



The OpenDSP General Purpose I/O ver. 1.1 to control three motor axes

OpenDSP System APPLICATION NOTE

PRELIMINARY – Ver. 0.9

INTRODUCTION.....	2
TECHNICAL SPECIFICATIONS.....	3
IMPLEMENTATION BY USING THE OPENDSP SYSTEM	5
ELECTRONIC INTERFACES	5
Digital Input Interfaces	5
Digital Output Interfaces.....	6
Differential Encoder Interfaces	8
Analog Output Interfaces.....	9
SCHEMATICS	10
DIGIN1 cable	11
DIGOUT1 cable.....	12
DIGITAL INPUT cable.....	13
DIGITAL OUTPUT cable	14
ANALOG IN cable	15
ANALOG OUT cable.....	16
SUPPLY cable	16
BOARDS	18
PLD SOFTWARE.....	18
OPENDSP GENERAL PURPOSE I/O BOARD VER. 1.1 ORIGINAL SOFTWARE ARCHITECTURE	18
The Control Unit.....	19
OPENDSP GENERAL PURPOSE I/O BOARD VER. 1.1 SOFTWARE ARCHITECTURE CHANGES	21
The Encoder Controller.....	25
The Encoder Unit	25
DSP SOFTWARE	26
THE CONSTANTS.....	27
THE ARRAYS	27
THE MACROS.....	27
AN EXAMPLE OF USE	30
Description of the program	30
Initialization	31
Use of the macros to read the encoder counters values	31
Use of the macros to force commands to the encoder counters.....	32
Use of the macros to control the digital outputs.....	32
Use of the macros to read the digital inputs values.....	33
Use of the macros to enable and disable the Differential Encoder Interfaces and the Digital Input Interfaces	33
Use of the macros to know the state of the Differential Encoder Interfaces and the Digital Input Interfaces	33
Use the test program from the Matlab command window.....	34



APPENDIX A – THE BOARD LAYOUTS	36
APPENDIX B – THE CODE OF THE ENCODER UNIT BLOCK.....	41
APPENDIX C – THE SOURCE CODE OF THE MATDSP EXAMPLE	45
BIBLIOGRAPHY	48

Introduction

The motor axis control is a very common task in the industry application. Typical applications include the control of processing machines, industrial process controls, industrial robots, plotting and printing machines, packaging machines, tool machines, industrial weighing, agricultural machinery, textile equipment, food processing, etc.

Typically a motor axis is provided with an inverter that drives the motor and controls the fast current control loop and the slower position loop. This inverter needs a position reference signal (given either with analog or digital signals) and a set of digital signals to control the motor movement.

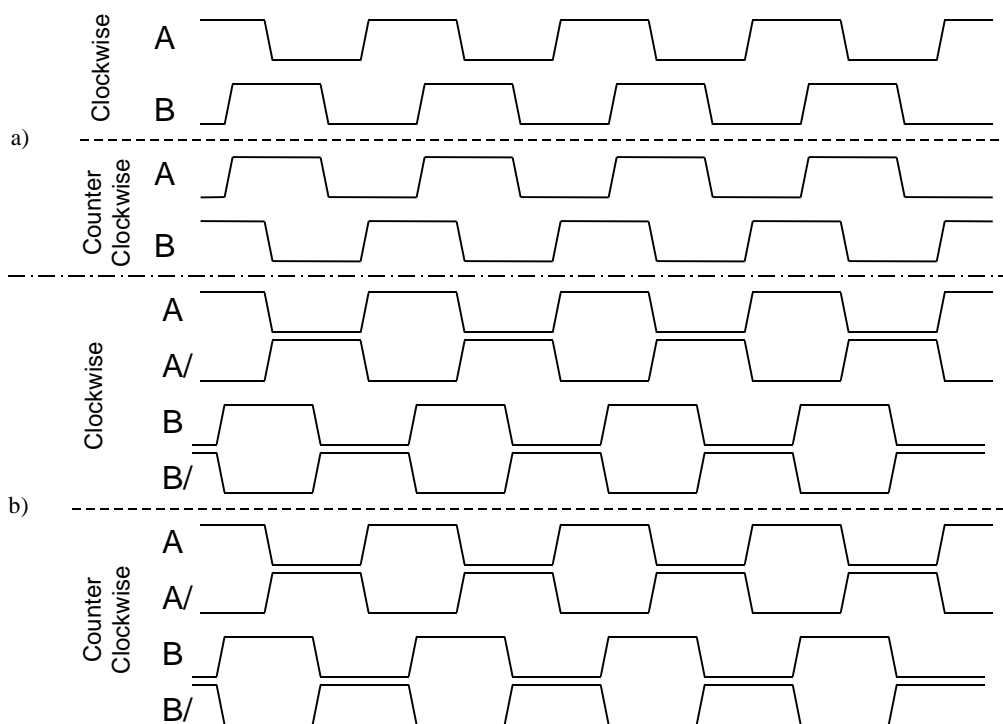


Figure 1 - Typical encoder signals: a) bidirectional unipolar encoder without zero signals, b) bidirectional differential encoder without zero signals.

Moreover, in a complex system composed by many motor axes, each axis position must be acquired to generate the correspondent position reference. The position can be obtained by using

encoders or from the axis inverter that provides the position information by means of simulated encoder signals built from the information obtained from an axial resolver.

The encoder signals depend of the encoder type and the electronic output stage. The encoders can be:

- monodirectional;
- monodirectional with zero;
- bidirectional;
- bidirectional with zero.

The electronic output stage can be unipolar or differential.

The signals of bidirectional unipolar encoders without zero are depicted in Figure 1a. The signals of bidirectional differential encoders without zero are depicted in Figure 1b.

Objective of this application note is to use the OpenDSP System (see [1] for more details) to control three motor axes, eventually designing and realizing hardware and software modifications.

Technical specifications

For this particular application, a 3 motor axes system must be connected to the OpenDSP System. In the following, the axes are respectively called Axis0, Axis1 and Axis2. All changes are designed for an OpenDSP System with box housing (see Figure 2).



Figure 2 - The OpenDSP box housing.

Each motor axis is completely defined by:

- 6 signals of a bidirectional differential encoder with zero. The signals are named A-A/, B-B/ and Z-Z/ (see Figure 3 for more details about the Z-Z/ signals). The Z-Z/ signals are relative to the zero encoder and they give a fixed position reference respect the physical 360-degree of the angular rotation. The electrical specifications are:
 - ❖ levels according to the RS422 standard,
 - ❖ $5\text{ V} \pm 5\%$ supply voltage,
 - ❖ 300 mA maximum current supply,
 - ❖ 1 MHz maximum frequency signal.
- 4 digital input signals with voltage levels of 0 V to the low level, 24 V to the high level. These signals are named FC, Reset, Auxiliary Input 1 and Auxiliary Input 2.
- 3 digital output signals with voltage levels of 0 V for the low level, 24 V for the high level; the outputs must be able to drive resistive loads with 0.5 A maximum current. These signals are START (Start/Stop motor drive), Auxiliary Output 1 and Auxiliary Output 2.
- 1 analog output signal with $\pm 10\text{ V}$ voltage range, named V_{REF} , used as reference to the motor drive; the outputs must be able to drive resistive loads with 5 mA maximum current.

In addition to these specifications, a 0-24V digital input is provided. This line, called External Latch, is used to trigger a latch that stores the value of the encoder counter that represents the encoder position (see “PLD software” for more details).

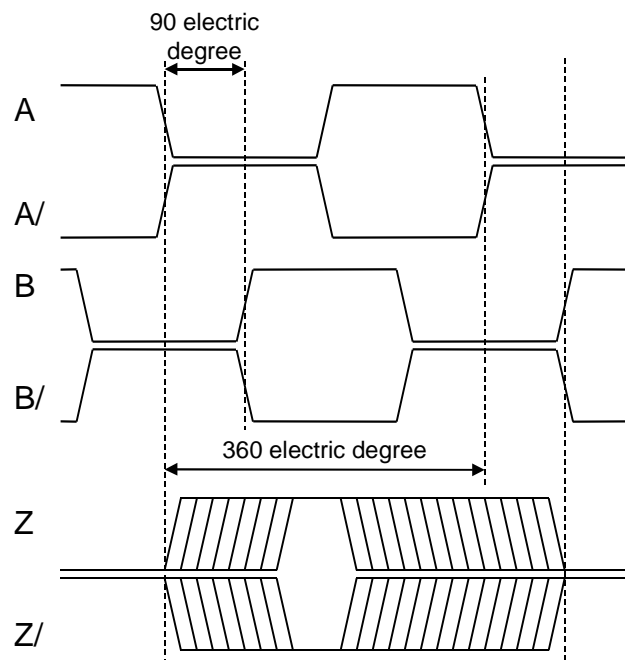


Figure 3 - Differential encoder signals sequence with clockwise rotation.



Implementation by using the OpenDSP System

In order to satisfy the technical specifications listed into the chapter “Technical specifications”, it is necessary to use a board with both digital and analog signals.

The OpenDSP General Purpose I/O ver. 1.1 accepts both digital and analog signals and provides both digital and analog outputs (see [2] for more details). The electrical specifications of the interfaces are quite different from that request for this specific application. These differences require the realization of electronic interfaces.

Electronic interfaces

Based on the characteristics described in “Technical specifications”, it is possible to group the electronic interfaces in:

- conversion of the digital input signals from 0-24V to 0-5V TTL compatible levels; this circuit must be able to electrically insulate the internal board logic signals from the signals carried from the field;
- conversion of the digital output signals from 0-5V TTL compatible levels to 0-24V; this interface must be able to supply until 500 mA with resistive loads; this circuit must be able to electrically insulate the internal board logic signals from the signals carried from the field;
- conversion of the analog output signals from $\pm 5V$ to $\pm 10V$; this interface must be able to supply until 5 mA with resistive loads;
- conversion of the differential encoder signals from RS422 compatible levels to 0-5V TTL compatible levels.

Digital Input Interfaces

The digital input interface can be implemented by using optocouplers. This interface guarantees the required electrical insulation.

Each 0-24V digital input must be provided with its interface. The electronic scheme of a digital input is shown in Figure 4b.

The chosen optocoupler is the HP 6N137; it has also an enable input that allows controlling the external input by software. More details about the software implementation are in “PLD software” and “DSP software”. The 6N137 internal scheme is shown in Figure 4a and its technical references are in [3]; the main characteristics are reported in Table 1. The maximum reverse input voltage allows 0V-undershoot until -5V.

As shown in Figure 4b, the electric scheme is very simple. The input resistor R_{LIM} is used to polarize the 6N137 input diode. The pull-up resistor connected to the output is necessary because the output is an open collector stage; the value is chosen according to the technical references. The capacitor connected between the supply pins and placed near to the device is recommended by the manufacturer to guarantee proper supply bypassing.

The enable input has also a pull-up resistor. This is required because all the axis enable inputs are driven by the same PLD output and the PLD maximum output current is 25 mA, as described in ALTERA technical references [4].

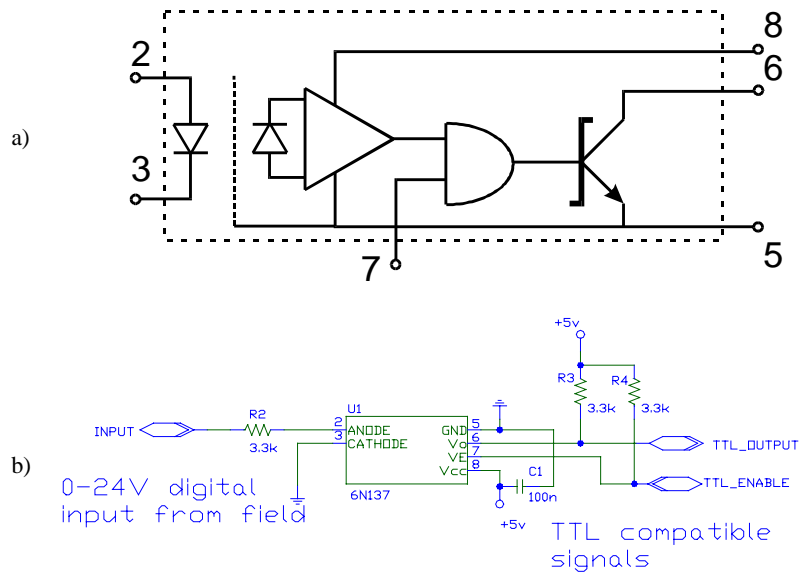


Figure 4 - a) HP 6N137 optocoupler internal scheme. b) Digital Input Interface.

Table 1 - 6N137 technical characteristics summary.

Description	min	typ	max	MU
V_{CC} – Power supply voltage		5		V
I_{OH} – High level output current		5.5	100	μA
V_{OL} – Low level output voltage		0.35	0.6	V
V_F – Input forward voltage	1.3		1.8	V
I_{FL} – Input current, low level	0		250	μA
I_{FH} – Input current, high level	5		15	mA
R_{PU} – Output pull-up resistor	300		4000	Ω
V_{EL} – Low level enable voltage	0		0.8	V
V_{EH} – High level enable voltage	2		5	V
I_{EH} – High level enable current	-0.7		-1.6	mA
I_{EL} – Low level enable current	-0.9		-1.6	mA
V_R – Reverse input voltage			5	V

Digital Output Interfaces

Each 0-24V digital output requires a signal conditioning.

In order to provide the required output current, the optocoupler use is advised against because it requires additional power electronics.

