

LaurTec

**5 Servo Motors
Controller Board**

Author : *Mauro Laurenti*

email : info.laurtec@gmail.com

ID: PJ30002-EN

License

The usage of any project, tutorial, software given by Mauro Laurenti (the Author) imply the acceptance of the following license.

The projects, tutorials, software (material) supplied herewith by Mauro Laurenti is intended for use solely and exclusively for personal use.

All the material is owned by the Author, and is protected under applicable copyright laws.

All rights are reserved.

Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws (Italian or International ones), as well as to civil liability for the breach of the terms and conditions of this license.

Commercial use is forbidden without a written acknowledgment with the Author.

Personal or educational use is allowed if the application containing any part of the given material doesn't aim to commercial use or monetary gain of any kind.

ALL THE MATERIAL IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS MATERIAL. THE AUTHOR SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

All other trademarks belong to their respective owners

Table of Contents

| | |
|-------------------------------------|----|
| Abstract..... | 4 |
| Specification..... | 4 |
| Project Overview..... | 4 |
| Assembling instructions..... | 11 |
| LED Interface..... | 12 |
| CAN Protocol Formatting..... | 14 |
| L2000-30002 Commands..... | 16 |
| ARE_YOU_THERE_COM..... | 16 |
| ORIGIN_POSITION_COM..... | 16 |
| WRITE_EEPROM_COM..... | 16 |
| READ_EEPROM_COM..... | 17 |
| READ_TEMPERATURE_COM..... | 17 |
| SET_POSITION_COM..... | 17 |
| READ_CURRENT_COM..... | 18 |
| IS_THERE_ACTU_COM..... | 18 |
| WARNING_ERROR_COM..... | 18 |
| RESET_BOARD_COM..... | 18 |
| CAN_TX_RX_COUNTER_COM..... | 18 |
| READ_HW_SW_VERSION_COM..... | 19 |
| BOOT_COMPLETE_COM..... | 19 |
| EEPROM..... | 20 |
| Standard EEPROM contents..... | 20 |
| Board specific EEPROM contents..... | 21 |
| Future Developments..... | 23 |
| History..... | 24 |
| Bibliography | 25 |

Abstract

When the Servo Motors control is only part of a much more complex system, you often need to delegate control aspects to the board to which the servo motors are connected. Through the board L2000-30002 is possible, thanks to the presence of a microcontroller, to delegate lower level controlling, leaving the system the ability to make decisions. The communication between the board and the system can be done using the protocol CAN, I2C or through the USART. The small size and computing power makes this board even suitable for making small robots. Each board can control 5 servo motors but more boards may be connected on the same bus in order to expand the number of controllable Servo motors.

Specification

Size: 43mm x 72mm

weight: 33g

Vcc¹: 5V ± 5%

Vdd²: 5V-6V depending on the used Servo Motors.

Icc³: 10-20mA (it depends on the LED status)

Idd⁴: It depends on the used servo motors. Max. 4A.

Isr⁵: 5mA @ 1Ω-1A

Control bus: CAN, I2C, UART

Number of Servo Motors: max. 5

Project Overview

In Figure 1 is shown the schematic of the L2000-30002 board, used to control 5 servo motors. The design core is the PIC microcontroller PIC18F2580 with the CAN engine. In the following documentation the assumption that the reader is familiar with the CAN bus protocol is made. More information can be found within the document “CAN bus”⁶. Within the document, the words “controlling system” are used to indicate the system to which the board L2000-30002 is connected to and from which is getting and sending the commands.

The main role of the board, beside communicating with other boards or controlling system, is to control the servo motors connected to the SV1-SV5 connectors. The connector pin out is as follow:

pin 1: Ground

pin 2: Vdd

pin 3: Servo control line

The resistors R13, R14, R18, R19 e R3 are used to avoid damages in case the servo motor is mounted improperly⁷. The Servo motors get the power from the SV7 connector which must be connected to the power supply board L2000-30003. Vdd power supply can be changed accordingly to

¹ Vcc is the voltage used to power the digital section.

² Vdd is the voltage used to power the Servo Motors.

³ Icc is the current sinked by Vcc.

⁴ Idd is the current sinked by Vdd.

⁵ Current resolution for the Servo current read out.

⁶ All the documentation just mentioned can be downloaded on www.LaurTec.com.

⁷ The protection is done limiting the current coming out form the PIC controlling pins.

the servo motors that are used. By the way since just one Vdd can be selected, the boards must host servo motors with the same Vdd specification.

Thanks to the resistors R6-R10 it is possible to measure the current that get sinked by each servo motor, allowing measuring the torque of each motor. This measure allows to protect the board from the servo that got damaged or start sinking too much current. The current measures also allow to calibrate the board, understanding when the servo has reached the end for the servo motor movement freedom.

