Interfaz de control y monitorización del equipo de procesamiento y captura de datos Viper.

Módulo de adquisición de datos ADAM 6017.

D. Cordobés, J.A. López Pérez,
R. Bolaño, C. Almendros

Informe Técnico IT - OAN 2009 - 1

CONTENIDO

I. Introducción

El Centro Astronómico de Yebes cuenta con un módulo de adquisición y procesamiento de la casa Arcom, modelo Viper. Este sistema está basado en el procesador RISC Intel PXA255, que le proporciona gran funcionalidad, haciéndolo válido para un amplio abanico de aplicaciones.

Viper puede gestionar distintas tarjetas de adquisición de datos, siempre y cuando sean del tipo PC104. En Yebes se disponen de dos tarjetas de ese tipo:

- *Aim104-IO32*, que tiene 32 salidas/entradas TTL.
- *Aim104-Multi-IO*, que dispone de 16 entradas analógicas codificadas con 12bits, 2 salidas analógicas de 12bits, y 8 entradas digitales.

En este informe se presenta el proceso que se ha seguido para poner en marcha y operar el sistema.

Figura 1. Módulo Viper de adquisición y procesamiento de datos

II. Instalación software

El proceso de instalación depende del sistema operativo sobre el que se desee trabajar. En nuestro caso se ha utilizado Linux.

Lo primero que hay que hacer es instalar las librerías del Viper que nos van a permitir desarrollar aplicaciones sobre él. Para ello, el fabricante nos proporciona un CD *"Development Kit"* que contiene el ejecutable Perl *"Install"* que automatiza el proceso. A continuación, hay que instalar los drivers de las tarjetas PC104 que se dispongan. Estos drivers, en su versión más reciente, se pueden descargar de la página web del fabricante.

Hecho esto, podemos entrar en el Viper sin más que abrir una sesión remota con el comando *ssh*. Una vez allí, lo primero que se debe hacer es cargar los módulos correspondientes a las tarjetas PC104, en particular se deben invocar los comandos:

/sbin/modprobe aim104-multi_io io_base=0xF7000180
/sbin/modprobe aim104-io32 io_base=0xF7000188

,ya que los jumpers de los módulos *aim104-multi_io* y *aim104-io32* se encuentran configurados en las direcciones 180h y 188h, respectivamente.

III. Desarrollo de aplicaciones

3.1) Pinout de las tarjetas PC104

A la hora de desarrollar aplicaciones, es necesario conocer el pinout de las tarjetas PC104 de que dispone el Viper.

En la **Figura 2** se muestra una vista trasera del Viper con la ubicación de las tarjetas PC104. El Viper dispone de dos tarjetas PC104, cada una de ellas con dos conectores DB25. La tarjeta *PC104 IO-32* está en la fila de arriba, mientras que la *PC104 Multi-IO* en la de abajo. Las señales presentes en los pines de las tarjetas son las siguientes:

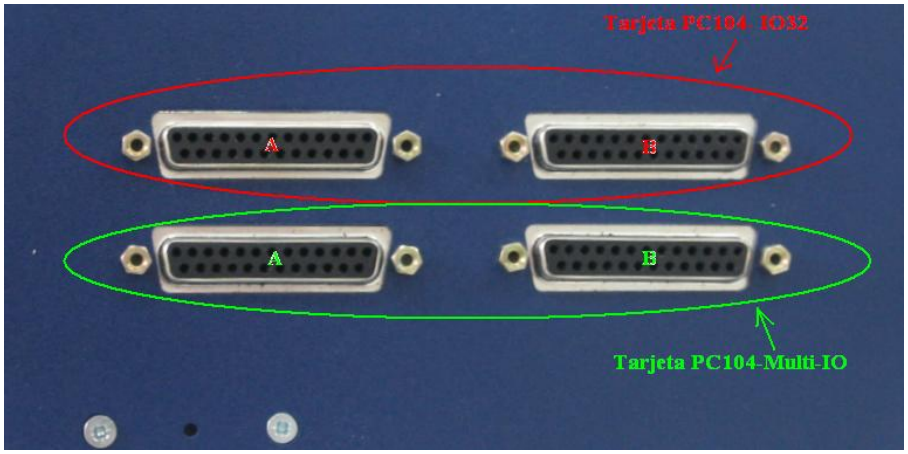


Figura 2. Vista de la parte trasera del Viper con las tarjetas PC104

3.1.1 Tarjeta PC104-IO32

Conector A		**Conector B**	
Pin 1	Canal I/O 19	Pin 1	GND
Pin 2	Canal I/O 21	Pin 2	Canal I/O 0
Pin 3	Canal I/O 23	Pin 3	Canal I/O 2
Pin 4	NC	Pin 4	Canal I/O 4
Pin 5	Canal I/O 25	Pin 5	Canal I/O 6
Pin 6	Canal I/O 27	Pin 6	GND
Pin 7	Canal I/O 29	Pin 7	Canal I/O 8
Pin 8	Canal I/O 31	Pin 8	Canal I/O 10
Pin 9	NC	Pin 9	Canal I/O 12
Pin 10	NC	Pin 10	Canal I/O 14
Pin 11	NC	Pin 11	GND
Pin 12	+12V	Pin 12	Canal I/O 16
Pin 13	+5V	Pin 13	Canal I/O 18
Pin 14	Canal I/O 20	Pin 14	GND
Pin 15	Canal I/O 22	Pin 15	Canal I/O 1
Pin 16	GND	Pin 16	Canal I/O 3
Pin 17	Canal I/O 24	Pin 17	Canal I/O 5
Pin 18	Canal I/O 26	Pin 18	Canal I/O 7
Pin 19	Canal I/O 28	Pin 19	NC
Pin 20	Canal I/O 30	Pin 20	Canal I/O 9
Pin 21	NC	Pin 21	Canal I/O 11
Pin 22	NC	Pin 22	Canal I/O 13
Pin 23	NC	Pin 23	Canal I/O 15
Pin 24	-12V	Pin 24	NC
Pin 25	+5V	Pin 25	Canal I/O 17