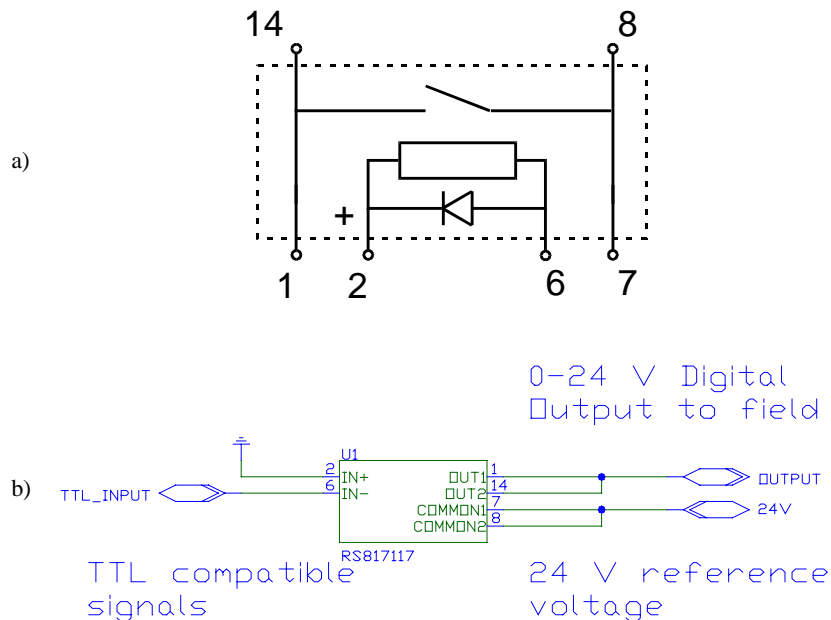


Figure 5 - a) RS 817-117 reed relay internal schema. b) Digital output interface.

Table 2 – RS 817-117 technical characteristics summary.

Description	min	typ	max	MU
I_{SW} – Switched current			0.5	A
V_{SW} – Switched voltage			100	V
P_{SW} – Switched power			10	W
V_{ISCC} – Insulation voltage contact to contact		250		V
V_{ISCW} – Insulation voltage contact to wire		500		V
R_{INC} – Initial contact resistance		0.15		Ω
R_{ISCC} – Insulation resistance contact to contact		10^9		Ω
R_{ISCW} – Insulation resistance contact to wire		10^9		Ω
t_{SW} – Switching time		0.5		ms
R_W – Wire resistance		500		Ω

Being not important the output commutation speed, a reed relay can be used. The chosen device has the RS Components part number 817-117; its outputs have normally open contacts.

Figure 5a shows the internal schema and Table 2 reports its technical references; notice that the insulation specifications are satisfied.

This device supplies a maximum output current of 500 mA and it can be driven by TTL compatible signals. The input current is 10 mA, thus the PLD output can be connected directly to the relay inputs. Figure 5b shows the electric scheme used.

The 24V power voltage and its relative ground reference must be supplied from the field on the Digital Output connector. This external voltage reference is also used to generate the power supply for the analog output interface, as described in the paragraph “Analog Output Interfaces”, then it is necessary to connect always these external voltage reference.

Differential Encoder Interfaces

Starting from the encoder differential signals, it is possible to obtain the TTL compatible signals by using a RS422 line receiver.

Many manufacturers build devices containing more than one RS422 line receiver; typically each Integrated Circuit (IC) contains 4 line receivers. The chosen IC is the Texas Instruments SN75175, whose technical references are in [3] and whose relative pinout is reported in Figure 6a. This device contains 4 line receivers that can be enabled in pairs. Also this feature allows controlling the encoder inputs via software.

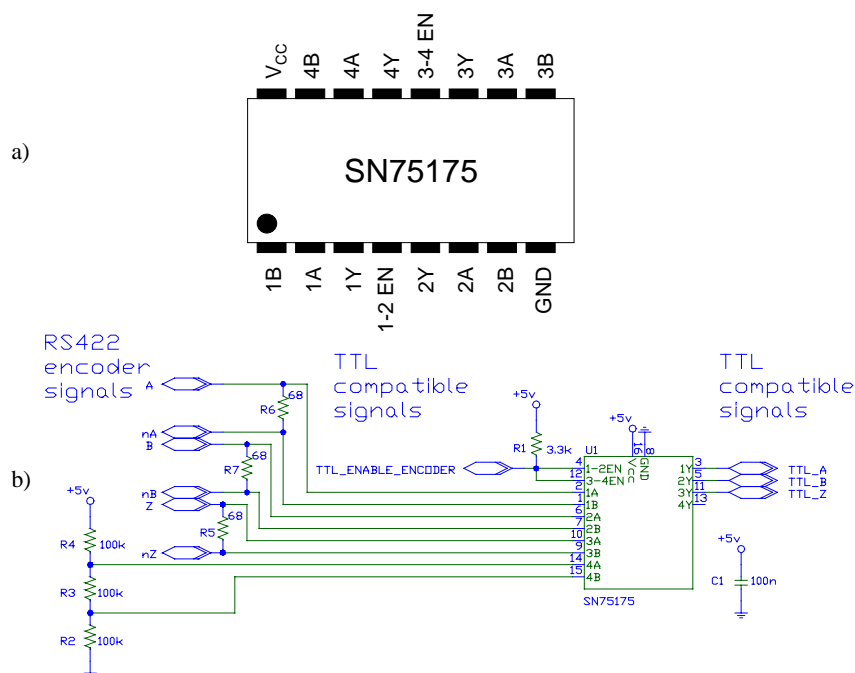


Figure 6 - a) TI SN75175 4 RS422 line receivers pinout. b) Differential encoder interface.

Each motor axis can be interfaced with one SN75175 in which 3 line receivers are used to obtain the A, B and Z TTL compatible signals. The fourth line receiver is not used; in order to minimize the noise, a polarization circuit is provided as shown in Figure 6b.

In order to reduce the line reflections, matching impedance is provided for each differential input; the chosen value is 68Ω .

Notice that the enable lines have a pull-up resistor. This allows the PLD to drive the RS422 line receivers enable and the Digital Input Interface of the External Latch signal.

Analog Output Interfaces

The OpenDSP General Purpose I/O Board ver. 1.1 provides 8 analog outputs with $\pm 5V$ voltage range. The specifications require analog voltage outputs range of $\pm 10V$. Thus, a non-inverting operational amplifier configuration is needed; obviously, the multiplier factor is 2.

No frequency band specifications are provided, thus an internally compensated operational amplifier can be chosen. For the 3 axes, a 4 operational amplifier single chip can be used. The LM348N, manufactured by National Instruments, has been chosen. Its technical references are in [6] and its pinout is shown in Figure 7a.

The analog output interface is very simple, as shown in Figure 7b. The resistors connected to the non-inverting pin are useful to reduce the effects of the operational amplifier bias and offset currents.

Since the required voltage range is $\pm 10V$, it is possible to supply the operational amplifier with $\pm 15V$: a d.c./d.c. converter is required. The TRACO NMA2415D is a device that converts 24V into $\pm 15V$; the IC pinout is shown in Figure 8a and the scheme to connect this device is really simple and it is shown in Figure 8b.

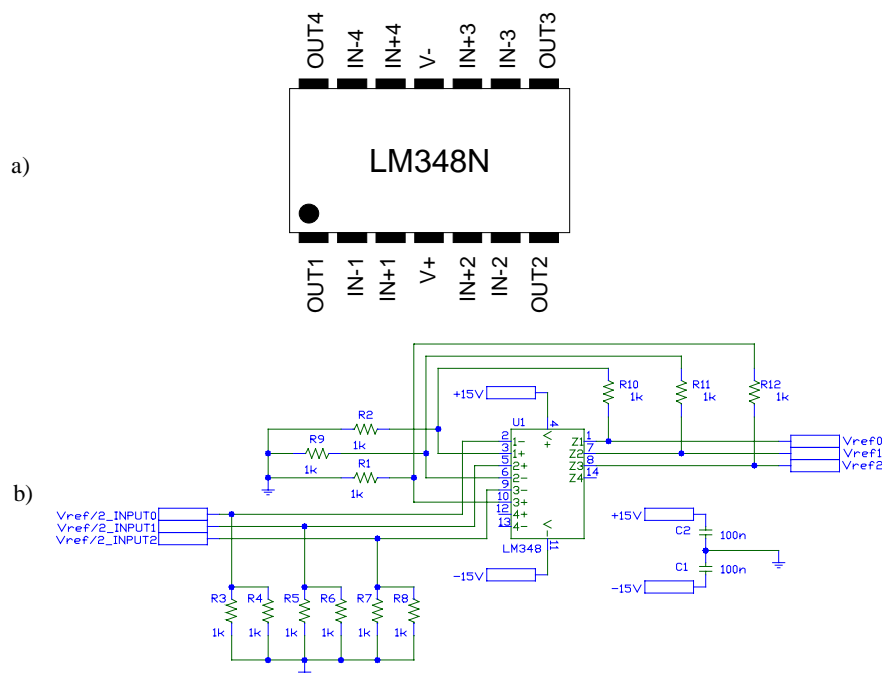


Figure 7 - a) National Instruments LM348N 4 $\mu A741$ operational amplifiers pinout.
b) Analog output interface.

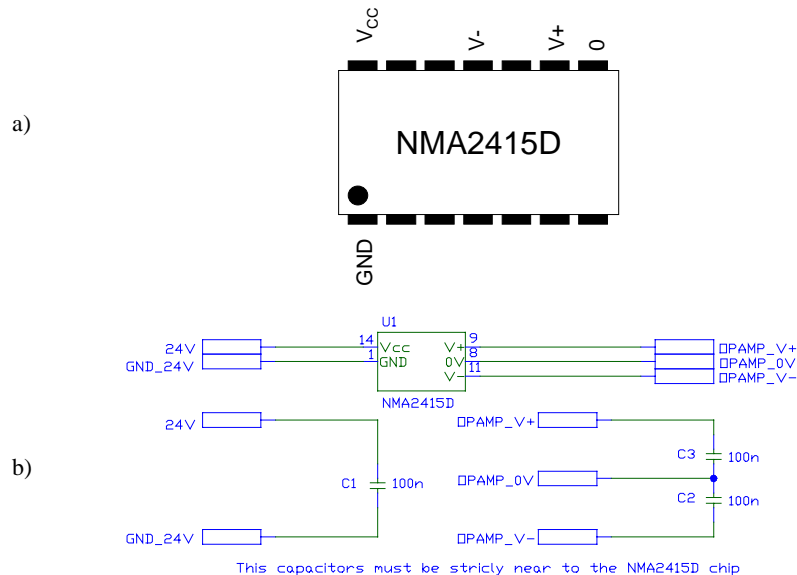


Figure 8 - a) TRACO NMA2415D d.c./d.c. converter pinout. b) 24V supply circuitry.

Schematics

The electric interface is realized in four boards:

- **Analog Interface Board** this board contains the 24V supply circuitry and the Analog Output Interfaces. It is connected to the OpenDSP General Purpose I/O Board ver. 1.1 by using the ANALOGIN cable and it is connected to the front panel by using the ANALOGOUT cable. The first three analog outputs carried from the ANALOGIN cable are used for the analog output interfaces inputs; the other analog outputs and all the analog inputs are connected directly to the ANALOGOUT connector.
- **Axis0 Interface Board** this board contains the interfaces for the Axis0 motor signals. In details, this board contains:
 - ❖ 1 Differential Encoder Interface;
 - ❖ 1 Digital Input Interface for the External Latch signal;
 - ❖ 4 Digital Input Interfaces for the 0-24V digital input signals;
 - ❖ 3 Digital Output Interfaces for the 0-24V digital output signals.

It is connected to the OpenDSP General Purpose I/O Board ver.1.1 by using the DIGIN1 and DIGOUT1 cables and it is connected to the front panel by using the DIGITALINPUT and DIGALOUTPUT cables.



- **Axis1 Interface Board** this board is similar to the Axis0 Interface Board and it interfaces to the Axis1 motor signals. Also the connections to the OpenDSP General Purpose I/O Board ver.1.1 and the front panel are the same of those described for the Axis0 Interface Board.
- **Axis2 Interface Board** this board is similar to the Axis0 Interface Board and it interfaces to the Axis2 motor signals. Also the connections to the OpenDSP General Purpose I/O Board ver.1.1 and the front panel are the same of those described for the Axis0 Interface Board.

An additional cable is used to connect all the four interface boards. This cable named SUPPLY is connected to the SUPPLY connectors and it carries the 24V reference voltage and the 5V supply voltage.

Blocks similar to those described in the “Electronic interfaces” section compose the schematics. In addition, some connectors are used to allow the connection among each interface board, the OpenDSP General Purpose I/O Board ver. 1.1 and the front panel by means of the ANALOGIN, ANALOGOUT, DIGIN1, DIGOUT1, DIGITALINPUT, DIGITALOUTPUT and SUPPLY cables.

DIGIN1 cable

A 50-wire flat cable and four PCB female flat connectors compose this cable. It connects the OpenDSP General Purpose I/O Board ver. 1.1 DIGIN1 connector with relate axis interface board connectors.

In the original version of the OpenDSP General Purpose I/O Board ver. 1.1, the DIGIN1 connector contains 32 TTL compatible inputs; in this application, all these signals have a new meaning listed in Table 3.

All pins named “Not Used” are connected to the connector and to the PLD on the OpenDSP General Purpose I/O Board ver. 1.1, but they are not routed in the boards that compose this application. Thus, they are usable if the PLD program is changed and downloaded in the PLD and the cable is connected to new boards.

Table 3 - DIGIN1 cable map.

Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side	Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side
1	GND	GND	26	AXIS0_FC	DATA_IN16
2	AXIS0_IN_AUX1	DATA_IN0	27	AXIS0_ENC_A	DATA_IN17
3	AXIS0_ENC_Z	DATA_IN1	28	GND	GND
4	GND	GND	29	AXIS0_ENC_B	DATA_IN18
5	AXIS0_IN_AUX2	DATA_IN2	30	AXIS0_RESET	DATA_IN19
6	AXIS1_IN_AUX2	DATA_IN3	31	GND	GND



Table 3 – Cont.

Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side	Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side
7	GND	GND	32	AXIS1_RESET	DATA_IN20
8	AXIS1_IN_AUX1	DATA_IN4	33	AXIS1_ENC_B	DATA_IN21
9	AXIS1_ENC_Z	DATA_IN5	34	GND	GND
10	GND	GND	35	AXIS1_ENC_A	DATA_IN22
11	AXIS0_XLATCH	DATA_IN6	36	AXIS1_FC	DATA_IN23
12	AXIS1_XLATCH	DATA_IN7	37	GND	GND
13	GND	GND	38	Not Used	GND
14	Not used	DATA_IN8	39	GND	GND
15	AXIS2_XLATCH	DATA_IN9	40	AXIS2_FC	DATA_IN24
16	GND	GND	41	AXIS2_ENC_A	DATA_IN25
17	AXIS2_ENC_Z	DATA_IN10	42	GND	GND
18	AXIS2_IN_AUX1	DATA_IN11	43	AXIS2_ENC_B	DATA_IN26
19	GND	GND	44	AXIS2_RESET	DATA_IN27
20	AXIS2_IN_AUX2	DATA_IN12	45	GND	GND
21	Not used	DATA_IN13	46	Not used	DATA_IN28
22	Not used	GND	47	Not used	DATA_IN29
23	Not used	DATA_IN14	48	Not used	GND
24	Not used	DATA_IN15	49	Not used	DATA_IN30
25	Not used	GND	50	Not used	DATA_IN31

DIGOUT1 cable

A 50-wire flat cable and four PCB female flat connectors compose this cable. It connects the OpenDSP General Purpose I/O Board ver. 1.1 DIGOUT1 connector with relate axis interface board connectors.