The current measurement is made using the analog PIC inputs AN0-AN4. Between the resistor and the analog inputs there aren't the operational amplifiers as used on the board L2000-30001. This limits the used components and the size of the board itself. For the board specification this is not actually limiting the resolution to unacceptable values, otherwise an operational amplifier would have been required. The resistor R6-R10 have been chosen of low value, to limit the voltage droop and the size itself. Depending on the used Servo motor, it is recommended to use different resistor values. For little servo motors also known as microServo 1Ω 1W resistor is recommended while for standard servo motors that sink 1A 0.1Ω 1W resistor is recommended. This difference is related to the fact that, since we have not used any operational amplifier, to get a good voltage droop on the resistors, different values are required. This let the ADC (Analog to Digital Converter) to properly work. For the Ohm's law the voltage droops is:

$$V = R \cdot I$$

For the people that have analyzed the schematics already with care, you would probably have noticed that there are two different grounds labeled AGND e DGND which stand for Analog Ground and Digital Ground. The digital ground is related to the 5V used to power the PIC microcontroller (Vcc) while the Analog ground is related to Vdd used to power the servo motors. The reason why there are two grounds is due to the fact the the servo motors could created spikes and voltage droops during their movement. To decouple the two power supplies it should have been used two different power supplies. By the way if the two power supplies would be totally decoupled the voltage measurement on the resistors R6-R10 could not be properly done due to the floating ground reference. The solution normally used to overcome this problem is to use either a transformer or an optocoupler or a filter or a resistor.

- The first solution is isolating the two power supplies but the transformers could be quite big and can not be used for DC measurements as in our case.
- The second solution has all the advantages of the first and it can be also used for DC measurements beside AC. The problem related with this solution is that it could be expensive and the circuitry get little bit more complex.
- The third solution is a filter, normally done by an inductor and one or two resistors⁸. This solution does not actually isolate the two power supplies but it filters the interference among these.
- The last solution is the most economic and as the previous one does not guaranty the isolation among the power supplies. By the way it limits the interference among the power supplies and allows the voltage measurement on the resistors. R5 is used for the function just described.

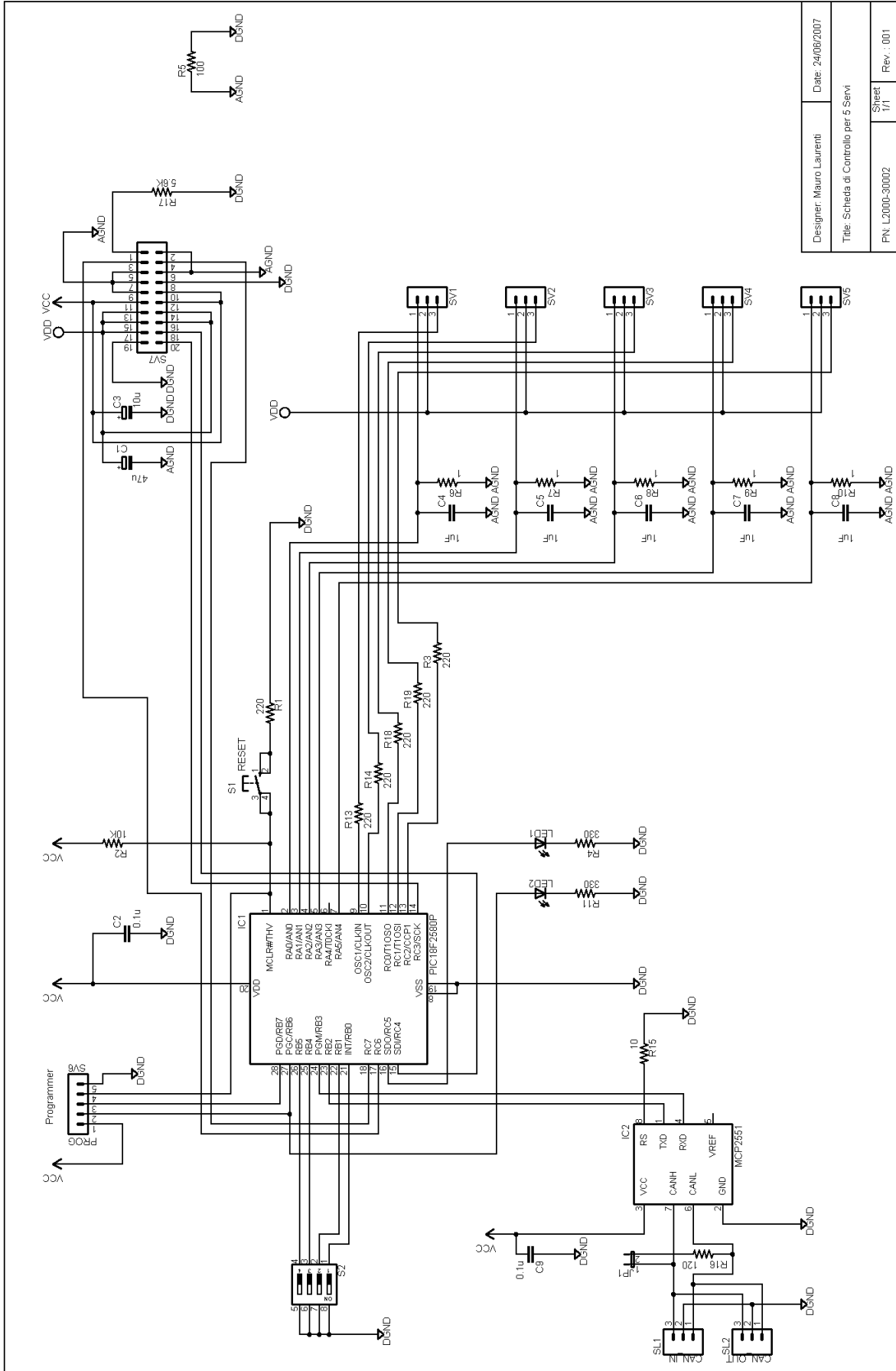
Continuing the hardware description is worth to list some of the hardware that is mounted on board:

- The push button S1 is used to reset the board at any time.
- The LEDs LED1 (green) and LED2 (red) are used as simple visual interface feature.
- 4 mini switch (dip switches) are used to set the board address. This option is handy in systems where is required to have more boards. Without changing the firmware it possible to handle 16

⁸ Variations with capacitors can be easily found.

boards. By the way as it will be shown later, the firmware address allows to change a wired internal address extending even more the boards supported on the same bus or system. It is worth to notice that there are not pull up resistors connected to the dip switches, since the internal PIC pull-up resistors are used (these are available just on PORTB).

Since the servo motors connected to SV1 and SV2 are controlled by the pins normally used for the external oscillator, just the internal oscillator can be used.



| | |
|---|------------------|
| Designer: Mauro Laurenti | Date: 24/06/2007 |
| Titolo: Scheda di Controllo per 5 Servi | |
| PN: L2000-30002 | Sheet: 1/1 |
| | Rev.: 001 |

Figure 1: Schematic

| Components | |
|------------------------------|------------------------------------|
| R1 = 220Ω 5% 1/4W | C6 = 1uF polyester 100V |
| R2 = 10KΩ 5% 1/4W | C7 = 1uF polyester 100V |
| R3 = 220Ω 5% 1/4W | C8 = 1uF polyester 100V |
| R4 = 330Ω 5% 1/4W | C9 = 0.1uF polyester 100V |
| R5 = 100Ω 5% 1/4W | |
| R6 = 1Ω / 0.1Ω 5% 1W | IC1 = PIC18F2480P o PIC18F2580P |
| R7 = 1Ω / 0.1Ω 5% 1W | IC2 = MCP2551 |
| R8 = 1Ω / 0.1Ω 5% 1W | |
| R9 = 1Ω / 0.1Ω 5% 1W | LED1 = green led 3mm |
| R10 = 1Ω / 0.1Ω 5% 1W | LED2 = red led 3mm |
| R11 = 330Ω 5% 1/4W | |
| R13 = 220Ω 5% 1/4W | JP1 = 2 pins jumper |
| R14 = 220Ω 5% 1/4W | S1 = micro-switch button |
| R15 = 10Ω 5% 1/4W | S2 = switch-dil 4 |
| R16 = 120Ω 5% 1/4W | |
| R17 = 5.6KΩ 5% 1/4W | SL1 = con-amp-quick MO3 |
| R18 = 220Ω 5% 1/4W | SL2 = con-amp-quick MO3 |
| R19 = 220Ω 5% 1/4W | SV1 = 3 pins Servo motor connector |
| | SV2 = 3 pins Servo motor connector |
| C1 = 47uF electrolytic 50V | SV3 = 3 pins Servo motor connector |
| C2 = 0.1uF electrolytic 100V | SV4 = 3 pins Servo motor connector |
| C3 = 10uF electrolytic 25V | SV5 = 3 pins Servo motor connector |
| C4 = 1uF polyester 100V | SV6 = Programmer |
| C5 = 1uF polyester 100V | SV7 = ML20 |

The board L2000-30002 has been designed to execute commands that are sent by a controlling system, such as a PC. To let the programmer be comfortable with his preferable interface, different protocols are supported, Currently the firmware supports the CAN interface but easy firmware changes could be done to support I2C bus or the EUSART interface. The reason why the CAN bus has been chosen is due to the fact that the board has been thought to be used within robotic applications.

The resistor R16 must be connected using the jumper JP1⁹ depending on whether or not you have to terminate the CAN bus. This termination is not part of the CAN protocol, since the physical layer is not described, by the way some other standards have been defined around the CAN bus and the transceiver MCP2551 requires this termination. The connection to the bus is made through the connectors SL1 and SL2 labeled CAN_IN and CAN_OUT¹⁰.

As mentioned Beside the CAN bus also the I2C and the EUSART interface is supported, but the bus connections are located to the connector SV7, also used to power the board.

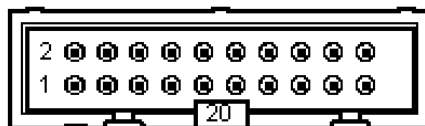
⁹ Just the first and the last node must be terminated in the bus obtaining the total bus impedance of 60Ω.

¹⁰ Since the two connectors are actually in parallel, IN and OUT can be actually swept.

