Telescopio Nazionale Galileo

Specification to Use the DLL to communicate with the CCD controller

24/4/2006 - Version 1.0
21/6/2006 - Version 1.1 (DLL ver. 3.61)
30/6/2006 – Version 1.2 (DLL ver. 3.91)
17/8/2006 – Version 1.3 (DLL ver. 3.91) – scan sequences with binning ADDED

Written by:
Rosario Cosentino

Specification to Use the DLL to communicate with the CCD controller

To use the DLL with the hardware of the TNG the files CTRL_PROCEDURES.H and CTRL_PROCEDURES.lib had to be included in the project.
To compile correctly the visual C code with the TNG hardware the following variable must be defined:

#define PCI_A100_1_00
#define SPC_A100
#define CDS_A100

All the commands return an integer that contain the error code. The list of error codes is showed in appendix A.

Booting of the system

The first step to control the CCD electronic is the boot of the PCI, the SPC and the CDS boards. The booting sequence is the following.

int = Boot_Kernel_PCI_A100(char *file_boot)
Where file_boot is the name of the bynary file for the DSP (.HEX files).
Int = Init_SPC_A100_by_PCI_A100(void);
Boot the SPC board
Int = Init_CDS_A100_by_PCI_A100(void);
Boot the CDS board

Commands to read the low level software configuration

There are a series of command to interrogate the system about the low level configuration software.
int Get_DLL_Release(char *release, int max_chars)
read the DLL release installed in the operative system
int Get_DLL_Model(char *release, int max_chars)
read the DLL model installed in the operative system
int Get_Driver_Release_CTR_S000(char *release, int max_chars, unsigned int &Ret_Code)
read the driver release installed in the operative system
int Get_Firmware_Release_PCI_A100(char *release, int max_chars)
read the firmware release installed in PCI board
int Get_Firmware_Model_PCI_A100(char *release, int max_chars)
the firmware model installed in PCI board

Command to program the environment of the system

These commands must be used with caution because can damage the detector. For this reason normally the programming of most of them are done through tables. Each command is described below and next to the command description the note ***caution*** is added for some of them.

Bias Programming (caution if the output is enabled):

Int = Bias_CDS_A100_by_PCI_A100(unsigned int Board_Addr, unsigned int Bias_Number, double Bias_Value)
Program the biases values of the CDS A100 board. This board furnishes 16 bias voltages with the ranges showed in table 1.
The parameter of this function is:
Board_addr = the board number, 0 if only one CDS is mounted.
Bias_Number = the bias number, from 1 to 16
Bias_Value = the value of the bias (the ranges permitted are showed in table 1)

Bias Number	Min Value	Max value
Bias 0 - 3	+15	+30
Bias 4 – 7	+5	+15
Bias 8 - 11	-5	+5
Bias 12 - 15	-10	+10

Table 1- Bias Ranges for the CDS A100 board

Int = CCD_Voltage_Control_CDS_A100_by_PCI_A100(unsigned int Enable) (caution)

Enable the biases voltages outputs. This command must be used only when the voltages value are compatible with the CCD bias. The value of Enable is 1 to enable, 0 to disable the biases.

Clock programming (caution)

Int = Clock_SPC_A100_by_PCI_A100(unsigned int Clock_Type,unsigned int Clock_Polarity, unsigned int Clock_Number, double Clock_Value, double max_spc_a100_clk_val=MAX_SPC_A100_CLK_VALUE, double min_spc_a100_clk_val=MIN_SPC_A100_CLK_VALUE)

Program the clocks voltages. There are two kinds of clocks: The horizontal and the vertical. Each kind had two different programmable levels: the High and the Low level.
The parameters of this function are:

Clock_Type = CLOCK_TYPE_HOR or CLOCK_TYPE_VERT
Clock_Polarity = CLOCK_POL_POS or CLOCK_POL_NEG
Clock_Number = the number of clock, from 1 to 16
Clock_Value = the value of the clock, from 0 to +10 volts for the CLOCK_POL_POS, from 0 to -10 volts for the CLOCK_POL_NEG
The other parameter is fixed by the environment

Command to program the Gain values:

Int = Set_Gain_CDS_A100_by_PCI_A100(unsigned int Board_Addr,unsigned int Gain_Number);

Set the gain of the first amplification stage of the signal. The values are 0, 1 or 2

Command to program the filter values:

The command:
Int = Set_Filter_CDS_A100_by_PCI_A100(unsigned int Board_Addr,unsigned int Filter_Number)
Is substituted by the command:
Int = Set_Band_CDS_A100_by_PCI_A100(unsigned int Board_Addr,unsigned int Filter_Number)

Set the filter applied at the signal. The values are 0, 1 or 2

Command to program the input offset:

Int = Offset_Input_CDS_A100_by_PCI_A100(unsigned int Board_Addr, unsigned int Offset_Number, double Offset_Value);

The parameter of this function are:
Board_Addr = the board number, 0 if only one CDS is mounted.
Offset_Number = the value included in the range 1-4 are permitted, each one set the channel.
Offset_Value = the value, in voltage, of the offset. The value from -5 to +5 is permitted.

Command to program the output offset:

Int = Offset_Output_CDS_A100_by_PCI_A100(unsigned int Board_Addr, unsigned int Offset_Number, double Offset_Value);

The parameter of this function is:
Board_Addr = the board number, 0 if only one CDS is mounted.
Offset_Number = the value included in the range 1-4 are permitted, each one set the channel.
Offset_Value = the value, in voltage, of the offset. The values from -5 to +5 are permitted.

Command to set the CCD temperature:

Int = Heater_Control_SPC_A100_by_PCI_A100(bool On_Off);

Enable/disable the temperature control.

Command to program CCD temperature:

The command:
Int Thrs_SPC_A100_by_PCI_A100(double Set_Temp)
Is substituted by the command:
Thrs_AD590_SPC_A100_by_PCI_A100(double Set_Temp m_fTempCCD)

Set the CCD temperature. The value is in Celsius degree.

Command to read telemetry:

There are two commands to read the telemetry of the CCD controller. One to read the CDS board (biases) and another one to read the SPC board (clocks and temperature). The furnished values are stored in a structure.

CDS Telemetry:

Telemetry data structure (definition)

```
typedef struct _TELEM_CDS_A100_DATA {
        unsigned int                    n_Valid_Board;
        unsigned int                    n_Data_Acq;
        unsigned int                    n_Channel;
        unsigned int                    n_Sample;
        unsigned int                    n_Mux_Channel;
        TELEM_CDS_A100_BOARD            Board[MAX_ACQ_BOARD];
} TELEM_CDS_A100_DATA, *pTELEM_CDS_A100_DATA;
```

Bias data structure (definition)

```
typedef struct _TELEM_CDS_A100_BOARD {
        double   Bias[MAX_BIAS_CDS_A100];
} TELEM_CDS_A100_BOARD, *pTELEM_CDS_A100_BOARD;
```

int GetTlm_CDS_A100_by_PCI_A100(pTELEM_CDS_A100_DATA pTelem_Data);

Each elements of bias telemetry can be retrieved from the structure.
i.e. the first bias of the first CDS A100 board is:
Telem_Data.Board[0].Bias[0];

SPC Telemetry:

Telemetry data structure (definition)

```
typedef struct _TELEM_SPC_A100_DATA {
        unsigned int            n_Sample;
        unsigned int            n_Data_Acq;
        double                  Clock_Hor_High[MAX_CLK_SPC_A100];
        double                  Clock_Hor_Low[MAX_CLK_SPC_A100];
        double                  Clock_Ver_High[MAX_CLK_SPC_A100];
        double                  Clock_Ver_Low[MAX_CLK_SPC_A100];
        double                  Temp[MAX_TEMP_SPC_A100];
        double                  V15P_Analog;
        double                  V15N_Analog;
} TELEM_SPC_A100_DATA, *pTELEM_SPC_A100_DATA;
```

int GetTlm_SPC_A100_by_PCI_A100(pTELEM_SPC_A100_DATA pTelem_Data);

