Technical Report

IRI-TR-15-02

# Staübli work-cell: Embedded controller

Sergi Hernandez Juan

March, 2015



Institut de Robòtica i Informàtica Industrial

## Abstract

This technical report describes the design (both hardware and software) of an embedded controller for the Staüli work-cell available at the Perception and Manipulation Laboratory at IRI. This system is based on a commercial embedded computer (Beaglebone Black [3]) and it is capable of managing all the work-cell devices. This includes a custom planar XY robot, two simple grippers at the end effector of each Staübli robot, a six degrees of freedom force and torque sensor and optionally the two Staülbi robots themselves.
This embedded system also monitors all the safety features integrated into the work-cell to provide updated information about the current state of the work-cell to all the device drivers and control programs. The safety features available at the Staübli work cell include several emergency stop buttons, a laser curtain surrounding the perimeter of the work-cell and two mechanical fuses at the last joint of the Staübli robot. See [6] for a more detailed description of the Staübli work-cell.
This document is intended as a complete reference manual of the embedded system, both from a software and a hardware points of view, for maintinance purposes and also to alow the addition of new features and upgrades in the future.

**Institut de Robòtica i Informàtica Industrial (IRI)**
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

**Corresponding author:**
Sergi Hernandez Juan
tel: +34 93 401 0857
shernand@iri.upc.edu
http://www.iri.upc.edu/staff/shernand

# 1 Introduction

The Perception and Manipulation Laboratory at IRI has a work-cell build around two Staübli RX60 serial manipulators with six degrees of freedom each. This work-cell has been extended over the years with a custom two degrees of freedom planar XY robot build on top of the whole cell, two simple grippers placed at the last joint of each of the Staübli robots and a six degrees of freedom force and torque sensor that can be placed in any manipulator.

The safety features integrated into the work-cell include a laser curtain that covers all the perimeter of the work-cell, two mechanical fuses at the end effector of each Staübli robot to cut power in case of excessive force and also several emergency stop buttons to be used by human operators: one for each robot and another one for the whole work-cell.

One of the main problems to set this work-cell in working order was that most of the devices listed before were not operational or were not even installed, and the ones that were, had some software and hardware requirements that made it very difficult to integrate them into the current software framework used at the Institute (the ROS middleware [7]).

So, it was decided to develop a new controller system capable of managing all the work-cell devices (robots, sensors, actuators and safety features), and integrate them in a homogeneous framework that made it easy for any one at IRI to use them. This work has been divided into two technical reports, this one which covers the design and development of an expansion board for an existing embedded controller, and [6] which covers the work-cell structure and integration of all the devices.

Fig 1 shows a block diagram of the different modules of the expansion board described in this technical report. The Beaglebone Black works as the central processing unit and runs ROS over an Ubuntu Linux distribution. This embedded computer can work in a standalone configuration for simple applications, or connect to a much larger ROS network over the Ethernet interface. Its main features and design considerations are described in detail in section 3.