The R15 10Ω resistor is used to regulate the transition edge slope, reducing the emitted radiation. This resistor can be changed depending to the design specification¹¹.

For space reason the pull-up resistor required for the I2C bus are not mounted on the board, this is due to the fact that just one system must terminate the bus any way. It is wort to point that among the bus traces SDA and SCL there is a ground trace to reduce the noise interference among the traces that are routed one next to the other. The resistor R17 apparently connected without any purpose has actually the function to identify the board. Depending on the resistor value the power board could recognize the connected board. The SV7 connector pin out is shown in Table 1.

SV7 pin out:



| PINs | Functions |
|------|---------------------------|
| 1 | Identifier resistor 5.6KΩ |
| 2 | AGND |
| 3 | Pin RC6 (TX EUSART line) |
| 4 | Pin RC7 (RX EUSART line) |
| 5 | AGND |
| 6 | AGND |
| 7 | AGND |
| 8 | DGND |
| 9 | AGND |
| 10 | +5V |
| 11 | +5V |
| 12 | +5V |
| 13 | Vdd |
| 14 | Vdd |
| 15 | Vdd |
| 16 | Vdd |
| 17 | Vdd |
| 18 | Pin RC4 (SDA I2C line) |
| 19 | DGND |
| 20 | Pin RC3 (SCL I2C line) |

Table 1: SV7 Connector pin-out

In Table 2 are shown all the connections used on the PIC. This table is particularly useful once you design a new application.

From what has been just described is possible to understand that the board L2000-30002 can be also used as main board for little robots. It will be up to the firmware to properly control the system.

¹¹ For more information about the value setting of this resistor is recommended reading the official datasheet.

| PIN | PORT | Feature |
|-----|---------|------------------|
| 1 | #MCLR | Board Reset di |
| 2 | RA0/AN0 | SV1 current sens |
| 3 | RA1/AN1 | SV2 current sens |
| 4 | RA2/AN2 | SV3 current sens |
| 5 | RA3/AN3 | SV4 current sens |
| 6 | RA4 | Not used |
| 7 | RA5/AN4 | SV5 current sens |
| 8 | Vss | DGND |
| 9 | RA7 | SV1 control line |
| 10 | RA6 | SV2 control line |
| 11 | RC0 | SV3 control line |
| 12 | RC1 | SV4 control line |
| 13 | RC2 | SV5 control line |
| 14 | RC3 | I2C SCL signal |
| 15 | RC4 | I2C SDA signal |
| 16 | RC5 | LED1 (green) |
| 17 | RC6 | EUSART TX signal |
| 18 | RC7 | EUSART RX signal |
| 19 | Vss | DGND |
| 20 | Vdd | +5V |
| 21 | RB0 | Bit_0 switch S2 |
| 22 | RB1 | Bit_1 switch S2 |
| 23 | RB2 | CAN TX signal |
| 24 | RB3 | CAN RX signal |
| 25 | RB4 | Bit_2 switch S2 |
| 26 | RB5 | Bit_3 switch S2 |
| 27 | RB6 | LED2 (red) |
| 28 | RB7 | Not used |

Table 2: PIC connections

Assembling instructions

The board L2000-30002 is made with a double layer PCB and through hole components. Its assembly is not very difficult since no SMD components are used, but some attentions are required anyway. To simplify the assembly, the PCB has a component lithography with names. Figure 2 shows the assembled PCB. For the assembly is useful to follow the rule of highness, mounting the low profile components first. The resistors will be mounted first then the diodes. The resistors must be mounted according to the color coding while the diodes must be mounted according to the cathode and anode as shown in Figure 2.

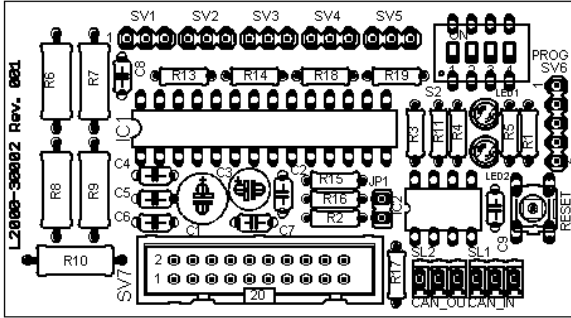


Figure 2: PCB assembly

The ICs should be mounted on IC sockets, this will avoid damaging the IC during the assembling and keeping the replacement easier¹³. This is also handy considering that you can mount different PIC on the board. After the IC sockets and the connectors you can mount the electrolytic capacitors, making sure that they are mounted with the right polarity, as shown on the PCB lithography. As shown in Figure 2, the lithography points out the terminal + while on the capacitor cases are normally shown the terminal -.

At the end of the assembly the board will be as shown in Figure 2 or so; in fact some components could be not mounted at all if not required¹⁴.