Each elements of bias telemetry can be retrieved from the structure.
Some examples are showed below:

```
Telem_Data.Clock_Hor_High[0] ----  first horizontal clock high
Telem_Data.Clock_Ver_High[0] ----  first vertical clock high
Telem_Data.Clock_Hor_Low[0] ----  first horizontal clock low
Telem_Data.Clock_Ver_Low[0] ----  first vertical clock low
Telem_Data.Temp[0]          ----  first Temperature telemetry (there are 4 temperature telemetry)
```

Acquisition (Pci_A100.h, PCI_A100_CDS_A100_PDS_A000.h)

To acquire the image from the CCD the system needs the tables of sequences (waveform to do the CCD scan), and a special variable that contain the microcode of the CCD scansion. This variable contains the information about the CCD scan, i.e. how many times each sequence had to be used.

In order to make an acquisition four actions must be made:

First action: Load the waveform

Int = PCI_A100_Load_Tables (char* Path_Tables, char* List_File_Name_Tables)

This command read a file that contain e list of sequences files to be loaded.
Path_tables = the path where the sequences files are saved
List_File_Name_Tables = the list of sequences files to be loaded. The maximum numbers of sequences files are 16. Depending by the order in the table, the system assigns a number (from 1 to 16) at the table.

Sequence file	Number assigned
Vert.wfN	0
FastHoriz.wfN	1
Horiz.wfN	2
Horiz_Light.wfN	3
Horiz_custom.wfN	4
.....	.....
Horiz_custom1.wfN	F

Second action: Generate and load the scan sequence for the CCD readout

Int = PCI_A100_Load_Scan_Sequence(unsigned int* Scan_Memory)

This command use the variable **Scan_Memory** to generate the sequences, by using the sequences files loads previously. The variable **Scan_Memory** contains the microcode with the command to generate the sequences. The sequence contained in the **Scan_Memory** variable depends by the CCD area, the box (or multibox) and the type of acquisition (full frame, frame transfer).

The microcode is the following:

END_TAB	0x000000	(Command)
TAB	0x1|0|0000	(Command | Num_Tab | Repeat_Tab)
LOOP	0x2|00000	(Command | Number of Loop)
REPEAT	0x300000	(Command)
VERTICAL	0x400000	(Command)
HORIZONTAL	0x500000	(Command)

A typical scan that read a CCD 1000 X 2000, in full frame mode had the following structure:

Begin a cycle of 2000
Use the vertical table
Number of table used like vertical table (depends by the load order of the table)
Use the horizontal table
Repeat 1000 times the table indicated (depends by the load order of the table)
End the cycle
End

The values in the **Scan_Memory** variable are:

Value	**Comments**
2007D0	LOOP (0x2|0007D0) (Command | Number of Loop)
400000	VERTICAL (0x400000) (Command)
100001	TAB (0x1|0|0001) (Command | Num_Tab | Repeat_Tab)
500000	HORIZONTAL (0x500000) (Command)
1103E8	TAB (0x1|1|03E8) (Command | Num_Tab | Repeat_Tab)
300000	REPEAT (0x300000) (Command)
000000	END (0x000000)

Third action: Load the wipe sequence

The wipe sequence is very similar to the acquisition sequence and depends only by the CCD area.

This sequence is used to clean the CCD before the exposition and normally use only the vertical waveform.

Int = PCI_A100_Load_Wipe_Sequence(unsigned int* Scan_Memory)

Forth action: Start the acquisition

Int CDS_A100_PDS_A000_Image_by_PCI_A100(double Sht_Time, unsigned int Enable_Sht, unsigned int readout_sync, unsigned int* number_word, void * pointer_mem, unsigned int mem_size_allocated, unsigned int = SPCA100)

Sht_Time	= the exposure time
Enable_Sht	= the flag that enable the open of the shutter
readout_sync	= a variable used to synchronize* the acquisition, the values range is from 0 to 3
number_word	= number word of the image
pointer_mem	= pointer of the image
mem_size_allocated	= memory allocated by the image
SPCA100	

*) to optimize the electronic noise this value must be chosen depending by the r.m.s. noise of a series of bias images, acquired with different value of readout_sync.

The image obtained by the function **_A100_PDS_A000_Image_by_PCI_A100** start at the memory location pointed by *pointer of the image*. In the memory are stored the value of the four channels in a sequential way.