Tabla 1. Pinout de la tarjeta PC104 IO-32

3.1.2 Tarjeta PC104-Multi-IO

Conector A		**Conector B**	
Pin 1	Entrada anal. Canal 12 ó Canal 6+ modo diferencial	Pin 1	GND Digital
Pin 2	Entrada anal. Canal 14 ó Canal 7+ modo diferencial	Pin 2	Entrada digital Bit 1
Pin 3	PDIFF	Pin 3	Entrada digital Bit 3
Pin 4	NC	Pin 4	Entrada digital Bit 5
Pin 5	Lazo de corriente de salida canal 0 anal.	Pin 5	Entrada digital Bit 7
Pin 6	Retorno de salida canal 0 anal.	Pin 6	GND
Pin 7	Lazo de corriente de salida canal 1 anal.	Pin 7	Entrada anal. Canal 1 ó Canal 0- modo diferencial
Pin 8	Retorno de salida canal 1 anal.	Pin 8	Entrada anal. Canal 3 ó Canal 1- modo diferencial
Pin 9	NC	Pin 9	Entrada anal. Canal 5 ó Canal 2- modo diferencial
Pin 10	Salida anal. Canal 1	Pin 10	Entrada anal. Canal 7 ó Canal 3- modo diferencial
Pin 11	NC	Pin 11	GND
Pin 12	NC	Pin 12	Entrada anal. Canal 9 ó Canal 4- modo diferencial
Pin 13	NC	Pin 13	Entrada anal. Canal 11 ó Canal 5- modo diferencial
Pin 14	Entrada anal. Canal 13 ó Canal 6- modo diferencial	Pin 14	Entrada digital Bit 0
Pin 15	Entrada anal. Canal 15 ó Canal 7- modo diferencial	Pin 15	Entrada digital Bit 2
Pin 16	GND	Pin 16	Entrada digital Bit 4
Pin 17	NC	Pin 17	Entrada digital Bit 6
Pin 18	GND	Pin 18	GND Digital
Pin 19	NC	Pin 19	Entrada anal. Canal 0 ó Canal 0+ modo diferencial
Pin 20	GND	Pin 20	Entrada anal. Canal 2 ó Canal 1+ modo diferencial
Pin 21	NC	Pin 21	Entrada anal. Canal 4 ó Canal 2+ modo diferencial

Pin 22	Salida anal. Canal 0	Pin 22	Entrada anal. Canal 6 ó Canal 3+ modo diferencial
Pin 23	NC	Pin 23	PDIFF
Pin 24	NC	Pin 24	Entrada anal. Canal 8 ó Canal 4+ modo diferencial
Pin 25	NC	Pin 25	Entrada anal. Canal 10 ó Canal 5+ modo diferencial

Tabla 2. Pinout de la tarjeta PC104 Multi-IO

La tarjeta puede trabajar con entradas de tensión unipolares o en modo diferencial, para lo cual hay que operar sobre los *jumpers* LK3 y LK4, tal y como se describe en el manual. En el modo unipolar de tensión, la tierra ha de ser tomada de las señales PDIFF.

3.2) Aplicaciones

Se han escrito dos aplicaciones en C++, una para cada tarjeta. El código completo se muestra en el Apéndice II.

3.2.1 Tarjeta PC104-IO32

Para esta tarjeta se emplean las siguientes funciones, todas retornan *0* en caso de éxito y valores negativos en caso de error:

```
int aim104_io32_enable_outputs(int fd, int enable);
int aim104_io32_set_all(int fd, int channel, unsigned char set);
int aim104_io32_inputs(int fd, int channel);
```

- **aim104_io32_enable_outputs:** Sirve para indicar que los pines de la tarjeta se van a emplear como salidas.
- **aim104_io32_set_all:** Se emplea para escribir datos en las salidas.
- **aim104_io32_inputs:** Sirve para leer datos de las entradas.

3.2.2 Tarjeta PC104-Multi-IO

Para esta tarjeta se emplean las siguientes funciones, todas retornan *0* en caso de éxito y valores negativos en caso de error:

```
int aim104_multi_io_inputs(int fd);
int aim104_multi_io_ADC(int fd, int channel, int single_ended);
int aim104_multi_io_DAC(int fd, int channel,
unsigned short output);
```

- **aim104_multi_io_inputs:** Retorna el valor de las entradas digitales.
- **aim104_multi_io_ADC:** Devuelve el valor presente en las entradas analógicas
- **aim104_multi_io_DAC:** Sirve para mandar datos a las salidas analógicas

3.3) Presentación de los resultados de las aplicaciones en el monitor del Viper

El Viper del que se dispone en Yebes lleva integrado un monitor VGA de 5.5" NEC NL3224BC35, con una resolución de 320x240pxl. Las aplicaciones vistas anteriormente pueden redirigir los resultados a esta pantalla sin mas que ejecutándolas con la coletilla `>> /dev/tty1` al final.

