-L
#load KW11-L TIMER
load chapi TIMER                        # TIMER is a CHAPI implementation
set TIMER dll=”..\kw11\kw11.dll”        # Indicates which dll implements it
# Sets CSR,VEC addr and register_window
set TIMER register=17777546 vector=100 register_window=2.
# Pass 50HZ as line frequency to the DLL
set TIMER parameters=”FREQUENCY:50HZ”
# Console/SLU
load DL11 SLU1
set SLU1 register=17777560 vector0=060
set SLU1 line="run terminal"
set SLU1 speed=9600.
...
```

3.2 KW11.C code listing

(See 3.3 for the references in the code listing)

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include "chapi.h"
```

```
#define TIMER_PRIORITY                6
#define KW11$M_IE                     0x0040
#define KW11$M_IM                     0x0080
```

```
typedef struct {
    int     interval;
    int     enabled;
    int     csr;
    chapi_in* chapi;
} timer_t;        ──────────────────────►        1
```

```
static void CHAPI timer_ast(void* arg) {
    timer_t* tmr = arg;
    tmr->chapi->put_irq(tmr->chapi->vector, TIMER_PRIORITY, 0, 0, 0);
}
```

```
static DWORD WINAPI timer_loop(LPVOID lpvThreadParm)     ──►    4
{
    timer_t* tmr = lpvThreadParm;
    while (tmr->enabled) {
            Sleep(tmr->interval);
            tmr->csr |= KW11$M_IM;
            if (tmr->csr & KW11$M_IE) {
                tmr->chapi->put_ast(timer_ast, tmr);
            }
    }
    return 1;
}
```

```
static void CHAPI kw11_start(void* instance)   ────────►      3
{
    timer_t* tmr = (timer_t*)instance;
   HANDLE th;
    DWORD thread_id;
   tmr->enabled = 1;
   th=CreateThread(NULL, 0, timer_loop, tmr, 0, &thread_id);
   SetThreadPriority(th,THREAD_PRIORITY_TIME_CRITICAL);
}

static void CHAPI kw11_reset(void* instance)
{
   timer_t* tmr = (timer_t*)instance;
   tmr->csr = 0;
}
static void CHAPI kw11_stop(void* instance)
{
   timer_t* tmr = (timer_t*)instance;
   tmr->enabled = 0;
}

static void CHAPI kw11_write(void* instance, size_t addr, int value,
                      int  is_byte_access)   →          5
{
   timer_t* tmr = (timer_t*)instance;
   int new_csr =is_byte_access ? ((tmr->csr & ~(0xFF << ((addr & 1) << 3)))
            | ((value & 0xFF) << ((addr & 1) << 3))) : value;

   if (!(new_csr & KW11$M_IE) && (tmr->csr & KW11$M_IE)) {
         tmr->chapi->clear_irq(tmr->chapi->vector);
   }
   tmr->csr = new_csr;
}

static int CHAPI kw11_read(void* instance, size_t addr, int is_byte_access)
{
   timer_t* tmr = (timer_t*)instance;   →           6
   return is_byte_access ? (char)(tmr->csr >> ((addr & 1) << 3)) : tmr->csr;
}
```

```
static int CHAPI kw11_set_configuration(void* instance, char* parameters)
{
   timer_t* tmr = (timer_t*)instance;
   int line_frequency;

   if (strlen(parameters) == 0) return 1;
   if (sscanf(parameters, "FREQUENCY:%dHZ", &line_frequency) == 1) {
         if ((line_frequency == 50) || (line_frequency == 60)) {
            tmr->interval = 1000/line_frequency;
            return 1;
         }
   }
   printf("KW11-Error Invalid Parameter specified for line clock frequency\n");
   return 0;
}
```

```
__declspec( dllexport )
void* DLLENTRY KW11_INIT(chapi_in* ci, chapi_out* co, char* name)
{              ──────────────────────────────────────▶     2
   timer_t* tmr = (timer_t*)malloc(sizeof(timer_t));
   tmr->chapi = ci;
   co->stop  = kw11_stop;
   co->start = kw11_start;
   co->reset = kw11_reset;
   co->read  = kw11_read;
   co->write = kw11_write;
   co->set_configuration = kw11_set_configuration;
   return tmr;
}
```