1 Board: 4 Channels (Channels wrote : 0 - 1 - 2 - 3)
// 16 BIT Array
1° Pixel Channel 1
1° Pixel Channel 0
1° Pixel Channel 3
1° Pixel Channel 2
2° Pixel Channel 1
2° Pixel Channel 0
2° Pixel Channel 3
2° Pixel Channel 2

Other commands related with the acquisition are available:

Int = Exposition_Readout_Abort_by_PCI_A100(void)

Abort acquisition

Int = Exposition_Time_Update_by_PCI_A100(int* millisec)

Update the exposure time

APPENDIX A – Examples of generation of wipe and scan sequences

The wipe sequence is a series of vertical scan of the whole CCD. This sequence is used by the CCD controller to clean the CCD before the exposition. Areay is the column number of the CCD.

// Start of Generation of the wipe sequences

```
index=0;
if (areay != 0)
        {
        Scan_Memory[index] = 4194304; // 400000 vertical
        index++;
        Scan_Memory[index] = 1048576 + areay;  // 100000 + temp Exe vert (split)
        index++;
        }
Scan_Memory[index] = 0;
index++;
```

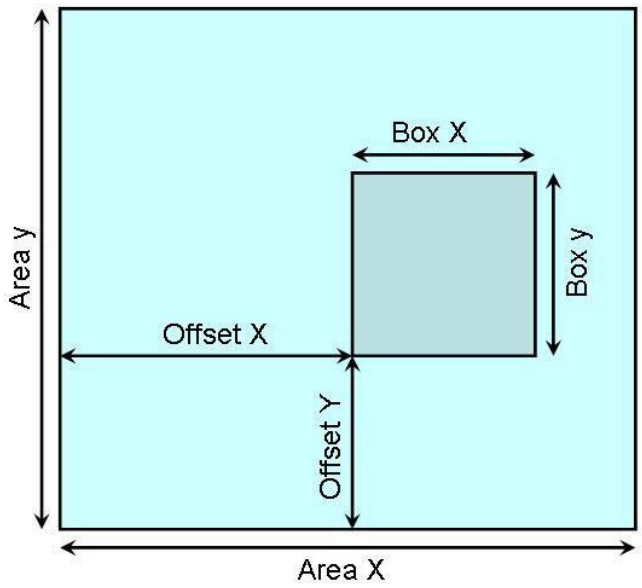
// end wipe generation

To generate a scan sequence that read a box of a CCD and skip the rest of the image the C code is the following:

areax is the image row number of the single channel image, it depends by the CCD areax, by the number of CCD, by the number of outputs and by the binning.
areax = ((CCD_areaX * number_of_CCD) / Number_of_output_used)/BinningX

areay is the CCD column number of the single channel image, it depends by the CCD area y and by the binning.
AreaY = CCD_areaY / Binning_Y