> *Ejem*: Si el ejecutable de la aplicación que controla la tarjeta PC104-IO32 se llama *prog_io32*, para ver los resultados en la pantalla del Viper, se escribiría: ./prog_io32 >> /dev/tty1

Anteriormente, se ha de cargar el driver que nos va a permitir interactuar con la pantalla, empleando la instrucción `modprobe pxafb`. A continuación, es recomendable desactivar el apagado automático de la consola ante inactividad, con el comando `setterm -term linux -blank 0 > /dev/tty1`.

3.4) Comunicación con los puertos serie

El Viper dispone de cuatro puertos serie que le permite comunicarse con dispositivos externos mediante el protocolo RS232. Se ha desarrollado un programa en C para comunicarse con dichos puertos.

El código hace uso de estas funciones (incluidas en el fichero *adrport.h*):

```
int OpenAdrPort (char* sPortNumber);
int WriteAdrPort(char* psOutput);
int ReadAdrPort(char* psResponse, int iMax);
void CloseAdrPort();
```

- **OpenAdrPort:** Abre el puerto serie especificado.
- **WriteAdrPort:** Escribe en el puerto serie.
- **ReadAdrPort:** Lee del puerto serie.
- **CloseAdrPort:** Cierra el puerto serie.

El ejecutable es *adrserial_mod* y hay que pasarle un número, correspondiente al puerto serie que se desea emplear, menos uno. Para una aplicación que emplee el ADR101, hay que utilizar los puertos COM2 o COM3.

Ejem: si se desea operar con el puerto COM2, el programa se lanzará con *adrserial_mod 1*.

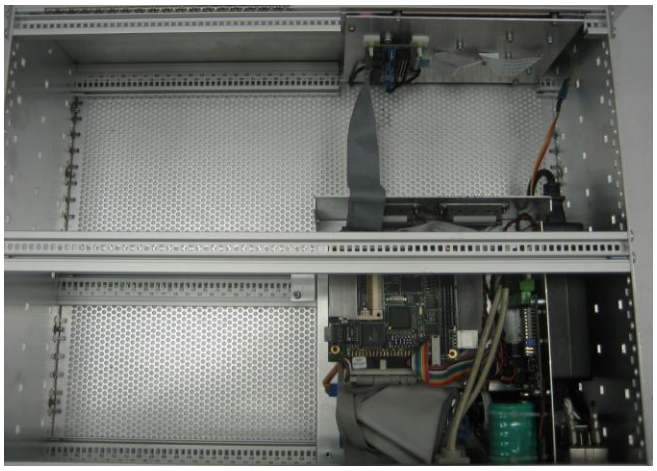
El código completo se encuentra en *root@viper:/home/arcom/serial* y se muestra en el Apéndice I.

Pin out del conexionado

Pin conector Viper (DB9 macho)	Pin conector ADR101 (DB9 hembra)
2	7
3	3
5	2

3.5) Montaje en rack

El equipo se instaló en un rack *Europac PRO* de la firma Schroff. Las fotografías del montaje final se muestran a continuación.



IV. Referencias

- www.arcom.com

- **The C Programming Language**, second edition, by Brian Kernighan and Dennis Ritchie. Ed Prentice Hall