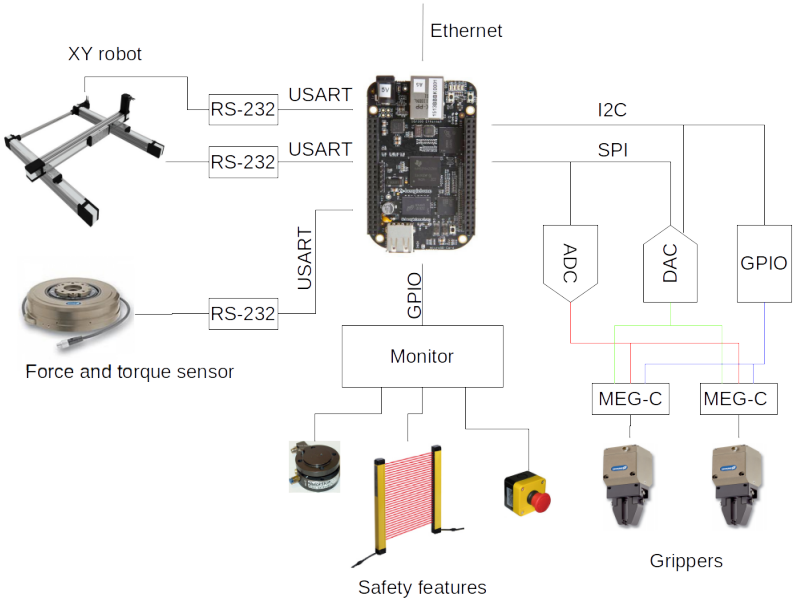
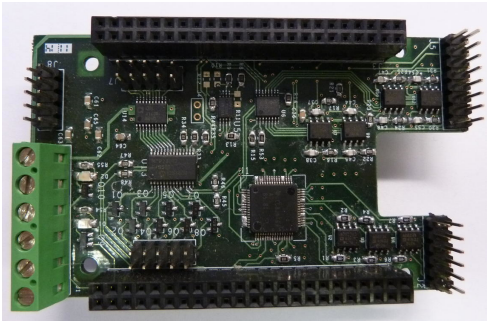
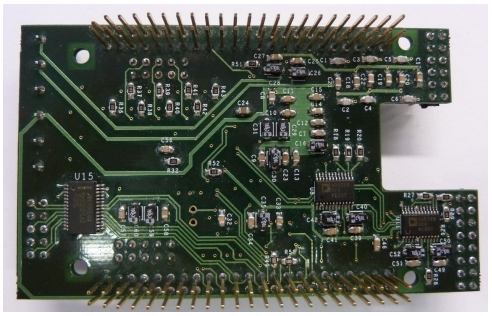



Figure 1: Block diagram of the different modules of the expansion board.

The expansion board described in this technical report has been designed in such a way that it has a wider range of features than the ones strictly required by the application. These include



(a) Top view of the developed expansion board



(b) Top view of the developed expansion board

Figure 2: Top and bottom views of the expansion board for the Staübli work-cell

a programmable output dynamic range for the analog outputs, a wide input dynamic range for the analog inputs and high voltages tolerant digital inputs and outputs. Fig 2a and 2b show the top and bottom view of the developed expansion board respectively.

Interfacing the gripper controllers to the embedded computer proved to be the most challenging part of the design, because both analog and digital electronics are needed. The design of each of the required modules to interface to the gripper controllers is presented in section 4. Next, section 5 describes the interface to monitor the work cell safety features to provide updated information about the current state of the system. Finally, section 6 covers the interface to control the XY robot and get data from the force sensor.

## 2 Power supplies

The expansion board has quite heterogeneous power supply requirements. Table 1 summarizes the voltages and maximum currents required for each of the modules presented later in the following sections. The values presented in Table 1 are worst case maximums, and the actual power requirements of the system will be well below these values on normal operational conditions.

Table 1: Summary of the power requirements of all the modules on the expansion board.

| | $5\,V$ | $3.3\,V$ | $+15\,V$ | $-15\,V$ | $24\,V$ |
|---|---|---|---|---|---|
| Grippers | $36\,mA$ | $9\,mA$ | $63\,mA$ | $30\,mA$ | $160\,mA$ |
| Beaglebone Black | $1\,A$ | 0 | 0 | 0 | 0 |
| XY robot<br>Force sensor | 0 | negligible | 0 | 0 | 0 |
| monitor | 0 | 0 | 0 | 0 | negligible |

The Beaglebone board already provides a $3.3\,V$ rail with a maximum output current of $500\,mA$ which is shared with some internal devices. Even though, it is more than enough to provide the required power to the modules on the expansion board. All other voltages are generated outside of the expansion board due to space limitations, and supplied to it through an external connector (J1). Fig. 3 shows the position and pinout of this connector.