In the original version of the OpenDSP General Purpose I/O Board ver. 1.1, the DIGOUT1 connector contains 32 TTL compatible outputs; in this application, all these signals have a new meaning listed in Table 4.

All pins named “Not Used” are connected to the connector and to the PLD on the OpenDSP General Purpose I/O Board ver. 1.1, but they are not routed in the boards that compose this application. Thus, they are usable if the PLD program is changed and downloaded in the PLD and the cable is connected to new boards.



Table 4 - DIGOUT1 cable map.

Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side	Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side
1	GND	GND	26	AXIS0 START	DATA_IN16
2	AXIS0 ENABLE ENCODER	DATA_IN0	27	AXIS0 AuxOUT0	DATA_IN17
3	AXIS0 ENABLE INPUT	DATA_IN1	28	GND	GND
4	GND	GND	29	AXIS0 AuxOUT1	DATA_IN18
5	AXIS1 START	DATA_IN2	30	Not Used	DATA_IN19
6	AXIS1 AuxOUT0	DATA_IN3	31	GND	GND
7	GND	GND	32	AXIS1 ENABLE ENCODER	DATA_IN20
8	AXIS1 AuxOUT1	DATA_IN4	33	AXIS1 ENABLE INPUT	DATA_IN21
9	Not Used	DATA_IN5	34	GND	GND
10	GND	GND	35	Not Used	DATA_IN22
11	Not Used	DATA_IN6	36	Not Used	DATA_IN23
12	Not Used	DATA_IN7	37	Not used	GND
13	Not used	GND	38	Not used	GND
14	Not used	DATA_IN8	39	Not used	GND
15	Not Used	DATA_IN9	40	Not Used	DATA_IN24
16	Not used	GND	41	Not Used	DATA_IN25
17	Not Used	DATA_IN10	42	GND	GND
18	AXIS2 AuxOUT1	DATA_IN11	43	AXIS2 ENABLE INPUT	DATA_IN26
19	GND	GND	44	AXIS2 ENABLE ENCODER	DATA_IN27
20	AXIS2 AuxOUT0	DATA_IN12	45	GND	GND
21	AXIS2 START	DATA_IN13	46	Not used	DATA_IN28
22	GND	GND	47	Not used	DATA_IN29
23	Not used	DATA_IN14	48	Not used	GND
24	Not used	DATA_IN15	49	Not used	DATA_IN30
25	Not used	GND	50	Not used	DATA_IN31

DIGITALINPUT cable

A 50-wire flat cable, three PCB female flat connectors and a front panel flat connector compose this cable. It connects the front panel DIGITAL INPUT connector with the DIGITALINPUT axis interface board connectors.

This particular application requires mapping again the signals on the cable and connectors. Figure 9a shows the front panel connector map whereas Figure 9b shows the PCB connectors map. Remember that the PCB connector is soldered on each single axis interface board: only the specific signals are routed from these connectors. Notice that the GND signals are internally connected together and they are connected to the +5V ground reference.

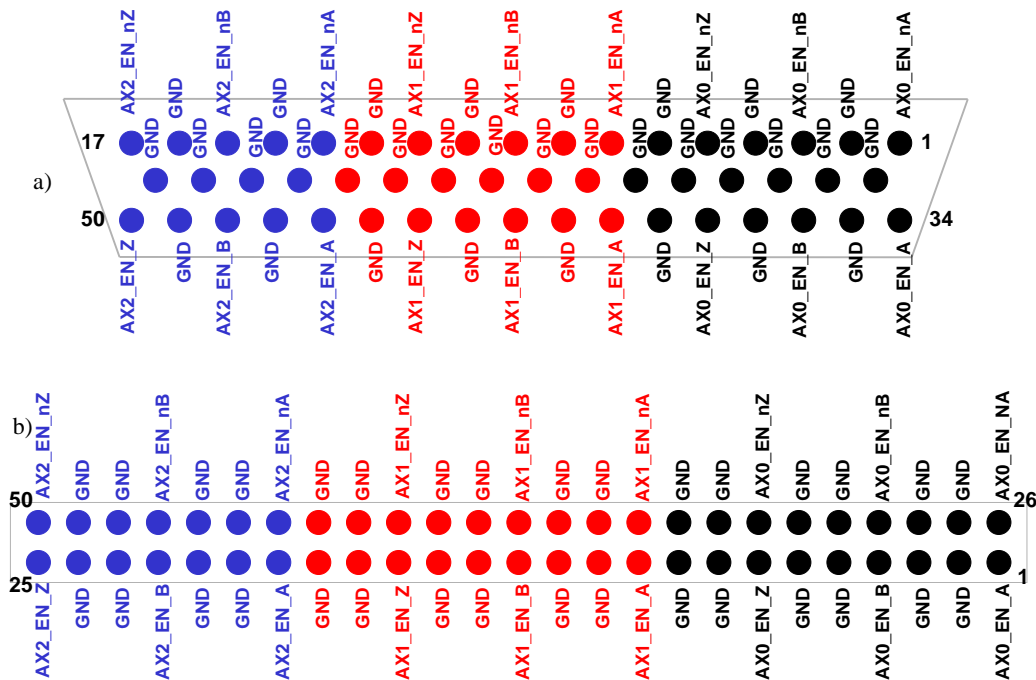


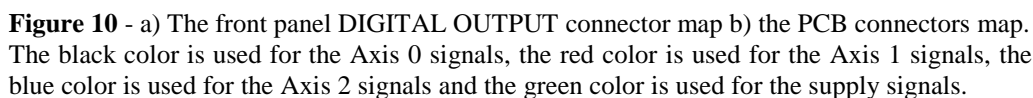
Figure 9 - a) The front panel **DIGITAL INPUT** connector map b) the PCB connectors map. The black color is used for the Axis 0 signals, the red color is used for the Axis 1 signals and the blue color is used for the Axis2 signals.

DIGITALOUTPUT cable

A 50-wire flat cable, three PCB female flat connectors and a front panel connector compose this cable. It connects the front panel **DIGITAL OUTPUT** connector with the **DIGITALOUTPUT** axis interface board connectors.

Also this cable requires mapping again the signals. Figure 10a shows the front panel connector map with indication of signals and relate ground reference whereas Figure 10b shows the PCB connectors map. Also this cable is connected to all single axis boards and only the specific signals are routed from these connectors. Notice that the GND signals are internally connected together and they are connected to the +24V ground reference; this ground reference is electrically insulated from the +5V ground reference.

This connector is used to supply the +24V voltage, thus, in order to make the system run properly, the 24V_SUPPLY pin and the GND_SUPPLY pin depicted in Figure 10a must be connected.



A 34-wire flat cable and two PCB female flat connectors compose this cable. It must be plugged between the OpenDSP General Purpose I/O Board ver.1.1 ANALOG1 connector and the Analog Interface Board. The first three analog outputs are used with the meanings of the reference voltage V_{REF} . Table 5 shows the new meanings of the signals.

Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side	Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side
1	CHANNEL0	CHANNEL0	26	GND	GND
2	GND	GND	27	CHANNEL1	CHANNEL1
3	CHANNEL2	CHANNEL2	28	GND	GND
4	GND	GND	29	CHANNEL3	CHANNEL3
5	CHANNEL4	CHANNEL4	30	GND	GND



Table 5 – Cont.

Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side	Pin number	Application board side	OpenDSP General Purpose I/O Board ver. 1.1 side
6	GND	GND	31	CHANNEL5	CHANNEL5
7	CHANNEL6	CHANNEL6	32	GND	GND
8	GND	GND	33	CHANNEL7	CHANNEL7
9	VCC	VCC	34	GND	GND
10	AXIS0 $V_{REF}/2$	VoutA_DAC0	35	VCC	VCC
11	GND	GND	36	AXIS1 $V_{REF}/2$	VoutB_DAC0
12	AXIS2 $V_{REF}/2$	VoutC_DAC0	37	GND	GND
13	GND	GND	38	VoutD_DAC0	VoutD_DAC0
14	VoutA_DAC1	VoutA_DAC1	39	GND	GND
15	GND	GND	40	VoutB_DAC1	VoutB_DAC1
16	VoutC_DAC1	VoutC_DAC1	41	GND	GND
17	GND	GND	42	VoutD_DAC1	VoutD_DAC1

ANALOGOUT cable

A 34-wire flat cable, a 34-pins PCB female flat connector and a 37-pins front panel flat connector compose this cable. It must be plugged between the front panel ANALOG I/O connector and the ANALOGOUT connector of the Analog Interface Board. The first three analog outputs are used to generate the reference voltage V_{REF} . Figure 11a shows the front panel connector map, whereas the Figure 11b shows the PCB connector.

SUPPLY cable

A 4-wire cable and four PCB female wire connectors compose this cable. It connects the single axis interface board and the analog interface board SUPPLY connector among them. Table 6 shows the new meanings of the signals.

Table 6 - SUPPLY cable map.

Pin number	Description
1	+24 V
2	+24 V ground reference
3	+5 V
4	+5 V ground reference

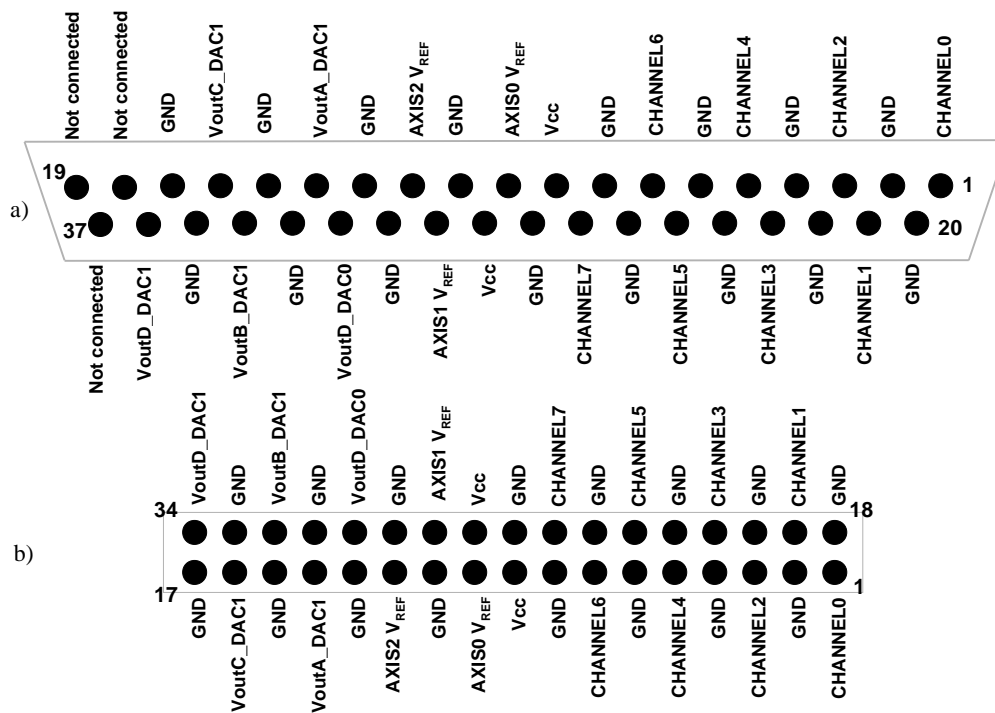


Figure 11 - a) The front panel ANALOGOUT connector map b) the PCB connectors map.

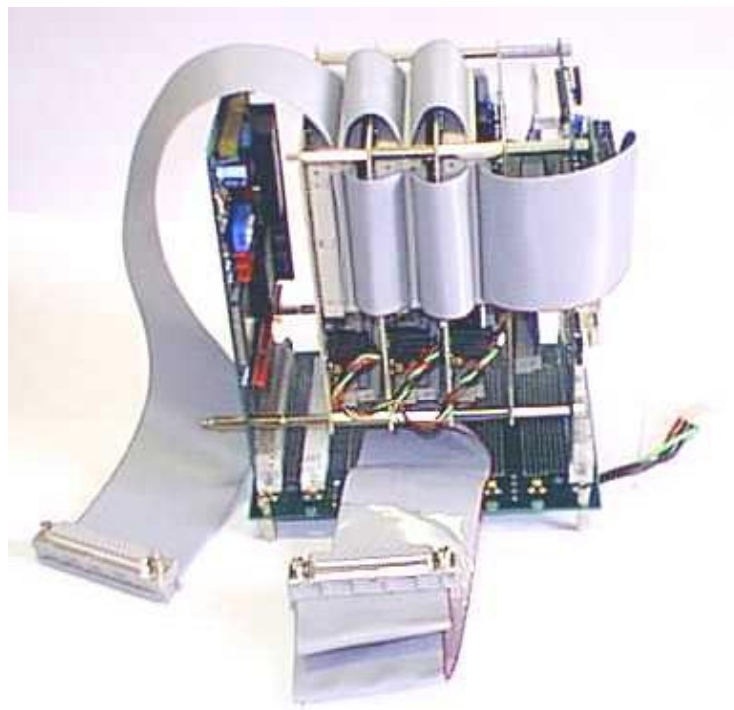


Figure 12 - The OpenDSP System boards mounted with the electronic interfaces designed and realized for this application.

Boards

Each interface board is realized to be directly mounted over the OpenDSP General Purpose I/O Board ver. 1.1. The connectors layout eases the plugging of the cables; Figure 12 shows the OpenDSP System boards mounted with the electronic interfaces boards. All components are arranged in order to find the interfaces described in “Electronic interfaces” easily.