Apéndice I: Código de las aplicaciones

Los programas se encuentran en `root@viper:/home/arcom/`

1) Tarjeta PC104-IO32

```
//prog-io32.h

#ifndef _PROG_IO32_H_
#define _PROG_IO32_H_

#include <stdio.h>
#include <errno.h>
#include <string>
#include <fcntl.h>
#include <math.h>
#include <arcom/libaim104.h>

//using namespace std;

class Aim104_io32
{
public:

        /**
         * Constructor.
         * @param device[] device number of the IO32 board : 0
         */
        Aim104_io32(const char *device);

        /**
         * Destructor.
         */
        ~Aim104_io32();



        //Esta funcion pone los canales que recibe de entrada a "1" y el resto a "0". Los
canales hay que proporcionarlos en orden creciente.
        //Ejem: Write("0","1","8","9","18","\n"); PondrÃa los canales 0, 1, 8, 9 y 18 a 5V,
el resto a 0V
        // El valor de retorno es 0 si no hubo problemas y -1 en el caso contrario
        int Write(const char write_channel_list[][3]);

        //Esta funcion lee los canales indicados en read_channel_list y devuelve 1 o 0 en
funcion de que haya 5 o 0V. Los canales hay que proporcionarlos en orden creciente.
        //Ejem: Read("0","1","8","9","18","\n"); Devolveria un puntero a un array de enteros
con los valores 0 1 0 1 1, lo que indica que el canal 0 esta a 0V, 1 a 5V, 8 a 0V, 9 a 5V y
18 a 5V.

        int *Read(const char read_channel_list[][3]);


private:
         int fd;
        int lectura_p[32];

        void binary(int);
```

```
        /**
        * Disable assignment operator.
        */
        Aim104_io32 & operator=(const Aim104_io32 &);

        /**
        * Disable copy constructor.
        **/
        Aim104_io32(const Aim104_io32 &);};

#endif
```

```
//prog-io32.cpp

#include "prog-io32.h"

int array[8],array_fin[8];
int num_bit;

Aim104_io32::Aim104_io32(const char *device)
{
        char char_device[24]="/dev/arcom/aim104/io32/";

/*      char write_channel_list[6][3]={"0","1","8","9","18","\n"};
        char read_channel_list[3][3]={"4","11","\n"};
*/
        strcat(char_device,device);

        if( (fd=open(char_device, O_RDWR ))<0 ) {
                exit(-1);
        }

}


Aim104_io32::~Aim104_io32()
{
        close(fd);

}

//Esta funcion pone los canales que recibe de entrada a "1" y el resto a "0"
//Ejem: Write("0","1","8","9","18","\n"); PondrÃa los canales 0, 1, 8, 9 y 18 a 5V, el resto
a 0V

int Aim104_io32::Write(const char write_channel_list[][3])
{
        int longitud,grupo,i,canal;
        int grupo_escritura[4]={0,0,0,0};


        if(aim104_io32_enable_outputs(fd,1)<0){
                fprintf( stderr, "Error en enable outputs\n");
                exit( 1 );
        }


        longitud=0;
        for(;;)
        {
                if(strcmp(write_channel_list[longitud],"\n")==0) break;
                longitud++;
        }

        for(i=0;i<longitud;i++){

                canal=atoi(write_channel_list[i]);

                if(canal<8) grupo=0;
                else if((canal>7)&&(canal<16)) grupo=1;
```

```
		else if((canal>15)&&(canal<24)) grupo=2;
		else if((canal>23)&&(canal<32)) grupo=3;
		else{
			printf("Canal incorrecto [0-31]\n");
			return 0;
		}

		canal=canal%8;
		canal=pow(2,canal);
		grupo_escritura[grupo]=grupo_escritura[grupo]+canal;


	}

	//set_all. Esta funcion pone 1 o 0 en el canal deseado. El 1 pone 0V (Si el Jumper
estÃ¡ en A) o 5V si el Jumper estÃ¡ en B.
	// P.Ejem: 0,0,1 pondrÃa un 1 en Ch0 => 0V con el Jump en A.
	// 0,0,0 pondrÃa un 0 en Ch0 => 5V con el Jump en A

	// 0,0,5 pondrÃa un 1 en Ch0 => 0V y Ch2 => 0V con el Jump en A.
	// 0,0,1 pondrÃa un 1 en Ch0 => 0V y un 0 en Ch2 => 5V con el Jump en A. (El Ch2
conmmutaria)

	//El segundo digito a 0 indica que ningun canal es entrada
	//Las manipulaciones de canales de un mismo grupo hay que hacerlas a la vez:
	//No se puede hacer

	//aim104_io32_set_all(fd,1,0,8) 1 en canal 11
	//aim104_io32_set_all(fd,1,0,7) 1 en canales 8,9,10
	//aim104_io32_set_all(fd,1,0,2) 1 en canal 9. 0 en canales 8 y 10
	//Tras hacer esto se tiene que el canal 11 esta a 0. Hay que hacer:

	//aim104_io32_set_all(fd,1,0,15) 1 en canales 8,9,10 y 11
	//aim104_io32_set_all(fd,1,0,10) 1 en canal 11 y 9. 0 en canales 8 y 10

	for(i=0;i<4;i++){

		if(grupo_escritura[i]!=0){
			grupo_escritura[i]=pow(2,8)-1-grupo_escritura[i];
			if longitud==0 grupo_escritura[i]=pow(2,8)-1;
			if(aim104_io32_set_all(fd,i,0,grupo_escritura[i])<0){
				fprintf( stderr, "Error en set_all\n");
				exit( -1 );
			}

		}

	}
	sleep(1);
	printf("---------------------------------\r\n");
	printf("Canales escritos OK\r\n");
	printf("---------------------------------\r\n");

	return 0;
}



//Esta funcion lee los canales indicados en read_channel_list y devuelve 1 o 0 en funcion de
que haya 5 o 0V
//Ejem: Read("0","1","8","9","18","\n"); Devolveria un puntero a un array de enteros con los
valores 0 1 0 1 1, lo que indica que el canal 0 esta a 0V, 1 a 5V, 8 a 0V, 9 a 5V y 18 a 5V.

int * Aim104_io32::Read(const char read_channel_list[][3])
{

	int
longitud=0,i,j,canal,grupo,aux,grupo_0=0,grupo_1=0,grupo_2=0,grupo_3=0,tope,canal_grupo[4][3
2];
	int canales_lectura[4]={0,0,0,0};
	int lectura[4];
```

```
for(;;)
{
        if(strcmp(read_channel_list[longitud],"\n")==0) break;
        longitud++;
}


for(i=0;i<longitud;i=i++){

        canal=atoi(read_channel_list[i]);

        if(canal<8){
         grupo=0;
         grupo_0++;
        }
        else if((canal>7)&&(canal<16)){
         grupo=1;
         grupo_1++;
        }
        else if((canal>15)&&(canal<24)){
         grupo=2;
         grupo_2++;
        }
        else if((canal>23)&&(canal<32)){
         grupo=3;
         grupo_3++;
        }

        canal_grupo[grupo][i]=canal%8;


        canal=canal%8;
        canal=pow(2,canal);
        canales_lectura[grupo]=canales_lectura[grupo]+canal;

}

aux=0;

for(i=0;i<4;i++){
        if(canales_lectura[i]!=0){

                if((lectura[i]=aim104_io32_inputs(fd,i,canales_lectura[i]))<0){
                        fprintf( stderr, "Error en inputs\n");
                        exit( 1 );
                }


                binary(lectura[i]);

                for(j=0;j<num_bit;j++){
                        array_fin[j]=array[num_bit-1-j];

                }
                if (i==0) tope=grupo_0;
                if (i==1) tope=grupo_1;
                if (i==2) tope=grupo_2;
                if (i==3) tope=grupo_3;


                for(j=0;j<tope;j++){
                        lectura_p[aux]=array_fin[canal_grupo[i][aux]];
                        aux++;
                }

        }
}
```

```
        return(lectura_p);
}

//Esta funcion convierte numeros de decimal a binario
void Aim104_io32::binary(int number) {
        int remainder;
        num_bit=0;
        if(number <= 1) {
                //printf("NUMfin %d",number);
               array[num_bit]=number;
               //escribe_adr101(number);
               num_bit++;
                return;
        }

        remainder = number%2;
        binary(number >> 1);
        //printf("NUM %d",remainder);
       array[num_bit]=remainder;
       //escribe_adr101(remainder);
       num_bit++;

}
```

```
//prog_io32_fin.cpp

#include "prog-io32.h"

//using namespace std;

int main()
{

        int i;

        int *lectura;

        Aim104_io32 prog_prueba("0"); //device 0

        char write_channel_list[6][3]={"0","1","8","9","18","\n"};
        char read_channel_list[7][3]={"4","6","11","18","27","30","\n"};

        printf("\r\n--------AIM104 IO32-------\r\n");

        lectura=prog_prueba.Read(read_channel_list);
        prog_prueba.Write(write_channel_list);


        for(i=0;i<6;i++)
                printf("Lectura canal %d : %d\r\n",atoi(read_channel_list[i]),lectura[i]);


        return 0;
}
```