offsetx is the start row of the box
offsety is the start column of the box
boxx is the box dimension (row)
boxy is the box dimension (column)
Xslow is the number assigned at the sequences file that perform the Xslow scan
Yslow is the number assigned at the sequences file that perform the Yslow scan
Xfast is the number assigned at the sequences file that perform the Xfast scan
Wipe is the number assigned at the sequences file that perform the wipe scan



```
// Start of generation of generic scan sequence

yslow=Table_used_to_Yslow_scan;// default = 0
xfast=Table_used_to_Xfast_scan;// default = 3
wipe = Table_used_to_Wipe_scan;// default = 0
if (binx == 1)
        xslow=Table_used_to_binx1_scan; // default = 1
if (binx == 2)
        xslow=Table_used_to_binx2_scan; // default = 4
if (binx == 4)
        xslow=Table_used_to_binX4_scan; // default = 5
//
if (AcqMode1 == 0) // Full Frame
        {
        index=0;
        temporaneo = (areay - offsety - boxy) * biny;
        if (temporaneo != 0)
                {
                // start vertical skip
                Scan_Memory[index] = 4194304; // 400000 vertical
                index++;
                Scan_Memory[index] = 1048576 + temporaneo + yslow*65536;
                // 100000 + temp Exe vert (split)
                index++;
                }
                //end vertical skip
        Scan_Memory[index] = 2097152 + boxy; // Loop on the column
        index++;
        Scan_Memory[index] = 4194304; // 400000 vertical
        index++;
        Scan_Memory[index] = 1048576 + biny + yslow*65536; // do 'biny' times the vertical scan
        index++;
        Scan_Memory[index] = 5242880; // 500000 horizontal
        index++;
        if (offsetx != 0)
                {
                // start horizontal skip
                Scan_Memory[index] = 1048576 + xfast*65536 + offsetx*binx;
                // 130000 (fast hor shift)
                index++;
                }
        Scan_Memory[index] = 1048576 + xslow*65536 + boxx; // 110000 (slow hor)
        index++;
        temporaneo = (areax-offsetx-boxx)*binx;
        if (temporaneo != 0)
                {
                Scan_Memory[index] = 1048576 + xfast*65536 + temporaneo;
                // 130000 (fast hor shift)
                index++;
                }
        Scan_Memory[index] = 3145728; // 300000 (fine ciclo)
```

```
        index++;
        if (offsety != 0)
                {
                Scan_Memory[index] = 4194304; // 400000 (vert)
                index++;
                Scan_Memory[index] = 1048576 + yslow*65536 + offsety*biny;
                // 100000 (shift vert)
                index++;
                }
        Scan_Memory[index] = 0;
        index++;
}
```

APPENDIX B - Return Codes

CTR_OK	1
FILE_NOT_FOUND	2
COMUNICATION_SPC_ERROR	3
PARAMETER_INCORRECT	4
LEN_MESSAGE_LARGE	5
MEMORY_FAIL	6
DSP_BOOT_FAIL	7
DLL_FAIL	8
PCI_TX_LINK_NOT_READY	9
PCI_RX_LINK_NOT_READY	10
COM_PAR_ERR	11
PCI_BOARD_NOT_PRESENT	12
LENGHT_TABLES_ERROR	13
FILE_TABLES_ERROR	14
NO_TABLES_LOADED	15
COMUNICATION_DSP_ERROR	16
MONITOR_FULL	17
CTR_NOT_FOUND	18
USER_ABORT	19
DRIVER_FAIL	20
BUFFER_FULL	21
INCOMPLETE_DATA	22
MESSAGE_UNKNOWN	23
OPEN_PAR_PORT_FAIL	24
PAR_PORT_INVALID	25
SPC_TX_LINK_NOT_READY	26
SPC_RX_LINK_NOT_READY	27
WRITE_PCI_DSP_FIFO_TIMEOUT	28
READ_PCI_DSP_FIFO_TIMEOUT	29
READ_PCI_DSP_WRONG_DATA_NUMBER	30
READ_PCI_DSP_WRONG_CHECKSUM	31
SPC_A000_RESET_FAIL	32
COMMAND_REJECTED_DURING_READOUT	33
PENDING_PROCESS_TERMINATED_OK	34
NO_PENDING_PROCESS_FOUND	35
PROCESS_ABORTED_BY_ANOTHER	36
EXPOSITION_TIME_UPDATE_ERROR	37
SPC_A200_STATUS_READ_FAIL	38
SCAN_SEQUENCE_LENGTH_ERROR	39
TAB_OF_TAB_LENGTH_ERROR	40
EXPOSITION_STARTED	41
EXPOSITION_TERMINATE	42
EXPOSITION_ABORTED_BY_ANOTHER	43
READOUT_ABORTED_BY_ANOTHER	44
PENDING_EXPO_TERMINATED_OK	45
PENDING_READOUT_TERMINATED_OK	46
READOUT_STARTED	47
LCA_NOT_LOADED	48
EVENT_NOT_LINKED	49
NO_TAB_OF_TAB_LOADED	50
TABLES_ALREADY_LOADED	51
WRONG_DSP_RETURNED_STATUS	52
TIMEOUT_WRITING_DSP_DATA	53
TIMEOUT_READING_DSP_DATA	54
CHECKSUM_ERROR_FROM_DSP	55
DSP_RESPONSE_CHECKSUM_WRONG	56
WRONG_STRING_LENGTH	57
PROCEDURE_NOT_SUPPORTED	58