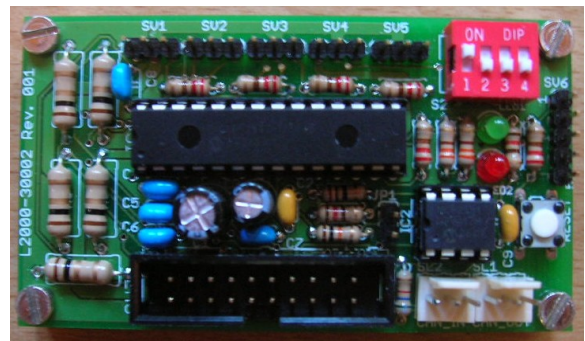


Figure 3: PCB Assembly

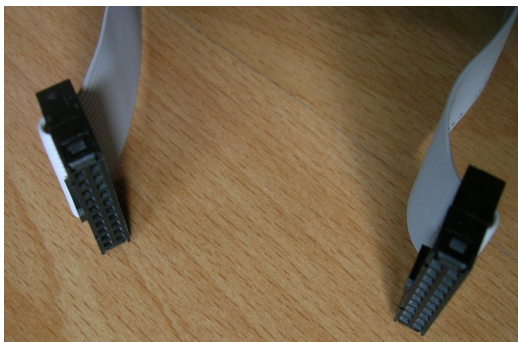


Figure 4: Cable assembly

A little additional note is needed to make the cables required for the board. These must be assembled as shown in Figure 4, make sure that the connector guide pin are located as shown in Figure 4¹⁵.

If you do not have the tool to make the cable assembly you can use a plier or something similar as large as the connector. You must push the connector keeping the pressure uniform on the connector itself, otherwise you can break it. To ensure a good connection you have to press the cable using the first plastic strip and then apply the plastic latch, on which you don't have to apply any compression.

¹² LED diodes have the cathode pointed by a truncated case. Another way to figure out who is the cathode is the pin length, the cathode is the short one.
¹³ The IC socket must be soldered first, the IC must be mounted just at the end of the PCB assembly.
¹⁴ The PCB could be changed without any notice and documentation update.
¹⁵ The ribbon cable is the same one used for the PC floppy disk or hard disk drives. The required ribbon cable has only ten wires. This can be purchased or got out of bigger ribbon cables.

LED Interface

The Servo Motor Controller board has two LEDs for ease visual interface. LED1 is green and LED2 is red. Once the board is powered up or get reset through the push button, both LEDs (red and green) are set to ON until the controlling system has not checked the ability of the board to properly work.

When the controlling system gives the trig to the board using the acknowledgment command¹⁶ the green LED starts blinking simulating the human heart (P and SQR beat are emulated at circa 60ppm) while the red LED is turned off. The red LED is used to show if any warning or error signals have been generated within the board. An error status is shown with the red LED ON and the warning status is shown with a blinking red LED. An error status is shown with higher priority then a warning.

An error status requires an action from the controlling system or from the operator. A warning status is signaled to the controlling system but not necessary requires an action. A warning status could terminate by itself if the warning condition terminates (like the warning current status). Both warning and error messages are signaled through the CAN bus

The following internal signals can generate an error status:

- The CAN engine is bus off. Since the node is off that status is not communicated to the controlling system. The controlling system can understand that through a *spider*¹⁷ application that is checking overtime the status of the board. The node can be reseted through a power reset or through an internal reset of the board. A manual reset would reset the status as well.
- If the current pass the threshold which has been set within the EEPROM it will generate an error. It will be up to the controlling system to determine what it is going wrong.

The following internal signals can generate a warning status:

- `WRONG_COMMAND_ON`: this signal is generated if the board receives a wrong command from the controlling system.
- `WARNING_CURRENT_ON` : this signal is generated if the servo motor is drawing more current than the one set within the EEPROM specification. The EEPROM is reflecting the control system specification¹⁸.
- `CAN_RX_ERR_WARN_ON` : this signal is generated if the CAN module has activated the internal RX error counter ON.
- `CAN_TX_ERR_WARN_ON` : this signal is generated if the CAN module has activated the internal TX error counter ON.
- `ADC_WARNING_ON` : This signal is generated if the ADC could not complete properly the conversion.

The error and warning status are stored within the variables:

```
unsigned char errorFlag = 0x00;
unsigned char warningFlag = 0x00;
```

the following enum structure are used to change the values of the variables above:

```
enum ERROR_FLAGS{
```

¹⁶ Refer to the board commands for more details.

¹⁷ The spider could be a thread that is checking time to time the status of the board.

¹⁸ Within Artorius Humanoid project these setting are written within the Genome file, which is the master for the EEPROM contents.

```
OVER_CURRENT_ON      = 0b00000001,  
OVER_CURRENT_OFF     = 0b11111110,  
CAN_BUS_INACTIVE_ON = 0b00000010,  
CAN_BUS_INACTIVE_OFF = 0b11111101,
```

```
};
```

```
enum WARNING_FLAGS {
```

```
WRONG_COMMAND_ON      = 0b00000001,  
WRONG_COMMAND_OFF     = 0b11111110,  
WARNING_CURRENT_ON    = 0b00000010,  
WARNING_CURRENT_OFF   = 0b11111101,  
CAN_RX_ERR_WARN_ON    = 0b00000100,  
CAN_RX_ERR_WARN_OFF   = 0b11111011,  
CAN_TX_ERR_WARN_ON    = 0b00001000,  
CAN_TX_ERR_WARN_OFF   = 0b11110111,  
ADC_WARNING_ON        = 0b00010000,  
ADC_WARNING_OFF       = 0b11101111,
```

```
};
```

CAN Protocol Formatting

Since the firmware currently supports only the CAN bus, just this protocol formatting will be considered in the next paragraphs.