2) Tarjeta PC104-Multi-IO

```
//prog-multi-io.h

#ifndef _PROG_MULTI_IO_H_
#define _PROG_MULTI_IO_H_

#include <stdio.h>
#include <errno.h>
#include <string>
#include <fcntl.h>
#include <math.h>
#include <arcom/libaim104.h>
#include <time.h>
#include <sys/timeb.h>

//using namespace std;

class Aim104_multi_io
{
public:

        /**
         * Constructor.
         * @param device[] device number of the multi-io board : 0
         */
        Aim104_multi_io(const char *device);

        /**
         * Destructor.
         */
        ~Aim104_multi_io();

        //Esta funcion escribe en la tarjeta multi-io del Viper la tensiÃ³n analÃ³gica
deseada [-5V a 5V] en el canal indicado [Ch 0 / Ch 1]
        int Escribe_analogica(double tension_analogica_deseada,int canal);

        //Lee la tensiÃ³n de las entradas analÃ³gicas. modo=1 => single ended canales [0-
15]. modo=0 => diferencial canales [0-7]
        double Lee_analogica(int canal,int modo);

        //Esta funciÃ³n retorna un numero [0-255] en funcion del estado de los 8 bits de las
entradas digitales
        int* Lee_digitales();
        //const double getTimeReference();

private:
         int fd;
        int lectura_p[8],num_bit;;
        void binary(int);
         /**
         * Disable assignment operator.
         */
         Aim104_multi_io & operator=(const Aim104_multi_io &);

         /**
         * Disable copy constructor.
         **/
         Aim104_multi_io(const Aim104_multi_io &);
};

#endif
```

```
//prog-multi-io.cpp

#include "prog-multi-io.h"
int array[8],array_fin[8];


Aim104_multi_io::Aim104_multi_io(const char *device)
{

        char char_device[28]="/dev/arcom/aim104/multi-io/";

        strcat(char_device,device);

        if( (fd=open( char_device, O_RDWR ))<0 ) {
                exit(-1);
        }
}

Aim104_multi_io::~Aim104_multi_io()
{
        close(fd);
}




//Esta funcion escribe en la tarjeta multi-io del Viper la tensiÃ³n analÃ³gica deseada [-5V
a 5V] en el canal indicado [Ch 0 / Ch 1]
int Aim104_multi_io::Escribe_analogica(double tension_analogica_deseada,int canal)
{
        int tension_analogica_comandada;

        tension_analogica_comandada=int((tension_analogica_deseada+5.)*2047./5.);

        if(aim104_multi_io_DAC(fd,canal,tension_analogica_comandada)<0){
                fprintf( stderr, "Error en DAC\n");
                exit(-1);
        }
        return(0);
}


//Lee la tensiÃ³n de las entradas analÃ³gicas. modo=1 => single ended canales [0-15]. modo=0
=> diferencial canales [0-7]
double Aim104_multi_io::Lee_analogica(int canal,int modo){

        int lectura_tension_analogica;
        double lectura_tension_long;

        if((lectura_tension_analogica=aim104_multi_io_ADC(fd,canal,modo))<0)
        {
                fprintf( stderr, "Error en ADC\n");
                exit(-1);
        }

        lectura_tension_long =-5.+(lectura_tension_analogica/2045.)*5.;
        return lectura_tension_long;

}


//Esta funciÃ³n retorna un numero [0-255] en funcion del estado de los 8 bits de las
entradas digitales
int* Aim104_multi_io::Lee_digitales(){
        int digitales,i;
        digitales=aim104_multi_io_inputs(fd);
        binary(digitales);

        for(i=0;i<8;i++){
```

```
                lectura_p[i]=0;

        }


        for(i=0;i<num_bit;i++){
                lectura_p[i]=array[num_bit-1-i];

        }



        return lectura_p;
}

void Aim104_multi_io::binary(int number) {
         int remainder;
        num_bit=0;
         if(number <= 1) {
                  //printf("NUMfin %d",number);
                array[num_bit]=number;
                //escribe_adr101(number);
                num_bit++;
                  return;
         }

         remainder = number%2;
         binary(number >> 1);
         //printf("NUM %d",remainder);
        array[num_bit]=remainder;
        //escribe_adr101(remainder);
        num_bit++;

}
```

```
//prog_multi-io_fin.cpp

#include "prog-multi-io.h"

//using namespace std;

int main()
{

        int *digitales,canal,canal_escritura,modo,i;
        double lectura,tension_escritura=1.3;

        Aim104_multi_io prog_prueba("0"); //device 0

        canal_escritura=1;
        prog_prueba.Escribe_analogica(tension_escritura,canal_escritura);

        modo=1; //0 es diferencial, 1 single ended
        canal=7;

        lectura=prog_prueba.Lee_analogica(canal,modo);

        printf("\r\n--------AIM104 Multi-IO-------\r\n");

        printf("Lectura tension analogica canal %d: %2.1f\r\n", canal,lectura);

        digitales=prog_prueba.Lee_digitales();

        for(i=0;i<8;i++)
```

```
                printf("Lectura canal digital %d : %d\r\n",i,digitales[i]);

}
```