The quoted board layouts and the photos of the realized prototype are in “Appendix A – The board layouts”.

PLD software

In order to interface the OpenDSP System to three motor axes according to the technical specifications taken on here, some PLD software changes are required.

Before to start the detailed analysis of the changes, a brief discussion of the OpenDSP General Purpose I/O Board ver.1.1 PLD existing software is reported (see [2] for more details).

OpenDSP General Purpose I/O Board ver. 1.1 original software architecture

It is possible to highlight five main blocks in the original PLD software architecture of the OpenDSP General Purpose I/O Board ver. 1.1, as shown in Figure 13.

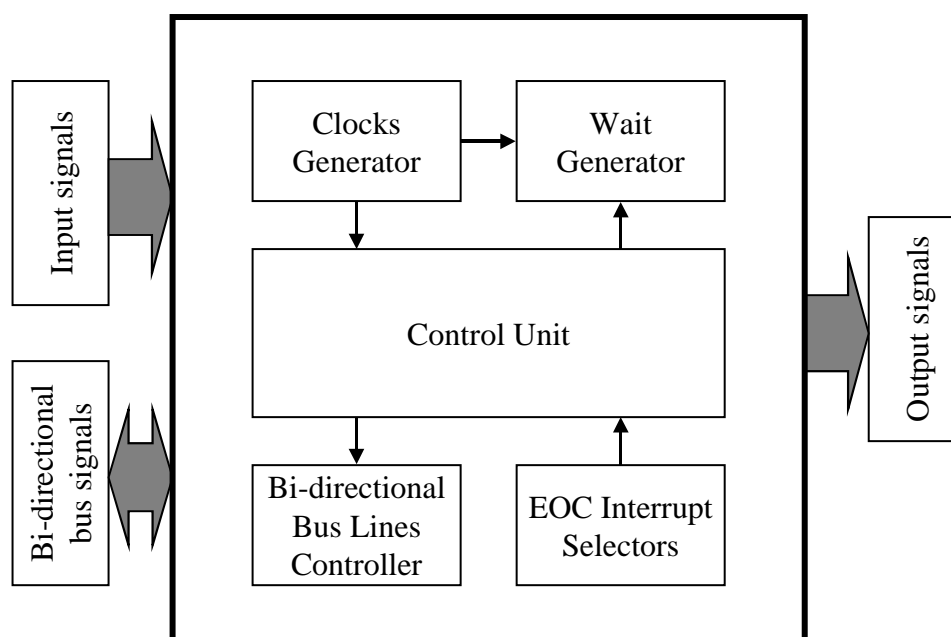


Figure 13 - OpenDSP General Purpose I/O Board ver.1.1 original PLD software architecture.

As shown in figure, the blocks are:

- **Clocks generator** this block generates the PLD internal clock and the clock used to synchronize the ADCs management; the Control Unit and the Wait Generator use these signals.

- **EOC Interrupt Selectors** the I/O Board can signal the ADC End Of Conversion (EOC) by generating an interrupt that can be selected by means of DIP switches. This block provides four signals to the Control Unit; only one at a time is signaled, according to the interrupt selected.
- **Control Unit** this block is the core of the PLD software architecture. This block controls the accesses to and from the OpenDSP Bus, the Start Of Conversion (SOC) of ADCs and DACs, the ADCs timings, the DACs timings, the ADC EOC signal generation.
- **Bi-directional Bus Lines Controller** this block controls the bi-directional bus lines (ADC Data Bus and OpenDSP Data Bus); it controls also the auxiliary I/O, present in the OpenDSP Bus and not used in this board.
- **Wait Generator** this block controls the MDB_nWAIT signal according to the timings of the electronic devices mounted on board and controlled by the Control Unit. For more details about the OpenDSP Bus signals, see [7].

The Control Unit

Some details about the Control Unit must be done. This block contains other blocks that carry out specific jobs. The architecture of the Control Unit is shown in Figure 14.

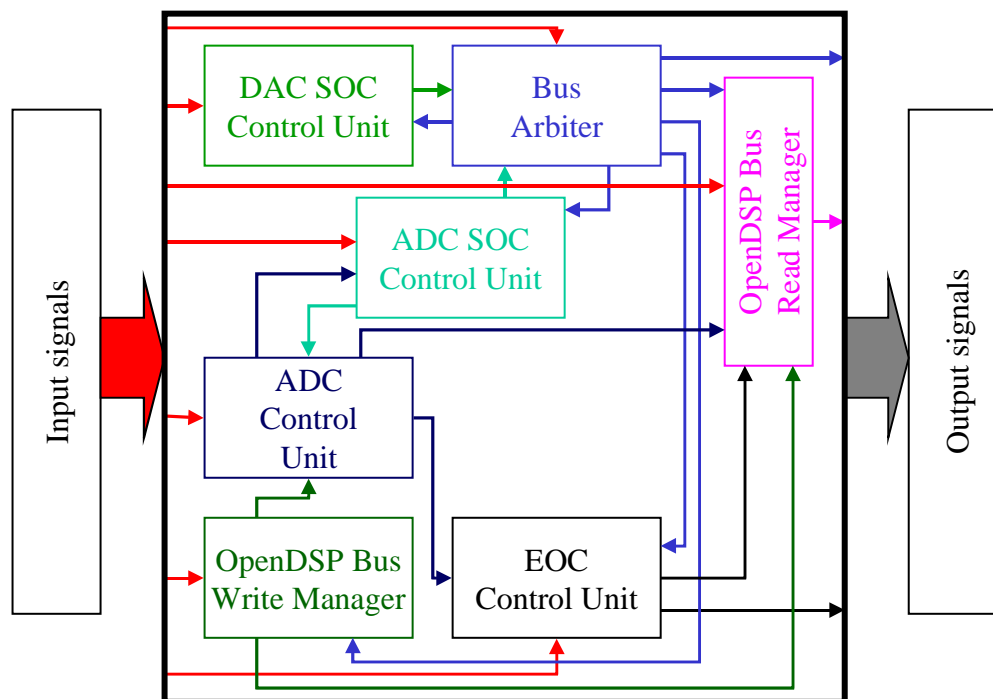


Figure 14 - The Control Unit architecture.

In details, each block performs the following function:

- **DAC SOC Control Unit** this block controls the Start Of Conversion (SOC) for the DACs. It generates three SOC signals, triggered by the T_{CLK0} , T_{CLK1} signals (see [7] for more details about the OpenDSP Bus signals) and via software.



- **Bus Arbiter** this block controls the direction of the ADC data bus, a signal used to control the MDB_nWAIT line, the DACs control lines and the End Of Conversion (EOC) signal.
- **ADC SOC Control Unit** this block controls the SOC for the ADC, generating it by the chosen source: T_{CLK0}, T_{CLK1} signals or via software.
- **ADC Control Unit** this block controls the ADC by means of the ADC control lines, the ADC data bus and the EOC signal. This block is strictly related to the IC of the ADC used.
- **EOC Control Unit** this block generates the EOC signal used by the Bus Arbiter block to generate the same signal used to generate an interrupt. It allows specifying until eight EOC sources; the EOC signal is generated if all the EOC sources selected has generated its signal.
- **OpenDSP Bus Write Manager** this block is used to control an OpenDSP Bus write operation and to set the internal registers according to the value written on the OpenDSP Data Bus. The registers map is shown in Table 7.
- **OpenDSP Bus Read Manager** this block is used to control an OpenDSP Bus read operation based on the internal registers. The registers map is shown in Table 7.

Table 7 - The registers map of the OpenDSP General Purpose I/O ver. 1.1 original PLD software.

Location	R/W	Description
BASE + 0x00	R/W	When read returns the ADC(0) value, when written sets the DAC(0) value
BASE + 0x01	R/W	When read returns the ADC(1) value, when written sets the DAC(1) value
BASE + 0x02	R/W	When read returns the ADC(2) value, when written sets the DAC(2) value
BASE + 0x03	R/W	When read returns the ADC(3) value, when written sets the DAC(3) value
BASE + 0x04	R/W	When read returns the ADC(4) value, when written sets the DAC(4) value
BASE + 0x05	R/W	When read returns the ADC(5) value, when written sets the DAC(5) value
BASE + 0x06	R/W	When read returns the ADC(6) value, when written sets the DAC(6) value
BASE + 0x07	R/W	When read returns the ADC(7) value, when written sets the DAC(7) value
BASE + 0x08	R/W	When read returns the value of the DIGIN1 Bit15-Bit0, when written sets the value of the DIGOUT1 Bit15-Bit0
BASE + 0x09	R/W	When read returns the value of the DIGIN1 Bit31-Bit16, when written sets the value of the DIGOUT1 Bit31-Bit16
BASE + 0x0A	R/W	ADC Enable Register



Table 7 – Cont.

Location	R/W	Description
BASE + 0x0B	R/W	ADC Configuration Register
BASE + 0x0C	R/W	SOC/EOC Command Register

In order to minimize the work to interface the board with the existing system and focalize the design only on new features, a designer needs to know how many blocks could be connected together both in terms of number of logic gates and number of PLD pins used. Table 8 shows for each block the number of logic cells (LCs) and the number of PLD pins used. Other functional blocks can use some PLD pins also; the number of PLD pins used by other block is in brackets in Table 8.

Table 8 - Number of logic cells and number of PLD pins used by each functional block in the PLD software. The number of PLD pins used also by other functional blocks is in brackets.

Block name	LCs used	PLD pins used
DAC SOC Control Unit	8	34 (34)
Bus Arbiter	18	26 (13)
ADC SOC Control Unit	7	34 (34)
ADC Control Unit	311	15 (2)
EOC Control Unit	19	1 (1)
OpenDSP Bus Write Manager	77	29 (29)
OpenDSP Bus Read Manager	314	28 (12)

OpenDSP General Purpose I/O Board ver. 1.1 software architecture changes

In order to accomplish the technical specifications, some changes must be made to the PLD software Control Unit block. These changes concern:

- the digital I/O mapping;
- the circuits to represent the encoders;
- the registers used to represent the I/O board devices on the OpenDSP Bus.

All these changes can be grouped into a unique block named Encoder Controller and require modifying the OpenDSP Read Manager, the OpenDSP Write Manager blocks and the main level of the PLD software. The main level of the PLD software must be modified because the digital I/O mapping requires changing the type of the output stages and the PLD software designer allows this operation only at this level.

Figure 15 shows as the new Encoder Controller block is inserted in the existing architecture of the Control Unit.

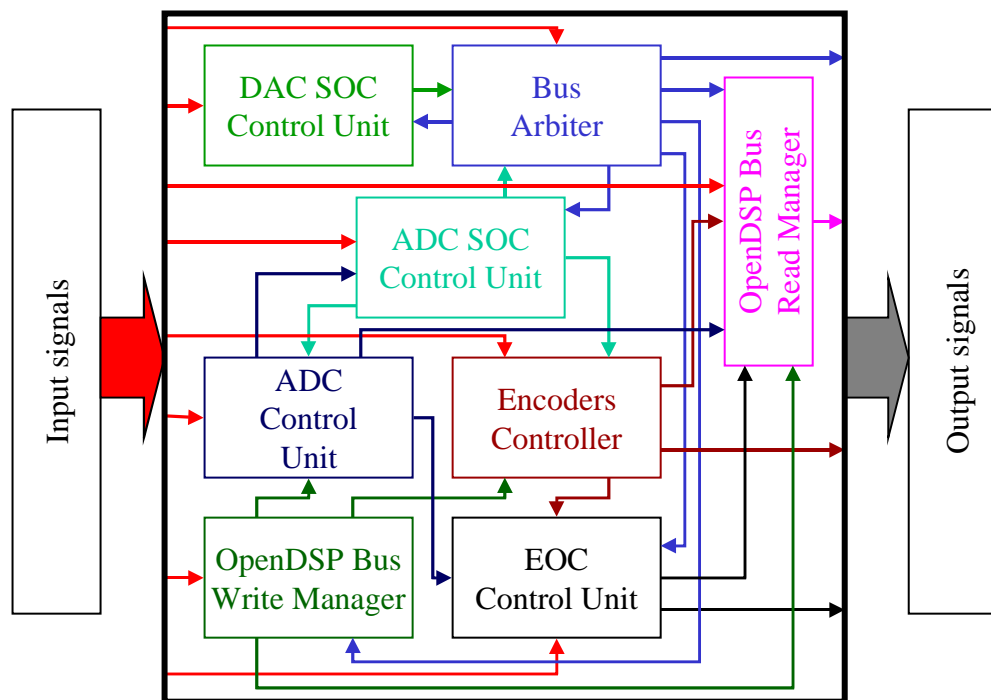


Figure 15 - The Control Unit with the changes required for the application object of these Application Notes.

The changes listed can be justified by these considerations:

- **the digital I/O mapping** each digital input and output line has a new meaning, as described in “DIGIN1 cable” and “Implementation by using the OpenDSP System”, then the PLD internal signals must change their names; moreover, in order to guarantee the drive capability of the enable digital input and enable encoder lines, the PLD output stages must be changed in open collector; also the digital inputs relative to the 0-24V Digital Input Interfaces must be complemented because these signals are generated as described in “Digital Input Interfaces”.
- **the circuits to represent the encoders** the encoder is used to measure the angular position of a shaft; when the encoder is incremental, it is necessary to count the edges of the encoder signals from a known time. This can be implemented by using a counter that can be reset (to get a reference point) and that is incremented or decremented according to the rotation sense.
- **the registers** each axis must contains:
 - ❖ a register for the counter that represents the encoder position;
 - ❖ a register for the latched value of the counter;
 - ❖ a register to read the value of the 0-24V Digital Inputs and to set the value of the 0-24V Digital Outputs;
 - ❖ a register to enable the Digital Input Interfaces and the Differential Encoder Interfaces;
 - ❖ a register to force via software a latch command and a zero command of the encoder counter.



The new registers map is in Table 9.

Table 9 - The registers map of the OpenDSP General Purpose I/O ver. 1.1 PLD software for this application.