CAN protocol consist of several fields but for our purposes we can imagine that just the identifier and data field exist, as shown in Figure 5.

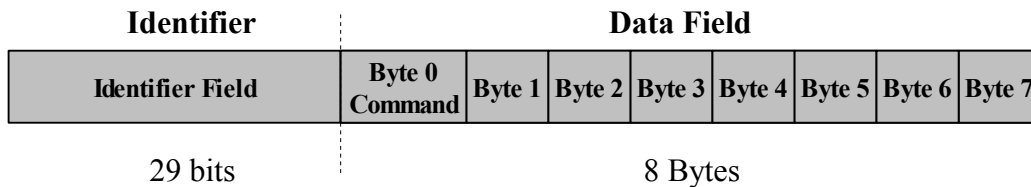


Figure 5: CAN fields structures

Just the extended version identifier will be considered, since it allows more information being transmitted within the same message. The structure shown in Figure 5 has been formatted in a way to support address oriented messages but at the same time keep the advantage of the message oriented feature which is one important characteristic of the CAN protocol. The message oriented structure could have caused problem due to the amount of nodes (Boards) that are used within the system. At the same time just get rid of the message oriented feature would have been silly. The Data formatting is getting advantages from address and message oriented communication¹⁹.

The Identifier (29 bits) in Figure 5 has been formatted as shown in Figure 6. This formatting is independent from whom is sending the information, the board or the controlling system.

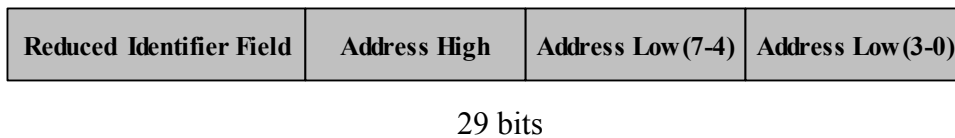


Figure 6: CAN Identifier formatting

Reduced Identifier Field : This field contains 13 bits out of 29 bits of the extended identifier.

Address High : This field it consists of 1 byte which identify the board type. For instance the Servo motors board is identified by 0x55. This byte is useful to manage more than 16 boards of the same type. In fact changing this address will let you address additional boards.

Address Low (7-4) : Each board has a unique address which can be set through the dip switches, those bits are the Address Low. When the dip switch is set to ON it's equal to 1²⁰. Any time the address of the board is changed a board reset is required to update the internal

¹⁹ What it will be described later is just an internal protocol associated with the Firmware. Different formatting would require firmware changes.

²⁰ Since the dip switches are connected to the internal PIC pull-up resistor, to set a 1 the switch needs to be OFF and to set a 0 the switch needs to be ON. An internal PIC function is actually inverting this value to let have 1 with ON and 0 with OFF.

variables.

Address Low (3-0) : This field consists of 4 bits which identify the actuator index.

The identifier can assume the values written in Table 3. The x x x x are used for the Address High and Address low fields.

| Type | Identifier | Meaning |
|------|---------------------|--|
| I. | 0 x 0 A 0 0 x x x x | Command from the controlling system to the addressed Servo board |
| II. | 0 x 0 A 0 1 x x x x | Command from the controlling system for all servo boards |
| III. | 0 x 0 A 0 2 x x x x | Servo feedback from the board |
| IV. | 0 x 0 A 3 0 x x x x | Error info from servo board |
| V. | 0 x 0 A 4 0 x x x x | Warning info from board |

Table 3: *Valid Identifiers*

As just described it is possible to understand that each Servo motor is pointed by the controlling system, using the identifier. The data field is formatted as shown in Figure 7.

| | | | | | | | |
|----------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| Command | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|----------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|

Figure 7: CAN Identifier formatting

Only the Byte 0 has a fixed place and it represents the command that is requested. All the boards that get a command request will reply writing the command back to Byte 0 location and using the identifier Type III. In that way the controlling system it will be able to discriminate all the messages that is getting back from the boards. The bytes from Byte 1 to Byte 7 have different meaning depending on the command. More information are explained later in this documentation.

All the nodes should retrieve the error and warning informations from other boards. This behavior can be used has a simple way to let nodes cooperate in case of problems without the controlling system interaction.

If you use the board L2000-30004 to convert the RS232 PC data stream to CAN signals and vice versa, please refer to the Project Document PJ30001-EN that you can Download from www.LaurTec.com . This document will be useful to cross the information about the two formats.