3.3 KW11.C code analysis

1. This structure is device specific and should contain every component needed by the emulated device as CSRs, global variables, circular buffers, etc. It also contains a chapi_in structure pointer that will be initialized with the value provided by Charon-11 upon DLL loading and initialization process. Further Charon-11’s calls to functions pointed by chapi_out structure will pass the instantiated device specific structure as their first parameter.

2. This is the DLL initialization routine. It´s name must be in uppercase. Charon-11 will convert the dll name specified in

   ```
   Set dll=”xxx.dll”
   ```

   to uppercase and add the string “_INIT”. In our case the dll name is kw11.dll, so Charon-11 will call KW11_INIT to initialize the dll. The compiler metadata __declspec(dllexport) indicates that the function KW11_INIT will be exported by this dll.

   Charon-11 calls this function with three parameters. The first and second are the chapi_in and chapi_out structure while the third is the name of the device associated with this instance by the configuration command

   ```
   “load chapi TIMER”.
   ```

   This initialization function allocates and instance of the device specific structure timer_t, saves the received chapi_in pointer and fills the chapi_out structure, returning the instantiated timer_t structure.

   The functions provided by chapi_out are called in the following order by Charon-11:

kw11_set_configuration	Called for each ‘set’ statement contained in the Charon-11 configuration file. The only call that is of interest is when parameters length > 0, which indicates it is called by the ‘set TIMER parameters’ command.
kw11_start	Called once when Charon-11 emulation starts.
kw11_reset	Called once or twice responding to emulated bus reset signals.
kw11_read	Called any time between kw11_reset and kw11_stop to read CSRs values.
kw11_write	Called any time between kw11_reset and kw11_stop to write CSRs values.
kw11_stop	Called once when the emulation stops.

3. The kw11_start routine is responsible for Threads creation and initialization procedures. Generally it sets a global variable (tmr->enabled) that commands threads execution.

   The routine kw11_stop sets this variable to 0 making executing thread to end.

4. The kw11_loop thread executes while tmr->enabled is 1. It sleeps for tmr->interval ms emulating the AC input voltage frequency. After tmr->interval elapses it will set the CSR Monitor interrupt bit to “1” and if interrupts are enabled for this device, it will request an interrupt.

   Posting an IRQ is considered an asynchronous event, so it must be done through the chapi_in put_ast routine passing as argument and ast_handler. This ast_handler will actually post the IRQ using chapi_in function put_irq when dequeued from the AST queue by Charon-11 main thread.

5. 6. These functions do the real work supporting the read and write operation to the device CSRs. The addr argument indicates which CSR is being accessed and the is_byte_access boolean constant determines if a WORD or BYTE reference is done.

Appendix A CHAPI function reference

A.1 DLL Initialization Routine

void *DLLENTRY *XXX_INIT (chapi_in* ci,chapi_out* co,char *name)

Description:

DLL initialization routine. XXX must be replaced with the module name in Uppercase.

This routine is called by CHARON-11 for each instance creation of the user written device emulation.

Parameters:

ci: Pointer to chapi_in structure.

co: Pointer to chapi_out structure. (Filled in by this function)

name: Name of the instance speficied in the LOAD configuration command.

Returns:

Pointer to the device specific structure instantiated by this function.

A.2 CHAPI_IN functions

void put_ast (ast_handler func,void *param)

Description:

put_ast is a synchronization primitive used to process asynchronous events.

Posting an IRQ is considered by Charon-11 main thread as an asynchronous event. This function inserts the request in an internal AST queue.

Parameters:

func: function to be called when this AST request is de-queued. The function uses the following prototype:

```
(CHAPI* ast_handler)(void *)
```

param: Parameter to be passed to the function.

void put_sst (long delay,int tag,sst_fun func,void *arg1, int arg2)

Description:

put_sst is used to introduce a programmed delay at code execution measured in pdp-11 instructions.