Location	R/W	Description
BASE + 0x00	R/W	When read returns the ADC(0) value, when written sets the DAC(0) value that represents the Axis0 V_{REF}
BASE + 0x01	R/W	When read returns the ADC(1) value, when written sets the DAC(1) value that represents the Axis1 V_{REF}
BASE + 0x02	R/W	When read returns the ADC(2) value, when written sets the DAC(2) value that represents the Axis2 V_{REF}
BASE + 0x03	R/W	When read returns the ADC(3) value, when written sets the DAC(3) value
BASE + 0x04	R/W	When read returns the ADC(4) value, when written sets the DAC(4) value
BASE + 0x05	R/W	When read returns the ADC(5) value, when written sets the DAC(5) value
BASE + 0x06	R/W	When read returns the ADC(6) value, when written sets the DAC(6) value
BASE + 0x07	R/W	When read returns the ADC(7) value, when written sets the DAC(7) value
BASE + 0x08	-	Reserved
BASE + 0x09	-	Reserved
BASE + 0x0A	R/W	ADC Enable Register
BASE + 0x0B	R/W	ADC Configuration Register
BASE + 0x0C	R/W	SOC/EOC Command Register
BASE + 0x0D	-	Reserved
BASE + 0x0E	R/W	Axes Configuration Register (default value 0x00)
BASE + 0x0F	R/W	Encoders Command Register (default value 0x00)
BASE + 0x10	R	Axis0 Encoder Counter Current Value Register
BASE + 0x11	R	Axis1 Encoder Counter Current Value Register
BASE + 0x12	R	Axis2 Encoder Counter Current Value Register
BASE + 0x13	-	Reserved
BASE + 0x14	R	Axis0 Encoder Counter Latched Value Register
BASE + 0x15	R	Axis1 Encoder Counter Latched Value Register
BASE + 0x16	R	Axis2 Encoder Counter Latched Value Register
BASE + 0x17	-	Reserved



Table 9 – Cont.

Location	R/W	Description
BASE + 0x18	R/W	Axis0 Digital I/O Register (default value 0x00)
BASE + 0x19	R/W	Axis1 Digital I/O Register (default value 0x00)
BASE + 0x1A	R/W	Axis2 Digital I/O Register (default value 0x00)
BASE + 0x1B	-	Reserved
BASE + 0x1C ⋮ BASE + 0x1F	-	Reserved

Having 32 registers mapped, the DIP_SWITCH[0] is not used. This DIP-SWITCH allows setting the Bit5 base address of the OpenDSP General Purpose I/O Board ver. 1.1, that represents odd addresses.

a)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	-	EDI2	EDI1	EDI0	-	EE2	EE1	EE0

b)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	-	-	L2	Z2	L1	Z1	L0	Z0

c)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	-	OutAUX1	OutAUX0	Start	InAUX2	InAUX1	Reset	FC

Figure 16 - a) The Axes Configuration Register bit meanings. b) The Encoders Command Register bit meanings. c) The Axis0, Axis1 or Axis2 Digital I/O Register bit meanings.

The meanings of the Axes Configuration Register Bits are shown in Figure 16a. Set the EE0 bit to enable the Axis0 Differential Encoder Interface, EE1 to enable the Axis1 Differential Encoder Interface, EE2 to enable the Axis2 Differential Encoder Interface. Similarly, set the EDI0 bit to enable the Axis0 Digital Input Interface, EDI1 to enable the Axis1 Digital Input Interface, EDI2 to enable the Axis2 Digital Input Interface. Reset the bits to disable the relative interfaces.



Figure 16b shows the meanings of the Encoders Command Register Bits. To force the reset of the counter that represents the Axis0, Axis1 or Axis2 encoder it is necessary to set and then reset the Z0, Z1 or Z2 bit respectively. To force the latching of the counter that represents the Axis0, Axis1 or Axis2 encoder it is necessary to set and then reset the L0, L1 or L2 bit respectively.

Figure 16c shows the meanings of the Axis0, Axis1 or Axis2 Digital I/O Register Bits. Reading this register, the most significant nibble is set to zero. Writing this register, the least significant nibble is ignored.

The Encoder Controller

This block makes four functions:

- changes the assignment from internal registers to the DIGOUT1 connector;
- changes the assignment from the DIGIN1 connector to the internal registers;
- implements the counters to represent the encoders;
- generates the EOC signal according to the interface specifications of the EOC Control Unit block.

The assignments are simply realized by changing the name of the signals.

The EOC signal is a pulse of duration 125 nsec, generated after 250 nsec from the edge of the SOC; the SOC signal is the ADC SOC.

The signals of a bidirectional encoder are connected to an Encoder Unit block; more details about this block are reported in “The Encoder Unit”. Notice that the latch command is generated from the logic OR of the software command signal and the external digital signal, as required in “Technical specifications”. The relative bit in the Axes Configuration Register controls the encoder counter by means of the ENC_ENABLE input (see “The Encoder Unit” for more details about the interface of the Encoder Unit block).

The Encoder Unit

This block implements a counter that allows counting the edges of the A and B encoder signals in order to provide information about the rotation of the encoder shaft.

This block provides the counter value in a 16-bits output called ENC_CURRENT_VALUE_REGISTER. It also provides the latched value in a 16-bits output named ENC_LATCH_VALUE; the counter value is latched if a positive edge of the ENC_LATCH_VALUE input occurs. The counter value can be reset if a positive edge of the ENC_ZERO input occurs.

The encoder counter can be enabled or disabled according to the value of the ENC_ENABLE input. Set the input to enable the encoder counter, reset it to disable the encoder counter.

The Encoder Unit block performs two main jobs. The first job, activated by a positive edge of the clock, decides if the counter value must be incremented or decremented, according to the sequence of the A and B encoder signals. The second job, activated by a negative edge of the clock, increments or decrements the encoder counter, verify if a zero or latch commands are occurred and performs the related operations. In both cases, if the board reset is asserted, the circuitry is reset.

The first job is realized by mean of a finite state machine (FSM). Figure 17a shows the A and B encoder signals and the relate states of the FSM. Figure 17b shows the FSM; it is a Moore FSM. The label on the oriented arcs indicates the combination of the encoder signals and, after the comma, the sense of the count.

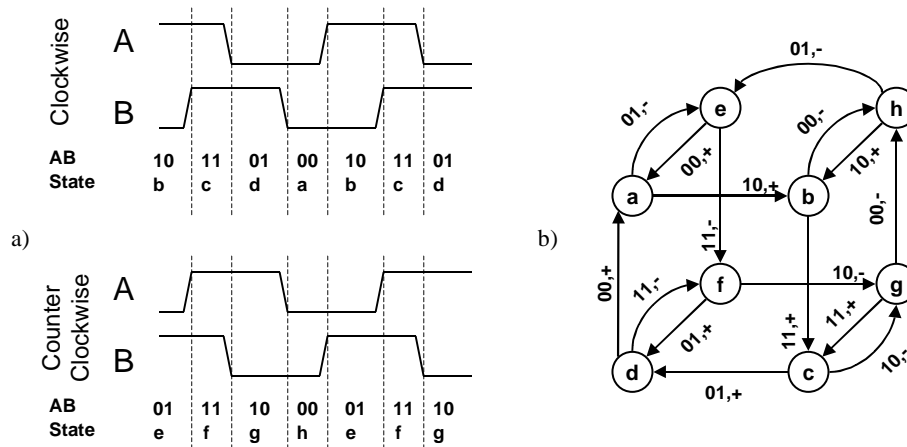


Figure 17 - a) The bidirectional encoder signals and the relate states of the finite state machine.
b) The finite state machine that implements the encoder counter.

The inputs are evaluated at each positive edge of the clock. If the inputs are not changed, the state does not change and the counter is neither incremented nor decremented (in Figure 17b this condition is not depicted). If the inputs change, the FSM goes in the new state and the counter is incremented (+ in Figure 17b) or decremented (- in Figure 17b).

“Appendix B – The code of the Encoder Unit block” reports the source code of this block; it is written in VHDL.

DSP software

Some changes are needed in the MatDSP ver. 3.2 in order to simplify the use of the new hardware. However, these changes allow using all the software written for the MatDSP ver. 3.2; remember that the first three analog outputs now have $\pm 10V$ output range.

MatDSP in Command Mode does not change, thus refer [8] for more information. Otherwise, MatDSP in Programming Mode changes new constants, arrays and macros are provided.



The constants

A new constant is defined for this application, see Table 10. This constant is added to the constants defined for the OpenDSP General Purpose I/O ver. 1.1 (see [8] for more information).

Table 10 - Constants.

Constants	Description
IOBOARD_NO_ENCODER	This constant defines the number of encoders (and then axes) present on board.

The arrays

Some new arrays are defined for this application, see Table 11. These arrays are added to the variables and arrays defined for the OpenDSP General Purpose I/O ver. 1.1 (see [8] for more information).

Table 11 - Arrays.

Constants	Description
ENC_CURRENT_CH[IOBOARD_NO_ENCODER]	This array contains the encoder counters current values; it can be update by calling the <code>read_current_value_enc()</code> macro.
ENC_LATCHED_CH[IOBOARD_NO_ENCODER]	This array contains the encoder counters latched values; it can be update by calling the <code>read_latched_value_enc()</code> macro.
ENC_DIGITAL_INPUT[IOBOARD_NO_ENCODER]	This array contains the Digital Inputs values; it can be update by calling the <code>read_digital_input()</code> macro.

The macros

Some new macros are defined for this application. These macros are added to the macros defined for the OpenDSP General Purpose I/O ver. 1.1 (see [8] for more information).

The macros can be grouped in five sets:

- macros to configure the axes Differential Encoder Interfaces and the Digital Input Interfaces, listed in Table 12;
- macros to force commands on the encoder counters, listed in Table 13;
- macros to read the encoder counters current and latched values and to read the digital inputs, listed in Table 14;
- macros to set and reset the digital outputs, listed in Table 15;
- macros to detect the configuration of the axes hardware, listed in Table 16.



Each macro needs to specify only the axis that must be manipulated.

Table 12 - Macros to configure the axes Differential Encoder Interfaces and the Digital Input Interfaces.

Macro	Description
ENABLE_ENCODERS(ch) ¹	This macro allows enabling the Axis0, Axis1 or Axis2 Differential Encoder Interface.
DISABLE_ENCODERS(ch)	This macro allows disabling the Axis0, Axis1 or Axis2 Differential Encoder Interface.
ENABLE_DIAXIS(ch)	This macro allows enabling the Axis0, Axis1 or Axis2 Digital Input Interface.
DISABLE_DIAXIS(ch)	This macro allows disabling the Axis0, Axis1 or Axis2 Digital Input Interface.
SET_ENCODER_MODULE(ch, NumBits)	This macro sets the count module of the Axis0, Axis1 or Axis2 encoder counter; the module is a power of two and NumBits specifies the exponent. The counter module is set to 0xFFFF by default if the IOBoardInit() function is used.

Table 13 - Macros to force commands on the encoder counters.

Macro	Description
FORCE_ZERO_ENCODER(ch) ²	This macro generates a pulse of the Z0, Z1 or Z2 bit (see Figure 16b for more details), according to the selected axis.
FORCE_LATCH_ENCODER(ch)	This macro generates a pulse of the L0, L1 or L2 bit (see Figure 16b for more details), according to the selected axis.

¹ Possible values for the input arguments are:

ch = 0, 1, 2;

NumBits = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;

² See footnote number 1 at page 28.



Table 14 – Macros to read the encoder counters current and latched values and to read the digital inputs.

Macro	Description
read_current_value_enc(ch) ³	This macro copies the Axis0, Axis1 or Axis2 encoder counter current value into the corresponding element of the ENC_CURRENT_CH[] array.
read_latched_value_enc(ch)	This macro copies the Axis0, Axis1 or Axis2 encoder counter latched value into the corresponding element of the ENC_LATCHED_CH[] array.
read_digital_input(ch)	This macro copies the Axis0, Axis1 or Axis2 digital inputs value into the corresponding element of the ENC_DIGITAL_INPUT[] array.

Table 15 - Macros to set and reset the digital outputs.

Macro	Description
SET_START_BIT(ch) ⁴	This macro sets the Axis0, Axis1 or Axis2 start output bit, according to the selected axis.
SET_OUTAUX0_BIT(ch)	This macro sets the Axis0, Axis1 or Axis2 outAUX0 output bit, according to the selected axis.
SET_OUTAUX1_BIT(ch)	This macro sets the Axis0, Axis1 or Axis2 outAUX1 output bit, according to the selected axis.
RESET_START_BIT(ch)	This macro resets the Axis0, Axis1 or Axis2 start output bit, according to the selected axis.
RESET_OUTAUX0_BIT(ch)	This macro resets the Axis0, Axis1 or Axis2 outAUX0 output bit, according to the selected axis.
RESET_OUTAUX1_BIT(ch)	This macro resets the Axis0, Axis1 or Axis2 outAUX1 output bit, according to the selected axis.

Table 16 - Macros to detect the configuration of the axes hardware.

Macro	Description
IS_ENC_ENABLED(ch) ⁵	This macro returns 0x01 if the Axis0, Axis1 or Axis2 Differential Encoder Interface is enabled, 0x00 otherwise.

³ See footnote number 1 at page 28.

⁴ See footnote number 1 at page 28.

⁵ See footnote number 1 at page 28.



Table 16 – Cont.

Macro	Description
IS_DIAXIS_ENABLED(ch)	This macro returns 0x01 if the Axis0, Axis1 or Axis2 Digital Input Interface is enabled, 0x00 otherwise.

An example of use

In order to illustrate how it is possible to use the macros given with this application, an example of use is presented.

The program is named `encecho.c` and it is saved into the source directory of the MatDSP library. It has been obtained by typing `dspnew('encecho')` in the Matlab command window and editing the obtained template file. The entire program source is reported in “Appendix C – The source code of the MatDSP example”. In the next paragraphs, only the significant parts are presented and commented.

Description of the program

The program must use all macros for each axis, in order to test each functionality. The test must be controlled from the Matlab command window, by using the functions of the MatDSP toolbox.

The idea to use each macro consists to execute in the `user_ctrl()` routine three operations:

- read the encoder counter current values, the encoder counter latched value and set the axes analog outputs;
- check the value of a variable changeable from the Matlab workspace and perform the corresponding operation; this variable is named `fCommand`;
- reset the value of the `fCommand` variable, in order to detect a new command during the next `user_ctrl()` call.