The $24V$ supply voltage is taken directly from the $24V$ power rail powering the whole Staübli work cell (see technical report [6] for more details). The other power supplies are generated by a dedicated switched power supply, the TMP 15515C from TracoPower, which provides up to $2\,A$ for the $5\,V$ supply and up to $150\,mA$ for each of the $\pm 15\,V$ supplies. Fig. 4 shows a diagram of the power supply architecture.

| J1 Connector | |
|---|---|
| **Pin** | **Function** |
| 6 | VCC_EXT |
| 5 | GND |
| 4 | VCC_5 |
| 3 | GND |
| 2 | VCC_+15 |
| 1 | VCC_-15 |

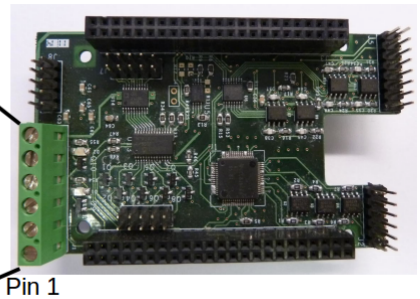



Figure 3: Pinout and position on the expansion board of the power connector (J1).

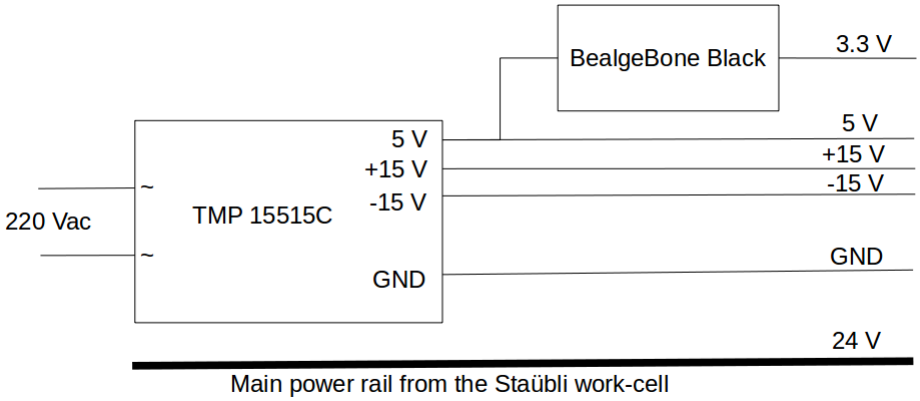



Figure 4: General sketch of the power supply architecture for the expansion board.

It is convenient to use the 3.3 $V$ power supply from the Beaglebone Black to reduce the expansion board size and component count, but it is also dangerous, because this voltage supply is also used internally by vital parts of the Beaglebone Black board (such as the Flash memory). Unexpected surges on this power rail, due to short circuits or accidental connections, may damage the Beaglebone Black board beyond repair.

# 3 Beaglebone Black

The Beaglebone Black is a low cost, small form factor embedded computer based on a 32 bits ARM architecture. Its main features are summarized in Table 2.

The board provides two 46 pins expansion headers to give access to several connectivity interfaces (such as USART, SPI and $I^2C$) and general purpose inputs and outputs. The expansion boards connected to this expansion headers are known as capes. Each cape may have an $I^2C$ EEPROM memory which provides information to the embedded system of the resources required by each of them, to avoid conflicts when several capes are used together (up to four capes can be stacked together). See section 3.6 for more details.

The Beaglebone black board uses the Device Tree framework ([1]) to initialize all the on-board components. The Device Tree is a data structure for describing hardware that is load at boot time. Before the Device tree was used, each embedded board without a BIOS needed a custom Linux kernel to properly initialize all its specific hardware components, which was difficult to develop and maintain. Now, it is possible to develop somewhat generic Linux distributions that

Table 2: Main features of the Beaglebone Black board.

| Feature | Value |
|---|---|
| **Processor** | Sitara AM3358BZCZ100 @ 1GHz (dual core) |
| **RAM** | 512 MB DDR3L @ 800 MHz |
| **Storage** | 4 GB MMC<br>SD-Card |
| **Peripherals** | 1 USB Host<br>1 USB device<br>1 Serial port<br>1 HDMI display<br>1 Ethernet port |

use the Device Tree to properly initialize the specific hardware of each board.

The expansion board makes extensive use of the communication devices and GPIO's available at the expansion connectors as shown in Fig. 1. The next few sections describe in detail the configuration and use of these communication devices.