3) Uso de los puertos serie

```
// adrserial_mod.c
//This application controls the RS232 interface connectors of the Viper

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "adrport.h"

int array[26];
int num_bit;

void binary(int);
int escribe_adr101(int);

int main(int argc, char *argv[])
{
        char sCmd[254];
        char sResult[254];
         int i;

        if (argc < 2 || argc > 2)
        {
                printf("adrserial needs 1 parameter for the serial port\n");
                printf("  ie. use 'adrserial 0' to connect to /dev/ttyS0\n");
                return 0;
        }

        printf("Type q to quit.\n\n");

        if (OpenAdrPort(argv[1]) < 0) return 0;

               strcpy(sCmd, "CPA00100000\r\n"); //PA6 entrada de alarma

               if (WriteAdrPort(sCmd) < 0) return 0;
               usleep(100000); // give the ADR card some time to respond


               strcpy(sCmd, "ma1\r\n"); //Enciendo el sint

               if (WriteAdrPort(sCmd) < 0) return 0;
               usleep(100000); // give the ADR card some time to respond

               sleep(2);

               strcpy(sCmd, "ma0\r\n"); //Apago sint

               if (WriteAdrPort(sCmd) < 0) return 0;
               usleep(100000); // give the ADR card some time to respond

               sleep(2);

               strcpy(sCmd, "ma1\r\n"); //Enciendo el sint

               if (WriteAdrPort(sCmd) < 0) return 0;
```

```
                usleep(100000); // give the ADR card some time to respond

                sleep(2);

                strcpy(sCmd, "ma0\r\n"); //Apago el sint

                if (WriteAdrPort(sCmd) < 0) return 0;
                usleep(100000); // give the ADR card some time to respond

                sleep(2);


                strcpy(sCmd, "RPA\r\n"); //Lectura del estado

                if (WriteAdrPort(sCmd) < 0) return 0;
                usleep(100000); // give the ADR card some time to respond

                sleep(2);

                if (ReadAdrPort(sCmd,i) < 0) return 0;

                printf("Lectura %s",sCmd);


        CloseAdrPort();

}
```

Apéndice II: Módulo de adquisición de datos ADAM 6017

En el Centro Astronómico de Yebes se dispone de un módulo de adquisición de datos ADAM 6017 de la firma Advantech, que cuenta con dos salidas digitales y 8 entradas analógicas diferenciales (**Figura A-1)**. Adicionalmente, este módulo tiene un conector RJ 45, lo que le permite mandar y recibir datos a través de Ethernet en paquetes UDP.

Figura A-1. ADAM 6017

1. Instalación hardware del equipo

El ADAM 6017 ha de ser alimentado con una tensión continua de entre 10 y 30V tal y como se muestra en la **Figura A-2**.