Notice that all the variables, changeable from the Matlab workspace, are float. This is due to the `dspvar()` function of the MatDSP toolbox.

The user can be read and write the variables listed in Table 17.

Table 17 - Variables and arrays that can be read or written by the Matlab command window.

Variable or array	Description
<code>fEncCurrentValue[]</code>	This array contains the encoder counters current values
<code>fEncLatchedValue[]</code>	This array contains the encoder counters latched values
<code>fDigitalInput[]</code>	This array contains the digital inputs values
<code>fCommand</code>	This variable forces the program to execute a particular macro
<code>fAnswer</code>	This variable is set after an <code>IS_ENC_ENABLED()</code> or <code>IS_DIAXIS_ENABLED()</code> macro execution.



Table 17 – Cont.

Variable or array	Description
Array1[]	This array is used to set the value of the axes analog outputs.

Initialization

The source code executed during the `user_init()` is composed by three sections.

The first section initializes the encoder modules by using the `SET_ENCODER_MODULE()` macro:

```
/* Initialize the ADC on IOBOARD */  
IOBoardInit();  
SET_ENCODER_MODULE(0,12)  
...
```

The `IOBoardInit()` allows the initialization of the analog output gains and saturations as described in [8]. This function call assures also that the encoder modules are initialized to `0xFFFF`. In this example, the `Axis0` encoder module is set to 4095 because the exponential of power equal to 2 is set to 12. These operations must be performed always in order to set properly the encoder modules.

The second section enables the Differential Encoder Interfaces and the Digital Input Interfaces by using the `ENABLE_ENCODERS()` and `ENABLE_DIAXIS()` macros:

```
/* Configure the axis digital inputs and encoders */  
ENABLE_ENCODERS(0)  
...  
ENABLE_DIAXIS(0)  
...
```

The third section initializes the array used to pass to Matlab the Digital Inputs values and enables the interrupts generation.

Use of the macros to read the encoder counters values

From this point, all the operations are performed in the `user_ctrl()` routine.

The user can use the `read_current_value_enc()` and `read_latched_value_enc()` macros to read the encoder counters current values and the encoder counters latched values. In the example, these two macros are used as in a loop over all the axes:

```
/* Read all encoders */  
for(i=0; i<IOBOARD_NO_ENCODER; i++)  
{  
    read_current_value_enc(i);  
}
```



```
fEncCurrentValue[i] = (float)ENC_CURRENT_CH[i];  
read_latched_value_enc(i);  
fEncLatchedValue[i] = (float)ENC_LATCHED_CH[i];  
  
...  
}
```

As described in Table 14, the `read_current_value_enc(i)` stores the encoder counter current value in the `ENC_CURRENT_CH[i]` element. This can be passed to the Matlab workspace simply assigning the value to a float variable. Same steps can be performed for the latched value.

Use of the macros to force commands to the encoder counters

The user can use the `FORCE_ZERO_ENCODER()` and the `FORCE_LATCH_ENCODER()` macros to force the reset of the encoder counters or to force the encoder counters latching. In the following, the Axis0 encoder counter is reset and then the same counter is latched.

```
case 1: FORCE_ZERO_ENCODER(0);  
    break;  
...  
case 4: FORCE_LATCH_ENCODER(0);  
    break;  
...
```

Use of the macros to control the digital outputs

The user can use the `SET_START_BIT()`, the `RESET_START_BIT()`, the `SET_OUTAUX0_BIT()`, the `RESET_OUTAUX0_BIT()`, the `SET_OUTAUX1_BIT()` and the `RESET_OUTAUX1_BIT()` macros to set and reset each Digital Outputs. In the following, the Axis0 Digital Outputs are set and reset.

```
case 7: SET_START_BIT(0);  
    break;  
case 8: RESET_START_BIT(0);  
    break;  
...  
case 13: SET_OUTAUX0_BIT(0);  
    break;  
case 14: RESET_OUTAUX0_BIT(0);  
    break;  
...  
case 19: SET_OUTAUX1_BIT(0);  
    break;  
case 20: RESET_OUTAUX1_BIT(0);  
    break;  
...
```




Use of the macros to read the digital inputs values

The user can use the `read_digital_input()` macro to read the axes Digital Inputs value. In the example, this macro is used to read the Axis0 Digital Inputs value:

```
case 25: read_digital_input(0);
        fDigitalInput[0] = (float)ENC_DIGITAL_INPUT[0];
        break;
...
```

As described in Table 14, the `read_digital_input(i)` stores the Digital Inputs value in the `ENC_DIGITAL_INPUT[i]` element. This can be passed to the Matlab workspace simply assigning the value to a float variable.

Use of the macros to enable and disable the Differential Encoder Interfaces and the Digital Input Interfaces

The user can use the `ENABLE_ENCODERS()`, the `DISABLE_ENCODERS()`, the `ENABLE_DIAXIS()` and the `DISABLE_DIAXIS()` macros to enable and disable the Differential Encoder Interfaces and the Digital Input Interfaces. In the following, the Axis0 Differential Encoder Interface and the Digital Input Interfaces are enabled and disabled.

```
case 28: ENABLE_ENCODERS(0);
        break;
...
case 31: DISABLE_ENCODERS(0);
        break;
...
case 34: ENABLE_DIAXIS(0);
        break;
...
case 37: DISABLE_DIAXIS(0);
        break;
...
```

Use of the macros to know the state of the Differential Encoder Interfaces and the Digital Input Interfaces

The user can use the `IS_ENC_ENABLED()` and the `IS_DIAXIS_ENABLED()` macros to know the state of the Differential Encoder Interfaces and the Digital Input Interfaces. In the following, if the Axis0 Differential Encoder Interface is enabled the `fAnswer` variable is set to 0.1; if the Digital Input Interfaces is enabled the `fAnswer` variable is set to 0.4.

```
case 40: if (IS_ENC_ENABLED(0) == 1) fAnswer = 0.1;
        break;
...
```



```
case 43: if (IS_DIAxis_ENABLED(0) == 1) fAnswer = 0.4;
        break;
...

```

Use the test program from the Matlab command window

Before to use the program, it is necessary to load and start the program executable in the OpenDSP System. This can be done by typing in the Matlab command window:

- `dspload('encecho')` this function is a MatDSP toolbox function that loads a program in the OpenDSP System;
- `dspstart` this function is a MatDSP toolbox function that starts a program previous loaded in the OpenDSP System.

Now it is possible to interact with the program in execution on the OpenDSP System.

For example, if the user want disable the Axis1 Differential Encoder Interface, he must type in the Matlab command window:

```
dspvar('fCommand',32)
```

If the user want read the Axis2 Encoder counter current value, he must type in the Matlab command window:

```
dspvar('fEncCurrentValue')
```

Notice that it is possible to use the `dspscope` to monitor this variable.

If the user want latch the Axis1 Encoder counter and read its value, he must type in the Matlab command window:

```
dspvar('fCommand',5)
dspvar('fEncLatchedValue')
```

If the user want reset the Axis0 Encoder counter, he must type in the Matlab command window:

```
dspvar('fCommand',1)
```

If the user want set the Axis2 Start output, he must type in the Matlab command window:

```
dspvar('fCommand',12)
```

If the user want read the Axis0 Digital Input value, he must type in Matlab command window:



```
dspvar('fCommand',25)  
dspvar('fDigitalInput')
```

If the user want know the state of the Axis1 Digital Input Interfaces, he must type in the Matlab command window:

```
dspvar('fAnswer',0)  
dspvar('fCommand',44)  
dspvar('fAnswer')
```

Appendix A – The board layouts

In this section the board layouts are presented. In each figure the sizes are in millimeters; in square brackets sizes in mils are reported.

Figure 18 shows the Axis0 Interface Board layout with the board sizes and the drills position. Figure 19 shows the prototype realized and highlights functional blocks.

Figure 20 shows the Axis 1 Interface Board layout with the board sizes and the drills position. Figure 21 shows the prototype realized and highlights functional blocks.

Figure 22 shows the Axis 2 Interface Board layout with the board sizes and the drills position. Figure 23 shows the prototype realized and highlights functional blocks.

Figure 24 shows the Analog Interface Board layout with the board sizes and the drills position. Figure 25a shows the component side of the prototype realized, whereas the Figure 25b shows the solder side.

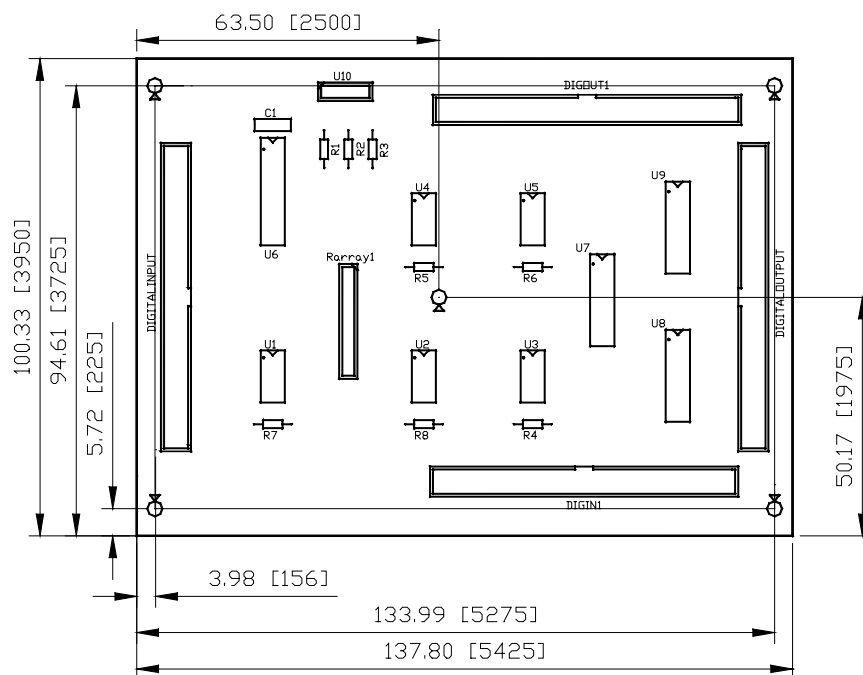


Figure 18 - Axis0 Interface Board layout. The measures are in millimeters (mils in brackets).

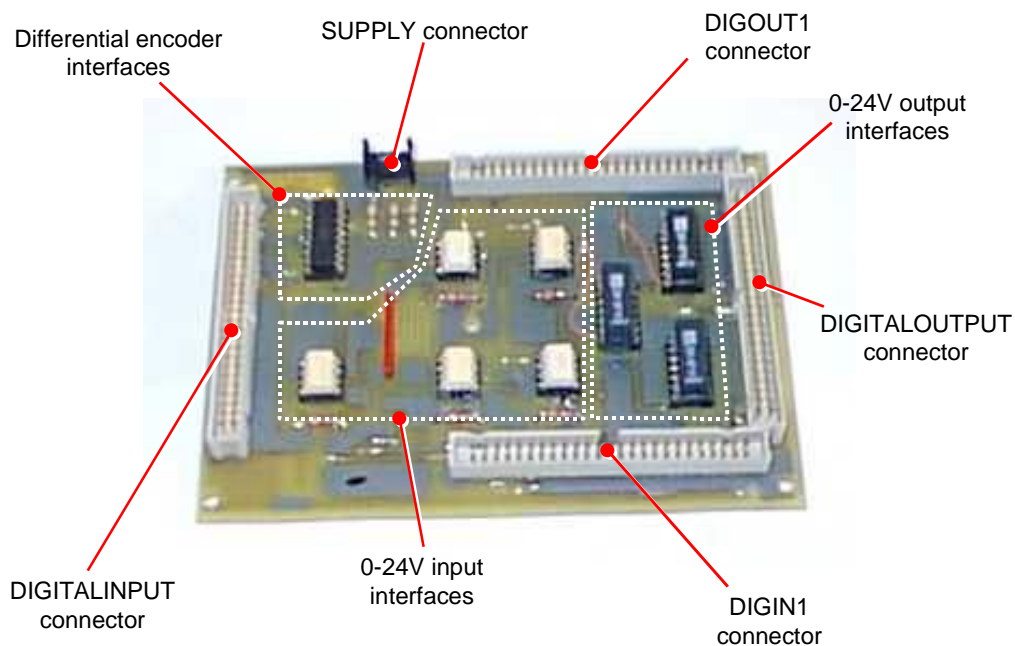


Figure 19 - The Axis0 Interfaces Board component side of the prototype realized for this application.

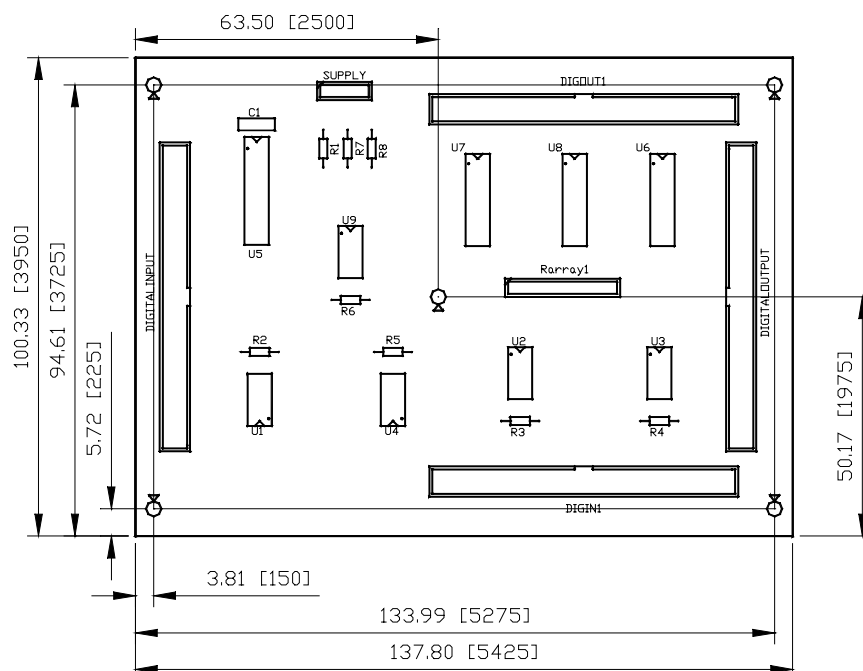


Figure 20 - Axis1 Interface Board layout. The measures are in millimeters (mils in brackets).