## 3.1 $I^2C$ bus

The $I^2C$ (inter-integrated circuit) is a multi-master, multi-slave, single-ended, half-duplex, serial computer bus used for attaching low-speed peripherals to computer motherboards and embedded systems ([9]). There exist several specifications regarding speed (up to 5 $MHz$), but the Beaglebone Black only supports the 100 $KHz$ and 400 $kHz$ versions.

As shown in Fig. 5, this bus is used to connect the GPIO expander to handle the digital inputs and outputs of the grippers (see section 4.1.1 for more details), and the digital potentiometers used to electronically change the dynamic output range of the analog outputs for the gripper module (see section 4.1.2 for more details). Fig. 5 also shows the $I^2C$ slave addresses assigned to each of the devices.

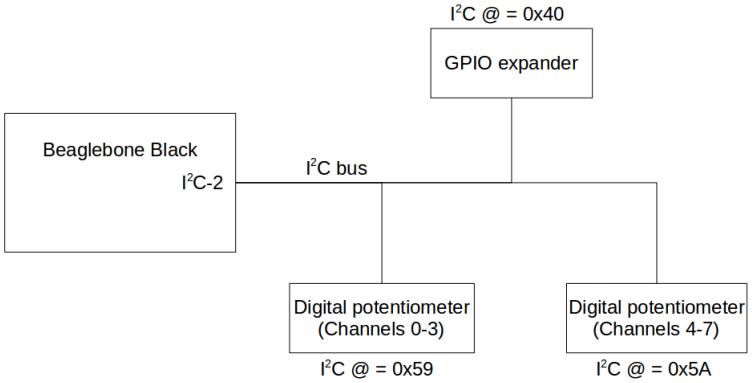



Figure 5: Devices connected to the $I^2C$ bus of the Beaglebone Black board with their addresses on the bus.

All $I^2C$ addresses are hard coded and can not be changed. Therefore, it will not be possible to connect more than one of these expansion boards to a single Beaglebone Board.

All theses devices are connected to the second $I^2C$ port (I2C1) of the Beaglebone Black board, which is not initialized by default. Listing 1 shows the device tree overlay used to initialize the desired $I^2C$ port. It is important to note that the Linux device name assigned to

it is i2c-2, because two other $I^2C$ ports have been previously initialized (i2c-0 used internally by the Beaglebone board, and i2c-1 used for the cape memories).

Listing 1: Device Tree overlay for the I2C1 bus of the Beaglebone Black

```
fragment@0 {
  target = <&am33xx_pinmux>;
  __overlay__ {

    bb_i2c1_pins: pinmux_bb_i2c1_pins {
      pinctrl-single,pins = <
        0x158 0x72 /* i2c1_sda, SLEWCTRL_SLOW */
                   /*          INPUT_PULLUP */
                   /*          MODE2 */
        0x15c 0x72 /* i2c1_scl, SLEWCTRL_SLOW */
                   /*          INPUT_PULLUP */
                   /*          MODE2 */
      >;
    };

  };
};

fragment@4 {
  target = <&i2c1>;
  __overlay__ {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&bb_i2c1_pins>;

    /* this is the configuration part */
    clock-frequency = <100000>;
    #address-cells = <1>;
    #size-cells = <0>;
  };
};
```

The *fragment*@0 code in Listing 1 is only the part of the whole code associated to the initialization of the $I^2C$ signals. The *fragment*@4 code initializes the $I^2C$ port itself, assigning a clock frequency of $100\,kHz$. Refer to the appendix A for the whole device tree overlay for the expansion board.

Linux provides a simple library to allow access to $I^2C$ devices. On Ubuntu distributions it can be easily installed executing the following command:

```
sudo apt-get install i2c-tools
```

This library also installs some tools that are useful to handle $I^2C$ devices. Specially the *i2cdetect* tool can be used to check that the on-boards devices are properly detected before launching any control application.