Parameters:

delay: Long integer with number of instructions to delay execution.

tag: MUST BE ZERO.

func: Function to be called when 'delay' instructions are executed.

The function uses the following prototype:

```
(CHAPI* sst_fun)(void *arg1, int arg2)
```

arg1,arg2: Parameters passed to ‘func’

void put_irq (int vec,int pri, int delay,irq_fun fun, int arg)

Description:

Posts and interrupt request at Bus Request defined by ‘pri’ argument.

Parameters:

vec:	Vector which will serve this interrupt request.
pri:	Bus Request priority.
delay:	Number of PDP-11 instructions to delay before requesting the interrupt.
func:	Function to call when Bus Request is granted. The function uses the following prototype: `(CHAPI* irq_fun)(int arg);`
arg:	Argument for irq_fun.

void clear_irq (int vec)

Description:

Cancels an interrupt request.

Parameters:

vec:	Vector associated with Interrupt Request to be cancelled.

int read_mem (size_t adr, size_t len,char *buf)

Description:

Reads a PDP-11 memory range specified by adr and len into buf. This function is used by DMA capable devices.

Parameters:

adr:	Starting address for the transfer.
len:	Number of bytes to transfer.
buf:	Pointer to buffer that will receive the transfer.

Returns:

Number of bytes actually transferred.

int write_mem (size_t adr, size_t len,char *buf)

Description:

Writes a PDP-11 memory range specified by adr and len from buf. This function is used by DMA capable devices.

Parameters:

adr:	Starting address for the transfer.
len:	Number of bytes to transfer.
buf:	Pointer to buffer that contains data to be written.

Returns:

Number of bytes actually transferred.

A.3 CHAPI_OUT functions

int set_configuration (void *instance, char *parameters)

Description:

This function is called by Charon-11 for each ‘set’ command encountered in the configuration file related to this device instance.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.
parameters:	Buffer containing string from the **`set xxx parameters=”..”`** configuration command. If set_configuration is called for other ‘set’ commands like **`“set xxx register=nnnn”`** then the parameters will contain a NULL string.

Returns:

TRUE:	If succesful. Receiving a zero length parameters value should return this value.
FALSE:	If error encountered. Returning this value will cause Charon-11 to eliminate this device from the configuration.

void start (void *instance)

Description:

This function is called by Charon-11 when starting simulation.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.

void reset (void *instance)

Description:

This function is called by Charon-11 when it simulates Bus Reset signals.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.

void stop (void *instance)

Description:

This functions is called by Charon-11 when it stops simulation.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.

void write(void *instance,size_t addr,int value, int is_byte_access)

Description:

This function is called by Charon-11 when writing to device registers.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.
addr:	22-bit address of register.
value:	New register value.
is_byte_access:	FALSE indicates WORD access. TRUE indicates BYTE access.

int read(void *instance,size_t addr,int value, int is_byte_access)

Description:

This function is called by Charon-11 when reading device registers.

Parameters:

instance:	Pointer to device internal structure instance as returned by the XXX_INIT function.
addr:	22-bit address of register.
is_byte_access:	FALSE indicates WORD access. TRUE indicates BYTE access.

Returns:

Integer containing the sign extended value of the resulting read operation.

Reader's comments

We appreciate your comments, suggestions, criticism and updates of this CHARON-11 manual by email. Our email address is: **CHARON@SRI-GVA.CH**

Please mention that it concerns the CHARON-11 Application interface manual, Version: 1 – February 2002, document number 10-01-040, and include your name, company (and phone number if you want to receive feedback that way).

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

CHARON-11 Customer Service

If at any time you have questions or suggestions about the CHARON-11 product, please:

- Send an Email to **info@charon-11.com**, or
- Call CHARON support at **+41 22 794 1071**, or
- Send a FAX to +41 22 794 1073