L2000-30002 Commands

Each Servo Motor Controller board can be controlled through the CAN bus commands. The commands can be sent by any node that is aware of the existence of the board. This is useful in case the controlling system computation is spread over multi PC or for basic boards cooperation. The command type which is specified by Byte 0 of Figure 7, are organized through the following enum structure:

```
enum COMMANDS {
    ARE_YOU_THERE_COM           = 0x00,
    ORIGIN_POSITION_COM         = 0x05,
    WRITE_EEPROM_COM            = 0x15,
    READ_EEPROM_COM             = 0x20,
    SET_POSITION_COM            = 0x30,
    READ_CURRENT_COM            = 0x35,
    IS_THERE_ACTU_COM           = 0x45,
    WARNING_ERROR_COM           = 0x50,
    RESET_BOARD_COM             = 0x55,
    CAN_TX_RX_COUNTER_COM       = 0x65,
    READ_HW_SW_VERSION_COM      = 0x70,
    BOOT_COMPLETE_COM           = 0x80,
};
```

After each command the board is returning a certain value depending on the operation that has been computed. The returned value is identified through with the feedback identifier.

- **ARE_YOU_THERE_COM = 0x00**

This command is sent by the controlling system to check if the board is connected. If the board is not connected no answer will be received. If a timeout is encountered the board should be considered disconnected.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x00

Byte 1: 0x01 if the card is connected

- **ORIGIN_POSITION_COM = 0x05**

This command is sent by the controlling system to set all the Servo motors to the origin position which is written within the EEPROM. No additional fields are required.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x05

Byte 1: 0x01 if the operation has been properly computed, 0 if an error occurred

- **WRITE_EEPROM_COM = 0x15**

This command is used to write a byte within the internal PIC EEPROM.

Send**Byte 1:** EEPROM address**Byte 2:** data byte to be written**Returns**

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x15**Byte 1:** EEPROM address**Byte 2:** 0x01 if the byte has been properly written, 0x00 if an error occurred

- **READ_EEPROM_COM = 0x20**

This command is used to read a byte within the internal PIC EEPROM.

Send**Byte 1:** EEPROM address**Returns**

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x20**Byte 1:** EEPROM address**Byte 2:** Read Byte (address)**Byte 3:** Read Byte (address + 1)**Byte 4:** Read Byte (address + 2)**Byte 5:** Read Byte (address + 3)**Byte 6:** Read Byte (address + 4)**Byte 7:** Read Byte (address + 5)

- **READ_TEMPERATURE_COM = 0x25**

Not implemented within the servo motor controller L2000-30002.

- **SET_POSITION_COM = 0x30**

This command is used to set the servo motor position. The servo index is retrieved from the identifier. The decimal degree is used to support high resolution servo motor. For standard servo motors just the Byte 1 setting is enough. The index within the identifier must point the servo motor of interest.

Send**Byte 1:** degree**Byte 2:** decimal degree**Byte 3:** motion speed**Returns**

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x30**Byte 1:** 0x01 if the operation has been properly computed, 0 if an error occurred

- ***READ_CURRENT_COM = 0x35***

This command is used to read the current from each servo motor. The servo index is retrieved from the identifier.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x35

Byte 1: Value of the current expressed in mA (main digit of the value defined as integer)

Byte 2: Value of the current expressed in mA (last digit of the value defined as integer)

- ***IS_THERE_ACTU_COM = 0x45***

This command is used by the controlling system to check if the servo motor of interest is connected.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x45

Byte 1: 0x01 if the servo is connected, 0x00 if the servo is not connected

- ***WARNING_ERROR_COM = 0x50***

This command is used by the controlling system to retrieve the board Error and Warning variables.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x50

Byte 1: errorFlag variable

Byte 2: warningFlag variable

- ***RESET_BOARD_COM = 0x55***

This command is used by the controlling system to reset the board. The boot flag is set to 1 and the command 0x30 is ignored. A boot complete acknowledgment command is required to reactivate the board.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x55

Byte 1: 0x01 if the reset has been completed

- ***CAN_TX_RX_COUNTER_COM = 0x65***

This command is used to retrieve the TX and RX error counter of the CAN engine.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x65

Byte 1: CANgetTXerrorCount ()

Byte 2: CANgetRXerrorCount ()

- ***READ_HW_SW_VERSION_COM = 0x70***

This command is used by the controlling system to get the Hardware and Firmware version of the board.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x70

Byte 1: Board Version

Byte 2: Firmware version

- ***BOOT_COMPLETE_COM = 0x80***

This command is used by the controlling system as acknowledgment for the board to communicate the end of the boot phase. After that command the green LED will start the heart beat. This command must be send after an EEPROM content check is performed. If the content are properly set the board can be activated.

Returns

Identifier type III. The xxxx of Table 3 are set as the received identifier.

Byte 0: Command 0x80

Byte 1: 0x01 if the command is executed, 0x00 if the command is not executed

EEPROM

The board contains an EEPROM²¹ which is used to store the basic settings of the board plus special settings. The first 0x29 EEPROM are used for standard settings.