In order to use the $I^2C$ ports from the user space, it is necessary to add an udev rule to the linux system. For simplicity, the group name used in this section and the ones that follow is

*staubli*. Any user should belong to this group in order to be able to use the $I^2C$ ports. Listing 2 shows the udev rule used to give read and write permissions to any $I^2C$ bus on the system to the members of the *staubli* group.

Listing 2: Udev rule to give access to all I2C ports to the staubli user

```
KERNEL=="i2c-*", GROUP="staubli", MODE="0660"
```

The udev rule presented here is only executed when a new device is created, which only happens at boot time or when a new Device Tree is loaded. Therefore, it will be necessary to reboot the system for the changes to take effect.

## 3.2 SPI bus

The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems ([10]). SPI devices communicate in full duplex mode using a master-slave architecture with a single master. Multiple slave devices are supported through selection with individual chip select lines.

As shown in Fig. 6, this bus is used to connect both the Analog to digital converter to get the current position of the gripper jaws and the digital to analog converter to control the position, force and speed of the gripper jaws (see sections 4.1.2 and 4.1.3 for more details). Fig. 6 also shows the chip select lines used or each converter.

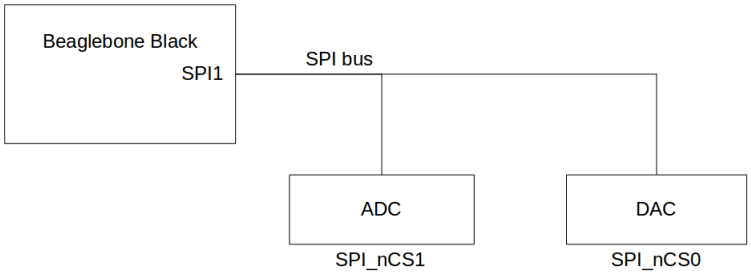



Figure 6: Devices connected to the SPI bus of the Beaglebone Black board with their chip select lines.

Although any GPIO pin can be used to select which device is accessed in the SPI bus, it is better to use the dedicated chip select lines of the peripheral to improve performance. Because the chip select lines are dedicated signal, it will not be possible to connect more than one expansion board on a single Beaglebone Black. However it will still be possible to connect other SPI devices to the same bus using other chip select signals available.

The Only available SPI bus on the expansion headers of the Beaglebone Black is the SPI1, which is not initialized by default. In this case, some of the signals required for the SPI interface are used by the HDMI module, which is initialized by default. So first, it is necessary to disable this module. To do that, first mount the boot partition to a temporary folder (the folder must exist prior to the execution of the following command).

```
sudo mount /dev/mmcblk0p1 /mnt/card
```

Then, edit the uEnv.txt file with your favorite text editor and add the following line:

```
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

After saving the file and rebooting the system, the HDMI module should no longer be loaded at boot time, and the necessary signals are freed for the SPI interface. Listing 3 shows the device tree overlay used to initialize the desired SPI port.

Listing 3: Device Tree overlay for the SPI1 bus of the Beaglebone Black

```
fragment@0 {
  target = <&am33xx_pinmux>;
  __overlay__ {

    bb_spi1_pins: pinmux_bb_spi1_pins {
      pinctrl-single,pins = <
        0x190 0x33/* mcasp0_spi1_sclk INPUT_PULLUP */
                  /*                   MODE3 */
        0x194 0x33/* mcasp0_spi1_d0 INPUT_PULLUP */
                  /*                MODE3 */
        0x198 0x13/* mcasp0_spi1_d1 OUTPUT_PULLUP */
                  /*                MODE3 */
        0x19c 0x13/* mcasp0_spi1_cs0 OUTPUT_PULLUP */
                  /*                 MODE3 */
        0x164 0x12/* mcasp0_spi1_cs1 OUTPUT_PULLUP */
                  /*                 MODE2 */
      >;
    };