Las tensiones analógicas a leer han de instalarse en el canal deseado **(Figura A-3).**

Las salidas digitales sólo proporcionan corriente, luego para que den tensión han de ser comandadas a través de un esquema como el que se presenta en la **Figura A-4.**

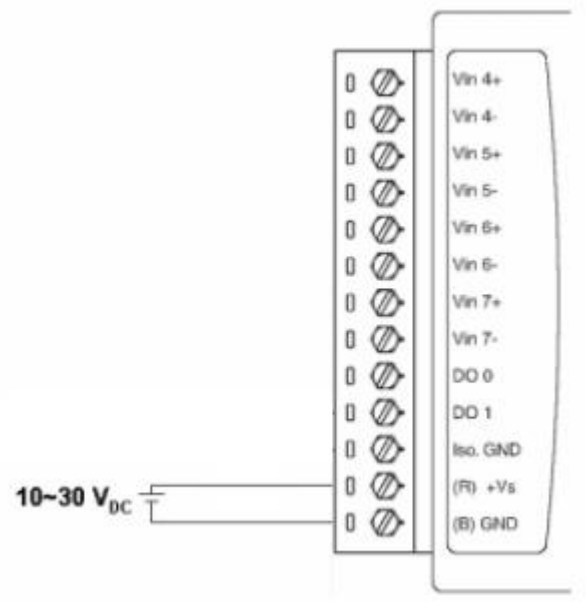


Figura A-2. Alimentación del ADAM 6017

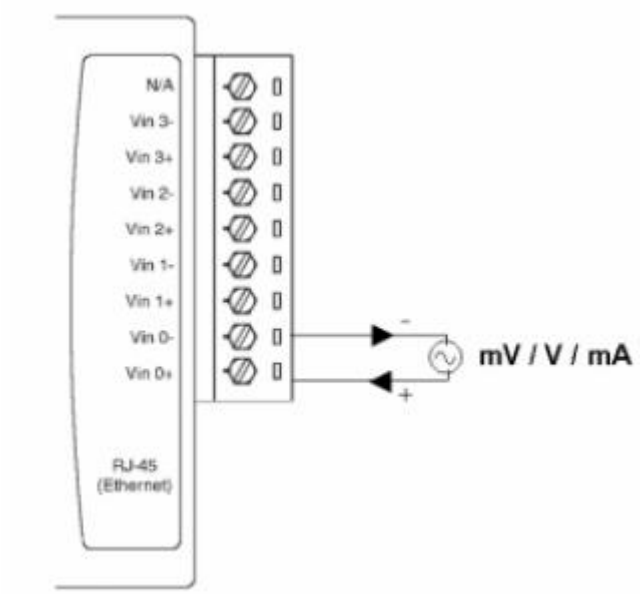


Figura A-3. Conexión de señal a las entradas analógicas

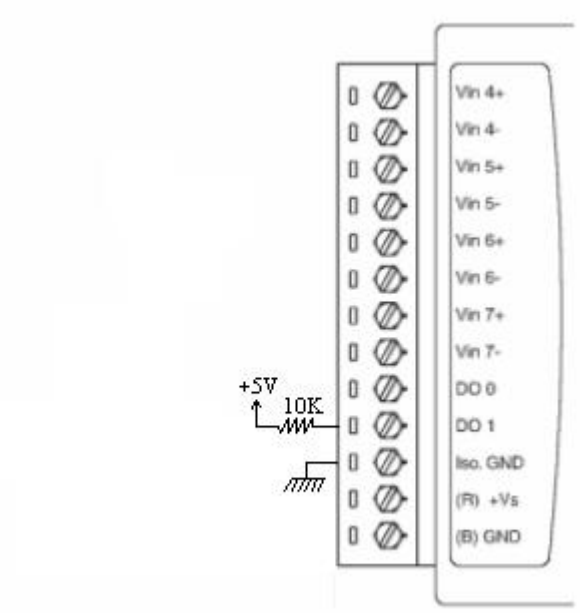


Figura A-4. Circuito driver para las salidas digitales

2. Instalación software y desarrollo de aplicaciones

El fabricante proporciona un CD con los drivers y un ejecutable tipo *Install* que automatiza el proceso de instalación. En nuestro caso, se instaló el dispositivo en un sistema operativo Linux. Una vez hecho esto, ya podemos interactuar con el equipo por medio de aplicaciones.

Se ha desarrollado una aplicación en C++ para probar las funcionalidades del módulo. En concreto, se prueba a leer un valor de tensión analógica y a escribir en una salida digital, para lo cual se mandan los siguientes comandos ASCII al módulo:

- ***$01Aff****:* Establece el tiempo de integración de la lectura de las señales de entrada analógicas.
    - *ff: Entero con el valor en mseg del tiempo de integración.*
- ***#01Annt****t*: Configura los canales de lectura.
    - *nn: Entero [0-7] con el canal que se desea leer.*
    - *tt: Modo de lectura*

tt	Rango de señal a medir
07	4 ~ 20mA
08	-10V ~ 10V
09	0V ~ 5V
0A	-1V ~ 1V
0B	0mV ~ 500mV
0C	-100mV ~ 100mV
0D	0 ~ 20mA

Tabla A-1. Configuración de las entradas analógicas

- ***#01n****:* Lee de la entrada analógica
    - *n: Entero [0-7] con el canal que se desea leer.*
- ***#01Dnd****:* Escribe en las salidas digitales
    - *n: Entero [0-1] con el canal que se desea escribir.*
    - *d: Entero [0-1] con el valor que se quiere escribir.*

