EDOC131

EUROGAM PROJECT

"tcard" Register Server

General Software Architecture

October, 10th 1991

Edition : 1.3
Author : P.KADIONIK

CENBG Bordeaux
CNRS-IN2P3 France

Register Server software

P. KADIONIK

10 octobre 1991

This document intends to present briefly the general architecture used for the "tcard" Register Server. Another more consistent document grouping all the informations on the global "tcard" software will be produced later...

1 General "tcard" software presentation :

The "tcard" Register Server is a software running in the VxWorks environment. It is the lower level of the global "tcard" software.

Three levels must be distinguished :

- the GUI software level (G. ZEHNACKER).
- the intermediate software level (C. DIARRA).
- the Register Server software level (P. KADIONIK).

The GUI software level proposes to the physician a usefull Graphical User Interface in order to affect easily all parameters of all VXI cards involved in an experiment. It is running in a UNIX environment.
The intermediate software is the link between the GUI interface software and the Register Server software. It maintains an important database. (See the C. DIARRA's document EDOC126) . It proposes a non graphical interface to the physician too. It is running in a UNIX environment.
The Register Server software permits to affect all the registers of all VXI cards in one VXI crate. It uses the RPC/XDR data representation and the TCP/IP transport level. It is running in the VxWorks environment in each VXI Crate Controller card.

2 Register Server general philosophy :

The most important thing was to find out a flexible software data representation of the VXI cards registers.

It was decided to use the following representation :

```
-------------------------
|     register number # |
-------------------------
|        channel        |
-------------------------
|        offset         |
-------------------------
| position of the fisrt |
|    bit to mask        |
-------------------------
|   width of the mask   |
-------------------------
|        mnemonic       |
-------------------------
|    D16/D32 register   |
-------------------------
```

where :

- register number : number of a VXI card register.
- channel number : a VXI card may have several channels.
- offset : offset of the register from the VXI base address of the card.
- position of the fisrt bit to mask.
- width of the mask to apply.
- mnemonic of the register.
- type of the register : D16 or D32.

To affect a register or a part of a register, this typical scheme is applied :
1- Choice of the operation : read, write, scope, switch, "mesures de tension" operations. There is one RPC server for each operation. The choice is performed by the UNIX intermediate client software.
2- All the informations coming from the intermediate level are collected by the Register Server using the RPC mechanism (see below for more details).
3- Determination of the type of the concerned card : 'G' for the Ge cards, 'S' for the BGO cards, 'R' for the ROCO card, 'T' for the Trigger card.
4- Verification if the card exits in relation with the VXI autoconfiguration database software (the Trigger card doesn't exit in all VXI crates for exemple).
5- Collection of informations in the database (first bit to mask, mask width, register type (D16, D32),...).
6- Search of the addressing mode (A16, A24, A32) in relation with the VXI autoconfiguration database.
7- Perform a mask operation (read, write, switch, scope,...).
8- Return of a status.

During each step, if something is going wrong, a status is sent to the intermediate software level according to an "errno.h" file.

An exemple of the data representation, the **ge.h** file, used by the Register Server is given below for the Ge card (in C language).