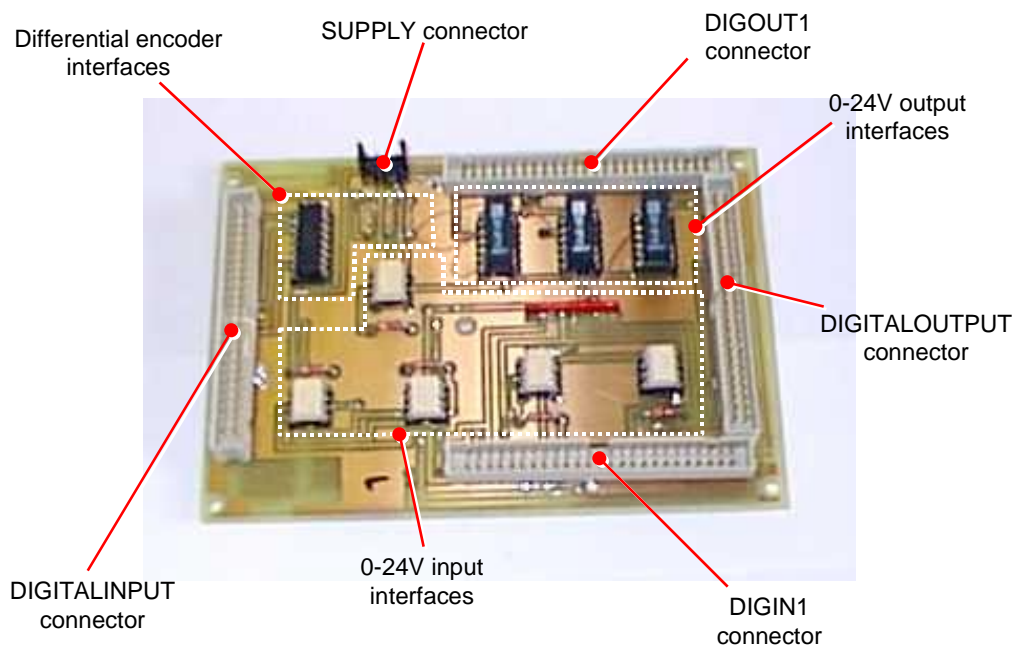


Figure 21 - The Axis1 Interfaces Board component side of the prototype realized for this application.

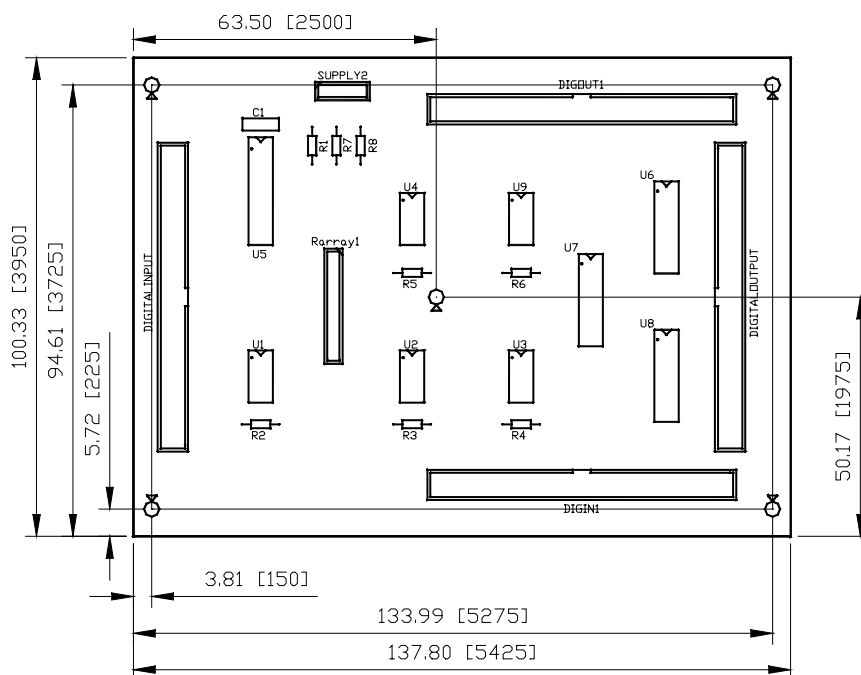


Figure 22 - Axis2 Interface Board layout. The measures are in millimeters (mils in brackets).

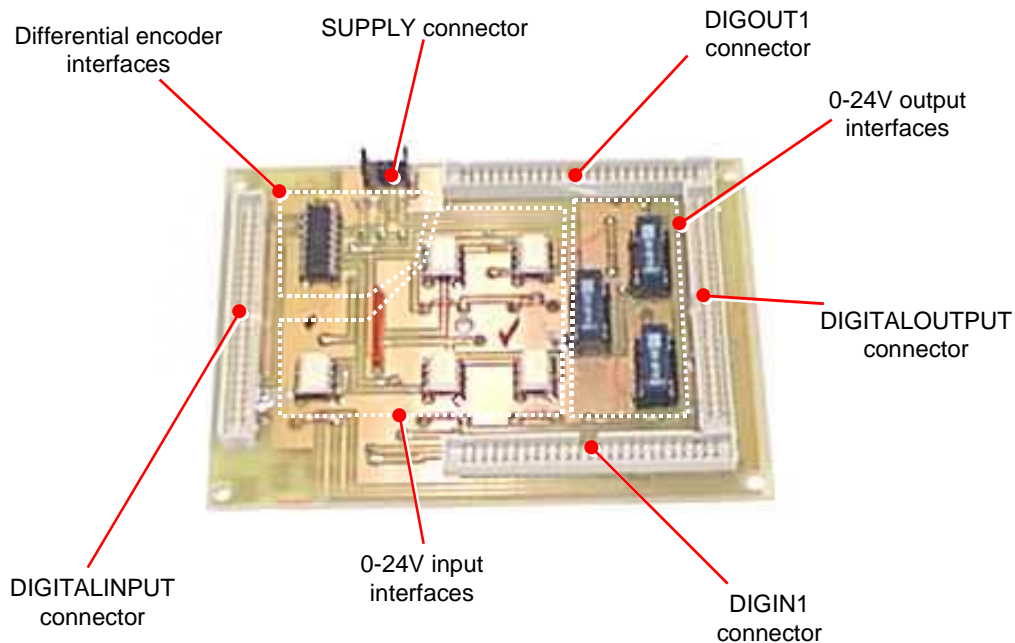


Figure 23 - The Axis2 Interfaces Board component side of the prototype realized for this application.

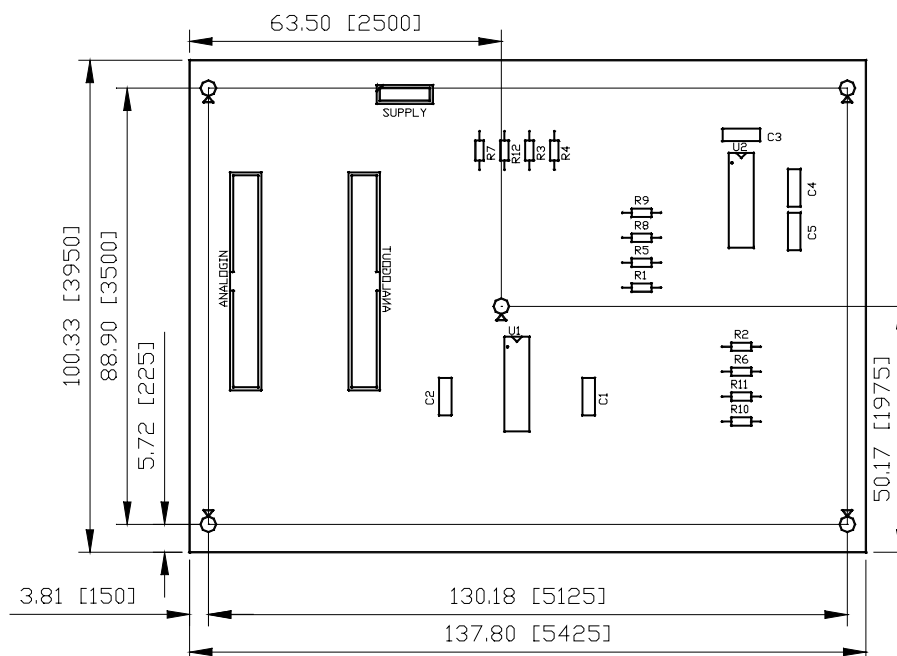


Figure 24 - Analog Interface Board layout. The measures are in millimeters (mils in brackets).

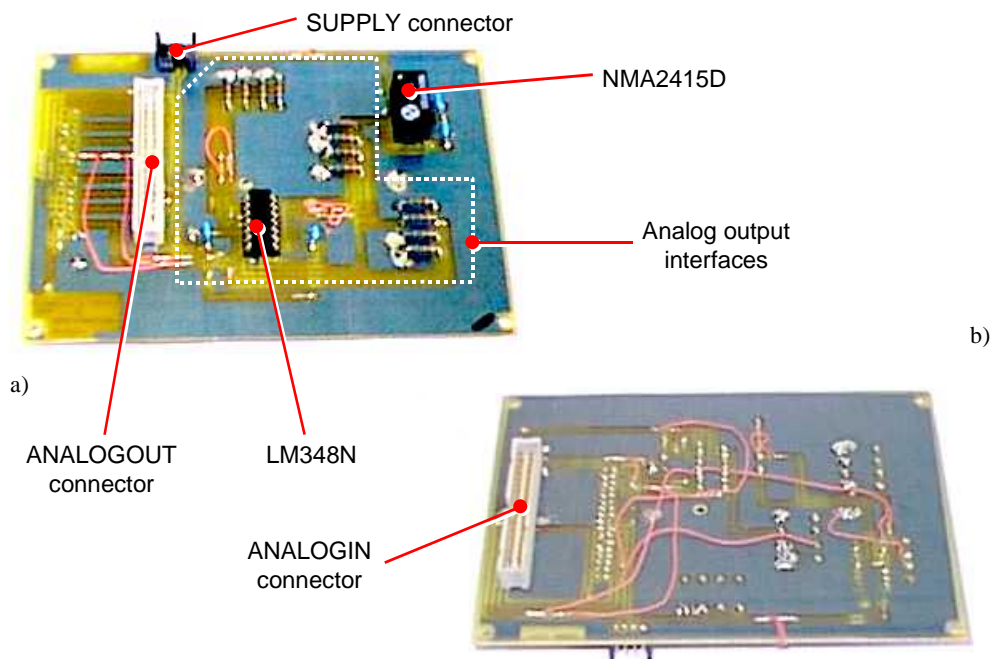


Figure 25 - a) The Analog Interfaces Board component side of the prototype realized for this application. b) The Analog Interfaces Board solder side of the prototype realized for this application.



Appendix B – The code of the Encoder Unit block

```
-- *****
-- Politecnico di Torino - CSPP - LIM
-- *****
-- EncoderUnit.vhd - Basic Unit to interface an encoder.
-- *****
-- Library block to control a single encoder. This block allows to control the 16-bit
-- counter, that represents the encoder, by using the A,B and Z typical lines. The counter
-- is increased or decreased of one unit at each edge of the A or B lines, according to the
-- time sequence depicted in the following schemes:
--
--      The axis turns in clockwise sense -> the counter
--      must be increased
--
--      A  _____|_____|_____|_____|_____|_____|_____
--      B  _____|_____|_____|_____|_____|_____|_____
--
--      The axis turns in counter clockwise sense -> the counter
--      must be decreased
--
--      A  _____|_____|_____|_____|_____|_____|_____
--      B  _____|_____|_____|_____|_____|_____|_____
--
-- Other features are:
--
-- + possibility to disable the encoder machine;
-- + possibility to reset the counter if the Z lines or an internal signal (i.e. control-
--   led by a register) has a positive edge.
--
-- *****
-- rev. 1.0 - 28/10/1998 Andrea Delmastro
--
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY EncoderUnit IS
    PORT
    (
        -- Board signals
        CLK                                     : IN      STD_LOGIC;
        -- General Encoder Input signals
        ENC_A_INPUT                           : IN      STD_LOGIC;
        ENC_B_INPUT                           : IN      STD_LOGIC;
        ENC_Z_INPUT                           : IN      STD_LOGIC;
        -- Encoder Registers
        ENC_CURRENT_VALUE_REGISTER            : INOUT   STD_LOGIC_VECTOR(15 downto 0); -- Current
value of the internal counter
        ENC_LATCH_VALUE                       : OUT     STD_LOGIC_VECTOR(15 downto 0);   --
Internal counter latched value
        -- Open-DSP Bus signals
        ODSPB_nRESET                         : IN      STD_LOGIC;
        -- Main reset (low active)
        -- Encoder Unit control signals
        ENC_ENABLE                            : IN      STD_LOGIC;
        ENC_LATCH                             : IN      STD_LOGIC;
        -- If '0' the counter increments machine is disabled
        -- A positive edge latches the internal counter value
        -- into the latch register
        ENC_ZERO                             : IN      STD_LOGIC;
        -- A positive edge force the initialization of the counter
        -- to 0x0000
    );
END EncoderUnit;

ARCHITECTURE EncoderArchitecture OF EncoderUnit IS
    SIGNAL ENC_ZERO_OR_Z_INPUT              : STD_LOGIC;
    -- Logic OR of the ENC_ZERO and ENC_Z_INPUT signals
```