El código se encuentra en `holo@holo:/home/holo/adam6017` y se muestra a continuación:

```
// adam6017.h

#ifndef _ADAM6017_H_
#define _ADAM6017_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "udpsocketclient.h"

#define RESP_LEN 200

//using namespace std;

class Adam6017
{
public:

        /**
         * Constructor.
         * @param ipnumber[] Adam6017 IP number
         */
        Adam6017(const char ipnumber[],const unsigned int & port);

        /**
         * Destructor.
         */
        ~Adam6017();

        double ReadAnalog(int channel,const char mode[]);
        /**
         * Reads output of the desired channel
         * @param mode It specifies the input range of the channel (described in the manual)
         * 07: 4-20mA // 08: -10 - 10V // 09: 0-5V // 0A: -1 - 1V // 0B: 0 - 500mV // 0C:-
100 - 100mV // 0D: 0-20mA
         */

        int SetIntegTime(int t_integ);
        /**
         * Sets integration time
         * @param t_integ With a 220V 50Hz line, this param should be 50 => integration time
= 60mseg
         * It returns a 0 if the operation was accomplished or -1 if not
         */

        int WriteDigital(int channel,int value);
        /**
         * Writes a value (0 /1) in the digital output channel (0 / 1)
         * It returns a 0 if the operation was accomplished or -1 if not
         */


private:
        char response[RESP_LEN];
        Udpsocketclient * sock;
        char cmd[RESP_LEN];
        char *str;
        int leidos;


         /**
         * Disable assignment operator.
         */
         Adam6017 & operator=(const Adam6017 &);

         /**
```

```
        * Disable copy constructor.
        **/
        Adam6017(const Adam6017 &);};

#endif
```

```
// adam6017.cpp

#include "adam6017.h"


Adam6017::Adam6017(const char ipnumber[],const unsigned int & port)
{
        sock = new Udpsocketclient(ipnumber,port);
        sock->Bind();

        sprintf(cmd,"#01F\r\n");
        sock->Write(cmd,strlen(cmd),0);
        usleep(500000);

        leidos=sock->Read(response,RESP_LEN,0);
        response[leidos-1]='\0';
        printf("Firmware Version : %s\n",response);
}

Adam6017::~Adam6017()
{
        delete sock;
}


int Adam6017::SetIntegTime(int t_integ)
{

        sprintf(cmd,"$01A%d\r\n",t_integ);
        sock->Write(cmd,strlen(cmd),0);
        usleep(500000);

        leidos=sock->Read(response,RESP_LEN,0);
        response[leidos-1]='\0';
        printf("Respuesta t_integ : %s\n",response);
        if(strcmp(response,"!01")==0)
                return 0;
        else return -1;

}

//Esta funcion pone un 1 o un 0 en la salida digital especificada
int Adam6017::WriteDigital(int channel,int value)
{
        if (value==0) value=1;
        else if (value==1) value=0;

        sprintf(cmd,"#01D%d%d\r\n",channel,value);
        sock->Write(cmd,strlen(cmd),0);
        usleep(100000);

        leidos=sock->Read(response,RESP_LEN,0);
        response[leidos-1]='\0';
        printf("Respuesta write Digital : %s\n",response);
        if(strcmp(response,"!01")==0)
                return 0;
        else return -1;

}

//Esta funcion lee de la entrada analogica especificada en channel. El modo indica el rango
de entrada aceptado (ver manual)
```

```
// Los mas comunes son 08 -> 10 a -10V, 09-> 0 a 5V, 0A->-1 a 1V, 0B->0 a 500mV, 0C-> -100 a
100mV

double Adam6017::ReadAnalog(int channel,const char mode[])
{
        sprintf(cmd,"$01A0%d%s\r\n",channel,mode);

        sock->Write(cmd,strlen(cmd),0);
        usleep(500000);

        leidos=sock->Read(response,RESP_LEN,0);
        response[leidos-1]='\0';
        printf("Config canal %d para %s \n",channel,mode);

        sprintf(cmd,"#01%d\r\n",channel);

        sock->Write(cmd,strlen(cmd),0);
        usleep(500000);
        leidos=sock->Read(response,RESP_LEN,0);
        response[leidos-1]='\0';
        str = strtok (response,">");
        printf("Lectura canal %d : %f\n",channel,atof(str));

        return atof(str);

}
```

```
// prog_adam6017.cpp

#include "adam6017.h"
#include <unistd.h>
#include <iostream>

//using namespace std;

int main()
{
        double lectura;
        int aux,i,clock,data,canal_lectura;

        Adam6017 adam6017_test("192.168.0.130",1025);

        aux=adam6017_test.SetIntegTime(50);

        canal_lectura=0;

        //Modo de lectura: Los mas comunes son 08 -> 10 a -10V, 09-> 0 a 5V, 0A->-1 a 1V,
0B->0 a 500mV, 0C-> -100 a 100mV
        lectura=adam6017_test.ReadAnalog(canal_lectura,"09");

        printf("Lectura canal %d: %f\n",canal_lectura,lectura);

        aux=adam6017_test.WriteDigital(0,0);
        sleep(4);
        aux=adam6017_test.WriteDigital(0,1);
        sleep(4);
        aux=adam6017_test.WriteDigital(0,0);
        sleep(4);
        aux=adam6017_test.WriteDigital(0,1);
        sleep(4);

        return 0;
}
```

Apéndice III: Hojas de características