It was decided to use the channel number 0 for all common parameters of a VXI card.

```
/*********************************************************************/
/*                                                                   */
/*                      % EXPLANATIONS %                             */
/*                                                                   */
/*                                                                   */
/* All Ge card registers will be referenced by an offset beginning   */
/* at their own VME_GeCard_BaseAdd one can read in the own offset    */
/* register of its Register Based Device.                            */
/*                                                                   */
/* Each Ge card has a set of global registers and a set of registers */
/* for each Ge Channel (6).                                          */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*                     % GE CARD MAPPING %                           */
/*                                                                   */
/* offset $000 : start of the global registers space                 */
/* offset $100 : start of the Ge channel 1 registers space           */
/* offset $200 : start of the Ge channel 2 registers space           */
/* offset $300 : start of the Ge channel 3 registers space           */
/* offset $400 : start of the Ge channel 4 registers space           */
/* offset $500 : start of the Ge channel 5 registers space           */
/* offset $600 : start of the Ge channel 6 registers space           */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
#define TOTAL_GE_REGISTERS 238

struct modelization
Ge_card[TOTAL_GE_REGISTERS] = {
/*********************************************************************/
/*      reg_nbr     ch      offset     pos  width     mnemo  D16/D32*/
/*********************************************************************/

/*********************************************************************/
/* CHANNEL 0 : COMMON REGISTERS                                      */
/*********************************************************************/
{     1,      0,      0x0000,      0,      8,     "CFDWith",   16},
{     2,      0,      0x0002,      0,      8,     "FTSample",  16},
```

```
{     3,      0,      0x0004,     0,      8,     "VALSample", 16},
{     4,      0,      0x0006,     0,      8,     "StRdOut",   16},
{     5,      0,      0x0010,     0,     16,     "TstGen",    16},
{     6,      0,      0x0030,     8,      3,     "AMuxCha2",  16},
{     7,      0,      0x0030,     0,      3,     "AMuxCha1",  16},
{     8,      0,      0x0030,    11,      5,     "AMuxPar2",  16},
{     9,      0,      0x0030,     3,      5,     "AMuxPar1",  16},
{    10,      0,      0x0032,     8,      3,     "DMuxCha2",  16},
{    11,      0,      0x0032,     0,      3,     "DMuxCha1",  16},
{    12,      0,      0x0032,    11,      5,     "DMuxPar2",  16},
{    13,      0,      0x0032,     3,      5,     "DMuxPar1",  16},
{    14,      0,      0x0034,     0,      3,     "VInMux",    16},
{    15,      0,      0x0034,     3,      5,     "VinInput",  16},
{    16,      0,      0x0036,     0,      1,     "CtlFEna",   16},
{    17,      0,      0x0038,     0,      1,     "EndModule", 16},
{    18,      0,      0x0038,     1,      1,     "PileUpRej", 16},
{    19,      0,      0x0038,    15,      1,     "emptyFifo", 16},
{    20,      0,      0x003a,     0,      1,     "TstGenAtt", 16},
{    21,      0,      0x003c,    16,     16,     "TstWFifo",  32},
{    22,      0,      0x003c,    16,     16,     "TstRFifo",  32},

*********************************************************************/
/* CHANNEL 1 : CHANNEL REGISTERS                                    */
/********************************************************************/
{    23,      1,      0x0100,     0,      8,     "CFDThresh", 16},
{    24,      1,      0x0102,     0,      8,     "CFDDelay",  16},
{    25,      1,      0x0104,     0,      8,     "PZAdj",     16},
{    26,      1,      0x0106,     0,      8,     "PDSGate",   16},
{    27,      1,      0x0140,    16,     16,     "RO20MTh",   32},
{    28,      1,      0x0144,    16,     16,     "RO4MTh",    32},
{    29,      1,      0x0148,    16,     16,     "ROTiTh",    32},
{    30,      1,      0x014c,    16,     16,     "ROCoTh",    32},
{    31,      1,      0x0150,    16,      8,     "ROAdd20MGr",32},
{    32,      1,      0x0150,    24,      6,     "ROAdd20MIt",32},
{    33,      1,      0x0154,    16,      8,     "ROAdd4MGr", 32},
{    34,      1,      0x0154,    24,      6,     "ROAdd4MIt", 32},
{    35,      1,      0x0158,    16,      8,     "ROAddTiGr", 32},
{    36,      1,      0x0158,    24,      6,     "ROAddTiIt", 32},
{    37,      1,      0x015c,    16,      8,     "ROAddCoGr", 32},
{    38,      1,      0x015c,    24,      6,     "ROAddCoIt", 32},
{    39,      1,      0x0160,    16,     16,     "ROTstR20M", 32},
{    40,      1,      0x0164,    16,     16,     "ROTstR4M",  32},
{    41,      1,      0x0168,    16,     16,     "ROTstRTi",  32},
{    42,      1,      0x016c,    16,     16,     "ROTstRCo",  32},
{    43,      1,      0x0170,    16,      1,     "ROS1S20MEna", 32},
{    44,      1,      0x0170,    17,      1,     "ROS1S4MEna",  32},
{    45,      1,      0x0170,    18,      1,     "ROS1STiEna",  32},
```

```
{    46,      1,      0x0170,     19,      1,      "ROS1SCoEna",  32},
{    47,      1,      0x0174,     16,      1,      "RO20MEna",   32},
{    48,      1,      0x0174,     17,      1,      "RO4MEna",    32},
{    49,      1,      0x0174,     18,      1,      "ROTiEna",    32},
{    50,      1,      0x0174,     19,      1,      "ROCoEna",    32},
{    51,      1,      0x0178,     16,     16,      "ROS1SReg",   32},
{    52,      1,      0x017c,     16,     16,      "ROTstReg",   32},
{    53,      1,      0x0120,      0,      1,      "CCRAmpExt", 16},
{    54,      1,      0x0120,      1,      1,      "CCRVetLem", 16},
{    55,      1,      0x0120,      2,      1,      "CCRTFAGai", 16},
{    56,      1,      0x0120,      3,      1,      "CCRBDCEna", 16},
{    57,      1,      0x0120,      4,      1,      "CCRTstEna", 16},
{    58,      1,      0x0120,      5,      1,      "CCRChDis",  16},

...
...
...
{    46,      6,      0x0670,     19,      1,      "ROS1SCoEna",  32},
{    47,      6,      0x0674,     16,      1,      "RO20MEna",   32},
{    48,      6,      0x0674,     17,      1,      "RO4MEna",    32},
{    49,      6,      0x0674,     18,      1,      "ROTiEna",    32},
{    50,      6,      0x0674,     19,      1,      "ROCoEna",    32},
{    51,      6,      0x0678,     16,     16,      "ROS1SReg",   32},
{    52,      6,      0x067c,     16,     16,      "ROTstReg",   32},
{    53,      6,      0x0620,      0,      1,      "CCRAmpExt", 16},
{    54,      6,      0x0620,      1,      1,      "CCRVetLem", 16},
{    55,      6,      0x0620,      2,      1,      "CCRTFAGai", 16},
{    56,      6,      0x0620,      3,      1,      "CCRBDCEna", 16},
{    57,      6,      0x0620,      4,      1,      "CCRTstEna", 16},
{    58,      6,      0x0620,      5,      1,      "CCRChDis",  16},
};
```