Center for Prototyping Services

Mechatronics Laboratory



```
SIGNAL PREVIOUS_ZERO : STD_LOGIC;
-- Memory of the previous value of the ENC_ZERO_OR_Z_INPUT
SIGNAL PREVIOUS_LATCH : STD_LOGIC;
-- Memory of the previous value of the ENC_LATCH line
SIGNAL NOT_CLK : STD_LOGIC;
-- Inversion of the clock
SIGNAL INCREMENT : STD_LOGIC;
-- A positive edge causes the increment of the internal counter
SIGNAL DECREMENT : STD_LOGIC;
-- A positive edge causes the decrement of the internal counter
SIGNAL STATE_COUNTER : STD_LOGIC_VECTOR(2 downto 0);
Counter of the finite state machine

BEGIN
-- Build the internal signals: the zero command and the clock source for the
-- finite state machine.
ENC_ZERO_OR_Z_INPUT <= ENC_ZERO OR (ENC_Z_INPUT AND ENC_ENABLE);
-- Invert the clock to synchronize the two processes.
NOT_CLK <= NOT CLK;

PROCESS (CLK, ODSPB_nRESET)
BEGIN
IF (ODSPB_nRESET = '0') THEN
-- Set the state machine counter
STATE_COUNTER <= "000";
-- set the counter controllers
INCREMENT <= '0';
DECREMENT <= '0';
ELSIF (CLK'EVENT AND CLK = '1') THEN
-- Manage the encoder A and B signals
IF (ENC_ENABLE = '1') THEN
-- If the encoder is enabled, it's possible to check
-- the transitions of the typical encoder signals, in
-- order to control the current value of the internal
-- counter. This is implemented by means of a Finite
-- State Machine depicted into the documentation.
--
-- REMEMBER THAT:
-- The ENC_A_INPUT and ENC_B_INPUT follows this rules:
--
-- The axis turns in clockwise sense -> the counter
-- must be increased
--
-- A
-- |_____|_____|_____|_____|_____|_____|_____|_____|
-- B
-- |_____|_____|_____|_____|_____|_____|_____|_____|
--
-- The axis turns in counter clockwise sense -> the counter
-- must be decreased
--
-- A
-- |_____|_____|_____|_____|_____|_____|_____|_____|
-- B
-- |_____|_____|_____|_____|_____|_____|_____|_____|
--
-- The internal counter is controlled by means of a finite state ma-
-- chine (FSM) that controls two internal lines: INCREMENT and DECREMENT.
-- In the second process (see the end of this file), the internal
-- counter is incremented or decremented according to the value of
-- these lines. Note that these lines are reset in this process
-- if the encoder is disabled or it's impossible to recognise a
-- correct transition of ENC_A_INPUT and ENC_B_INPUT for the particu-
-- lar state in which the FSM is stopped.
--
-- BEGIN OF THE FINITE STATE MACHINE IMPLEMENTATION

IF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "000") THEN
-- From A to B with one internal counter increment
INCREMENT <= '1';
DECREMENT <= '0';
STATE_COUNTER <= "001";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "001") THEN
-- From B to C with one internal counter increment
INCREMENT <= '1';
DECREMENT <= '0';
STATE_COUNTER <= "011";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "011") THEN
-- From C to D with one internal counter increment
INCREMENT <= '1';
DECREMENT <= '0';
STATE_COUNTER <= "010";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "010") THEN
-- From D to A with one internal counter increment
INCREMENT <= '1';
DECREMENT <= '0';
STATE_COUNTER <= "000";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "000") THEN
```



Center for Prototyping Services

Mechatronics Laboratory



```
-- From A to E with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "100";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "100") THEN
-- From E to F with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "110";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "110") THEN
-- From F to G with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "111";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "111") THEN
-- From G to H with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "101";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "101") THEN
-- From H to E with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "100";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "101") THEN
-- From H to B with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "001";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "111") THEN
-- From G to C with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "011";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "110") THEN
-- From F to D with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "010";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "100") THEN
-- From E to A with one internal counter increment
INCREMENT <= '1';
DECREMENT <= '0';
STATE_COUNTER <= "000";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "011") THEN
-- From C to G with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "111";
ELSIF (ENC_A_INPUT = '0' AND ENC_B_INPUT = '0' AND STATE_COUNTER = "001") THEN
-- From B to H with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "101";
ELSIF (ENC_A_INPUT = '1' AND ENC_B_INPUT = '1' AND STATE_COUNTER = "010") THEN
-- From D to F with one internal counter decrement
INCREMENT <= '0';
DECREMENT <= '1';
STATE_COUNTER <= "110";
ELSE
INCREMENT <= '0';
DECREMENT <= '0';
END IF;

-- END OF THE FINITE STATE MACHINE IMPLEMENTATION

ELSE
INCREMENT <= '0';
DECREMENT <= '0';
END IF;
END IF;
END PROCESS;

-- This process is clocked by the inverted clock and it's purpose is update the
-- latch and zero PREVIOUS signals and control the internal counter.
PROCESS (NOT_CLK, ODSPB_nRESET)
BEGIN
IF (ODSPB_nRESET = '0') THEN
-- At the reset, all the registers are initialized
PREVIOUS_ZERO <= '0';
PREVIOUS_LATCH <= '0';
ENC_CURRENT_VALUE_REGISTER <= "0000000000000000";
ELSIF (NOT_CLK'EVENT AND NOT_CLK = '1') THEN
-- At each clock stroke, it updates the memory of the ZERO command
IF (ENC_ZERO_OR_Z_INPUT = '0') THEN
PREVIOUS_ZERO <= ENC_ZERO_OR_Z_INPUT;
END IF;
END IF;
```



Center for Prototyping Services

Mechatronics Laboratory



```
-- Manage the reset command
IF (ENC_ZERO_OR_Z_INPUT = '1' AND PREVIOUS_ZERO = '0') THEN
    -- If the reset command occurs, the internal counter must be reset.
    ENC_CURRENT_VALUE_REGISTER <= "0000000000000000";
    PREVIOUS_ZERO <= '1';
ELSIF (ENC_Z_INPUT = '1' AND ENC_ENABLE = '0') THEN
    -- If the encoder is disabled and the encoder zero input is high, it's
    -- necessary to delete the reset request.
    PREVIOUS_ZERO <= '1';
ELSE
    -- If no reset command occurred, it's possible to manage the internal counter.
    -- Manage the internal counter to increment its value.
    IF (INCREMENT = '1') THEN
        ENC_CURRENT_VALUE_REGISTER <= UNSIGNED(ENC_CURRENT_VALUE_REGISTER) + 1;
    END IF;
    -- Manage the internal counter to decrease its value.
    IF (DECREMENT = '1') THEN
        ENC_CURRENT_VALUE_REGISTER <= UNSIGNED(ENC_CURRENT_VALUE_REGISTER) - 1;
    END IF;
END IF;
-- At each clock stroke, it updates the memory of the LATCH command
-- and eventually of the internal counter latch value
IF (ENC_LATCH = '0') THEN
    PREVIOUS_LATCH <= ENC_LATCH;
END IF;
IF (ENC_LATCH = '1' AND PREVIOUS_LATCH = '0') THEN
    ENC_LATCH_VALUE <= ENC_CURRENT_VALUE_REGISTER;
    PREVIOUS_LATCH <= '1';
END IF;
END IF;
END PROCESS;
END EncoderArchitecture;
```



Appendix C – The source code of the MatDSP example

```
/*
This template file was created by MatDSP v.3.2
*/
#include "matdsp.h"

/*
  Authors:          Andrea Argondizza, Andrea Delmastro
  Date:            24/05/98
  Last editing:    16/12/98
  Purpose:         Copy the three encoders values
*/

/* # DO NOT EDIT THIS LINE */
/***** Place here below your own INCLUDE and DEFINE statements *****/

/*----- End of #include/#define section -----*/

/***** Place here below your own GLOBAL VARIABLES *****/

float  fEncCurrentValue[IOBOARD_NO_ENCODER];
float  fEncLatchedValue[IOBOARD_NO_ENCODER];
float  fDigitalInput[IOBOARD_NO_ENCODER];
float  fCommand = 0, fAnswer = 0;

/*----- End of global variables section -----*/

/***** User's INITIALIZATION ROUTINE *****/
void user_init(void)
{
    int i;

    SET_TASK_ID(11);          /* User encoder echo */
    SET_SYSTEM;
    /* Initialize the ADC on IOBOARD */
    IOBoardInit();
    IO1_AXIS_CONF_REG = 0x00;
    SET_ENCODER_MODULE(0,12)
    SET_ENCODER_MODULE(1,12)
    SET_ENCODER_MODULE(2,12)

    /* Configure the axis digital inputs and encoders */
    ENABLE_ENCODERS(0)
    ENABLE_ENCODERS(1)
    ENABLE_ENCODERS(2)
    ENABLE_DIAXIS(0)
    ENABLE_DIAXIS(1)
    ENABLE_DIAXIS(2)

    /* Initialize the fDigitalInput array */
    for(i=0;i<IOBOARD_NO_ENCODER;i++)
        fDigitalInput[i] = 0.0;

    /* Enable the INT2 and INT3 generation */
    EnableINT23 = ENABLE_ODSPB_INT23;
}
/*----- End of user_init() -----*/

/***** User's INTERRUPT-CALLED ROUTINE (at any sampling time) *****/
void user_ctrl(void)
{
    int i;

    /* Read all encoders */
    for(i=0;i<IOBOARD_NO_ENCODER;i++)
```



```
{
    read_current_value_enc(i);
    fEncCurrentValue[i] = (float)ENC_CURRENT_CH[i];
    read_latched_value_enc(i);
    fEncLatchedValue[i] = (float)ENC_LATCHED_CH[i];

    /* Update DAC outputs */
    OUT_CH[i] = array1[i];
    write_da(i);
}

/* Each macro can be tested according to the value written into fCommand */
/* by the host.
switch ((int)fCommand)
{
    case 1: FORCE_ZERO_ENCODER(0);
            break;
    case 2: FORCE_ZERO_ENCODER(1);
            break;
    case 3: FORCE_ZERO_ENCODER(2);
            break;
    case 4: FORCE_LATCH_ENCODER(0);
            break;
    case 5: FORCE_LATCH_ENCODER(1);
            break;
    case 6: FORCE_LATCH_ENCODER(2);
            break;
    case 7: SET_START_BIT(0);
            break;
    case 8: RESET_START_BIT(0);
            break;
    case 9: SET_START_BIT(1);
            break;
    case 10: RESET_START_BIT(1);
            break;
    case 11: SET_START_BIT(2);
            break;
    case 12: RESET_START_BIT(2);
            break;
    case 13: SET_OUTAUX0_BIT(0);
            break;
    case 14: RESET_OUTAUX0_BIT(0);
            break;
    case 15: SET_OUTAUX0_BIT(1);
            break;
    case 16: RESET_OUTAUX0_BIT(1);
            break;
    case 17: SET_OUTAUX0_BIT(2);
            break;
    case 18: RESET_OUTAUX0_BIT(2);
            break;
    case 19: SET_OUTAUX1_BIT(0);
            break;
    case 20: RESET_OUTAUX1_BIT(0);
            break;
    case 21: SET_OUTAUX1_BIT(1);
            break;
    case 22: RESET_OUTAUX1_BIT(1);
            break;
    case 23: SET_OUTAUX1_BIT(2);
            break;
    case 24: RESET_OUTAUX1_BIT(2);
            break;
    case 25: read_digital_input(0);
            fDigitalInput[0] = (float)ENC_DIGITAL_INPUT[0];
            break;
    case 26: read_digital_input(1);
            fDigitalInput[1] = (float)ENC_DIGITAL_INPUT[1];
            break;
    case 27: read_digital_input(2);
            fDigitalInput[2] = (float)ENC_DIGITAL_INPUT[2];
            break;
    case 28: ENABLE_ENCODERS(0);
            break;
    case 29: ENABLE_ENCODERS(1);
            break;
    case 30: ENABLE_ENCODERS(2);
            break;
    case 31: DISABLE_ENCODERS(0);
            break;
    case 32: DISABLE_ENCODERS(1);
            break;
    case 33: DISABLE_ENCODERS(2);
            break;
    case 34: ENABLE_DIAXIS(0);
            break;
}
```



Center for Prototyping Services

Mechatronics Laboratory



```
case 35: ENABLE_DIAXIS(1);
        break;
case 36: ENABLE_DIAXIS(2);
        break;
case 37: DISABLE_DIAXIS(0);
        break;
case 38: DISABLE_DIAXIS(1);
        break;
case 39: DISABLE_DIAXIS(2);
        break;
case 40: if (IS_ENC_ENABLED(0) == 1) fAnswer = 0.1;
        break;
case 41: if (IS_ENC_ENABLED(1) == 1) fAnswer = 0.2;
        break;
case 42: if (IS_ENC_ENABLED(2) == 1) fAnswer = 0.3;
        break;
case 43: if (IS_DIAXIS_ENABLED(0) == 1) fAnswer = 0.4;
        break;
case 44: if (IS_DIAXIS_ENABLED(1) == 1) fAnswer = 0.5;
        break;
case 45: if (IS_DIAXIS_ENABLED(2) == 1) fAnswer = 0.6;
        break;
}

/* reset the command variable. */
fCommand = 0.0;
}
/*----- End of user_ctrl() -----*/

/***** User's BACKGROUND ROUTINE (not related with sampling time) *****/
void user_background(void)
{

}
/*----- End of user_background() -----*/

/***** User's STOP ROUTINE (called at dspstop command) *****/
void user_stop(void)
{

}
/*----- End of user_stop() -----*/

/***** User's RESTART ROUTINE (called at dspgo command) *****/
void user_restart(void)
{

}
/*----- End of user_restart() -----*/

/***** Place here below your own functions *****/
```



Bibliography

- [1] S. Carabelli, “OpenDSP System Overview”, Politecnico di Torino, 1998
- [2] E. Miranda and M. Chiaberge, “The OpenDSP General Purpose I/O ver. 1.1”, Politecnico di Torino, 1998
- [3] www.hp.com/HP-COMP/isolator/6n137.html, “High CMR, High Speed TTL Compatible Optocouplers”
- [4] <http://www.altera.com/html/literature/lf10k.html>, “FLEX 10K Embedded Programmable Logic Family Data Sheet, ver. 1.0”
- [5] <http://www.ti.com/sc/docs/psheets/abstract/datasht/slls145b.htm>, “SN65175, SN75175 QUADRUPLER DIFFERENTIAL LINE RECEIVERS”
- [6] <http://www.national.com/pf/LM/LM348.html>, “LM348 – Quad 741 Operational Amplifier”
- [7] Andrea Delmastro and Eduardo Miranda, “OpenDSP Bus”, Politecnico di Torino, 1998
- [8] Andrea Argondizza and Andrea Delmastro, “MatDSP ver. 3.2 for OpenDSP System notes”, Politecnico di Torino, 1998