Standard EEPROM contents

In Table 4 are shown the EEPROM contents.

| Address | Meaning |
|-------------|---|
| 0x00 – 0x0A | PCA Part number (ex. L2000-30002). Character - must be included. ASCII code must be used. |
| 0x0B | Board revision (0..256) |
| 0x0C | Firmware revision (0..256) |
| 0x0D - 0x0F | Last update date from the PC (BCD formatting dd-mm-yy). 0x0D address contains the dd and so on. |
| 0x10 - 0x11 | Last update time from the PC (BCD formatting hh:mm). 0x10 address contains the hh and so on. |
| 0x12 - 0x17 | Serial number of the board (Ex. IT33000001). The text is written using ASCII code and numbers are written using BCD code. 0x12 address contains the I and so on. |
| 0x18 - 0x20 | Board Production Date (BCD formatting dd-mm-yy). 0x18 address contains the dd and so on. |
| 0x21 | This address is used to check the first software run. If the 0x21 content is different from 0xAA it means first software run. If the first software run occurs default value must be written in the board, depending on the board itself. |

Table 4: Standard EEPROM contents

²¹ The internal PIC microcontroller EEPROM is used.

Board specific EEPROM contents

Beside the standard EEPROM contents the Servo Motor Controller has additional fields as written in Table 5. These fields are used to store the Servo Motor information and some basic calibration information. The meaning of each field has been kept as closer as possible to the meaning used with other similar boards such as the L2000-30001 DC Motor controller board. The first servo motor specific EEPROM information, starts from address 0x30. Each Servo motor has assigned ACTUATOR_FIELD = 0x20 bytes. This means that the second servo will have the info starting from address 0x50. The *Italic* fields are not yet supported since no automatic calibration is performed.

| Address | Meaning |
|---------|---|
| 0x30 | MAX_SPAN: This field keeps the maximum span of the actuator without considering the actual constraints, expressed in degree. |
| 0x31 | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x32 | SERVO_MIN: This field keeps the minimum position expressed in equivalent time. This is automatically calculated adjusted by calibration. It depends on the frequency of the Quartz. This is used to create the proper pulse to drive the servo. |
| 0x33 | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x34 | SERVO_MAX: This field is as the field before but pointing the maximum position. |
| 0x35 | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x36 | START_POINT: This field is used to set the starting point of the actuator considering the constraints. |
| 0x37 | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x38 | END_POINT: This field is as the field before but used to set the final point considering the constraints. |
| 0x39 | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x3A | <i>CAL_START_POINT: This field is adjusting the points above through a calibration.</i> |
| 0x3B | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x3C | <i>CAL_END_POINT: This field is adjusting the points above through a calibration.</i> |
| 0x3D | This field is left empty for future decimal points support and structure compatibility with the Motor Controller board. |
| 0x3E | ORIGIN: this field is used to point the origin point where the servo will be placed after an hardware reset or a power cycle. |

| | |
|------|---|
| 0x3F | VOLTAGE_1: This field keeps the most significant digit of the servo voltage . |
| 0x40 | VOLTAGE_2: This field keeps the less significant digit of the servo voltage (decimal points). |
| 0x41 | CURRENT_1: This field keeps the most significant digits of the maximum current which the servo is supposed to draw (defined as integer). If the limit is passed an internal error is generated. Current is expressed in mA. |
| 0x42 | CURRENT_2: This field keeps the less significant digits of the maximum current which the servo is supposed to draw (defined as integer). If the limit is passed an internal error is generated. |
| 0x43 | CURRENT_WARNING_1: This field keeps the most significant digits of the warning current that the servo will draw under stress (defined as integer). If the limit is passed an internal warning is generated. Current is expressed in mA. |
| 0x44 | CURRENT_WARNING_1: This field keeps the most significant digits of the warning current that the servo will draw under stress (defined as integer). If the limit is passed an internal warning is generated. Current is expressed in mA. |

Table 5: Servo Motor special EEPROM contents

History

| Date | Version | Name | Change Description |
|------------|---------|----------------|--|
| 18/07/2009 | 1.0 | Mauro Laurenti | Original Version partially translated by the Italian version |

Future Developments

- Different programmable baud rate support
- Firmware update using boot loader.
- errorFlag should carry more information.
- warningFlag should carry more information.
- Automatic Calibration

Bibliography

[1] www.LaurTec.com : Electronic web page where you can find the most updated document version and further documents.

[2] www.microchip.com : Microchip technology home page. PIC18F4580 and MCP2551 Datasheet can be downloaded from here.

[a] : AN713 “Controller Area Network (CAN) Basics

[b] : AN738 “PIC18C CAN Routine in C”

[c] : AN916 “Comparing CAN and ECAN modules”

[d] : DS39500A “PICmicro® 18C MCU Family Reference Manual”

[3] www.can.bosch.com : CAN bus specification from Bosch.

History

| Date | Version | Name | Change Description |
|------------|---------|----------------|---|
| 18/07/2009 | 1.0 | Mauro Laurenti | Original Version partially translated by the Italian version |
| 24/08/2009 | 1.0a | Mauro Laurenti | Table 1 has been translated in English, since it was left in Italian. |