A **bgo.h** and **roco.h** files were created in the same way (the **trigger.h** file will be produced later).

3 Data representation structure for RPC calls :

A **data_in.h** file was created for the in-RPC call given below in C language.

```
struct data_in {
/* Name of the vxi crate :                                              */
/* Official names :  -vxi1                                              */
/*                   -vxi2                                              */
/*                   -vxi3                                              */
/*                   -vxi4                                              */
char crate_name[20];
```

```
/* Slot number in the VXI crate :                                    */
/*                    - 1 to 12                                      */
unsigned short int slot_number;

/* Register number of a specified card type. See typical data        */
/* structure of the card for more informations.                      */
unsigned short int reg_number;

/* Channel number of the specified card. A card may have several     */
/* channels (6 channels (1 to 6) plus the common channel numbered    */
/* "0" for the Ge card) or no channel (channel_number = 0 in this    */
/* case).                                                            */
unsigned short int channel_number;

/* Data to write. Just used in a RPC-write operation. Set to 0x0000  */
/* for the RPC-read operations.                                      */
unsigned long int value1;

/* Second value for specific applications like the scope RPC         */
unsigned long int value2;

/* Third value for specific applications like the mt RPC             */
/* A is for analogic, and L is for logic.                            */
char value3[20];

/* Card type :        - 'G' for Ge cards                             */
/*                    - 'S' for BGO cards                            */
/*                    - 'T' for Trigger cards                        */
/*                    - 'R' for Roco cards                           */
char card_type[20];
};
```

In the same way, a **data_out.h** file was created for the out-RPC call :

```
struct data_out {
/* Status according to the errno.h file database of the current      */
/* RPC-read, RPC-write, RPC-scope, RPC-button operation.             */
short int status;

/* Result of the RPC call.                                           */
unsigned long int result;
};
```

A **errno.h** was created to send a status for all kinds of errors.

By using the SUN "rpcgen" tool, XDR files were generated for read operations (**reg_read.h**), write operations (**reg_write.h**)...

4 User libraries :

Two main libraries were created :

- The **rpclib.c** library
- The **vxilib.c** library

The rpclib.c library performs general tasks :
- search of informations on a register for a card.
- search of the addressing type.
- search of the VXI base address of a card.
- bits mask operations.
- ...

The vxilib.c performs the read/write operations :

```
- VXIReadA16D16 operation          - VXIWriteA16D16 operation
- VXIReadA24D16 operation          - VXIWriteA24D16 operation
- VXIReadA32D16 operation          - VXIWriteA32D16 operation
- VXIReadA24D32 operation          - VXIWriteA24D32 operation
- VXIReadA32D32 operation          - VXIWriteA32D32 operation
```

5 RPC call synthax :

The commands used by the UNIX intermediate software to perform an RPC call to the Register Server depends on the type of the operation :

A structure of type data_in (request) is initialized and the result is returned in a data_out (result) structure :

```
result = type_of_RPC_Call(request);
```

The request structure contains :
- The crate name.
- The slot number.
- The register number.
- The channel number.
- Value(s).
- The card type.

The result structure contains :
- A status.

- A value.

The informations used for the in-RPC call are issued from the database of the intermediate software. This database is updated after each VXI autoconfiguration phase (after the power-up of the VXI crates) because a VXI crate may contain more or less VXI cards for a specific experiment. This loopback software running in the VXI Crate Controller in relation with the autoconfiguration software and the Register Server software is not completely finished yet.

6 Specific RPC procedures :

For each RPC operation (read, write,...), a typical RPC procedure was written in the Vx-Works environment and integrated in the Register Server software running in the VXI Crate Controller.

7 Conclusion :

The read, write, scope, switch functionalities have been implemented for the VXI Ge, BGO, ROCO cards.
The global "tcard" software has given entire satisfaction to the hardware people.

For the Register Server software :
The read, write, scope, switch functionalities have been implemented for the VXI Ge, BGO, ROCO cards.
The trigger card will be integrated. The loopback software to the intermediate level software for the database updating and the "mt" function will be implemented.