  };
};

fragment@1 {
  target = <&spi1>;
  __overlay__ {
    #address-cells     = <1>;
    #size-cells        = <0>;
    status             = "okay";
    pinctrl-names      = "default";
    pinctrl-0          = <&bb_spi1_pins>;
    cs-gpios           = <&gpio4 17 0>, <&gpio1 7 0>;

    channel@0{
      #address-cells     = <1>;
      #size-cells        = <0>;
      compatible         = "spidev";
      reg                = <0>;
      spi-max-frequency = <1000000>;
    };
    channel@1{
      #address-cells     = <1>;
      #size-cells        = <0>;
```

```
        compatible          = "spidev";
        reg                 = <1>;
        spi-max-frequency = <1000000>;
      };
    };
  };
```

The *fragment*@0 code in Listing 3 is only the part of the whole code associated to the initialization of the SPI signals. Refer to the appendix A for the whole device tree overlay for the expansion board. The *fragment*@1 code initializes the SPI bus itself. Notice that two channels are specified, each one will use a different chip select signal, and a separate Linux device will be created for each one, *spidev1.0* for channel 0 and *spidev1.1* for channel 1.

The Linux operating system does not provide any special interface to program the SPI devices, but they can be accessed as any file or device with the *open*, *close*, *read* and *write* functions. Listing 4 shows the udev rule used to give read and write permissions to any SPI device on the system to the members of the *staubli* group.

Listing 4: Udev rule to give access to all SPI devices to the staubli user

```
KERNEL=="spidev*", GROUP="staubli", MODE="0660"
```

### 3.3 Serial ports

Three standard serial ports are required by the expansion board, two to control the custom XY robot, and another one to handle the force and torque sensor. The Beaglebone Black board provides up to 5 serial ports through the expansion headers, but none of them are initialized by default, except for the */dev/ttyO0*, which is used as the default terminal, that is configured at boot time.

Listing 5 shows the device tree overlay used to initialize the serial ports 1,2 and 4 of the Beaglebone Black. It is important to note that the Linux device name assigned to these serial ports do not coincide with the physical port number: */dev/ttyO1* is assigned to the *uart2* peripheral, */dev/ttyO2* is assigned to *uart3* and */dev/ttyO4* is assigned to *uart5*.

Listing 5: Device Tree overlay for the three serial ports needed

```
fragment@0 {
  target = <&am33xx_pinmux>;
  __overlay__ {
    bb_uart1_pins: pinmux_bb_uart1_pins {
      pinctrl-single,pins = <
        0x184 0x00
        0x180 0x20
      >;
    };
    bb_uart2_pins: pinmux_bb_uart2_pins {
      pinctrl-single,pins = <
        0x154 0x01
        0x150 0x21
      >;
    };
    bb_uart4_pins: pinmux_bb_uart4_pins {
      pinctrl-single,pins = <
```

```
            0x070 0x26
            0x074 0x06
          >;
        };
      };
    };
    fragment@2 {
      target = <&uart2>;  /* really uart1 */
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart1_pins>;
      };
    };
    fragment@3 {
      target = <&uart3>;  /* really uart2 */
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart2_pins>;
      };
    };
    fragment@5 {
      target = <&uart5>;        /* really uart4 */
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart4_pins>;
      };
    };
```

The *fragment*@0 code in Listing 1 is only the part of the whole code associated to the initialization of the serial port signals. Refer to the appendix A for the whole device tree overlay for the expansion board. Each of the fragments 2, 3 and 5 in Listing 1 initialize a single serial port. The specific configuration of each port is given when the serial port is opened inside an application.

The selection of these serial ports instead of any other was mainly based on simplifying the routing process of the final PCB. No udev rule is needed for the serial ports, but all the users that must have access to them must be added to the *dialout* group, in this case only the *staubli* user.

Any serial port can be assigned to any function, but Table 3 shows the default and recommended configuration.

Table 3: Default assignment of the serial ports

| **serial port** | **function** |
|---|---|
| */dev/ttyO1* | XY robot, X axis |
| */dev/ttyO2* | XY robot, Y axis |
| */dev/ttyO4* | Force and Torque sensor |

## 3.4 GPIO

Some modules of the expansion board have control and/or status signals that are handled by general purpose inputs and outputs (GPIO) of the Beaglebone Black board. Linux provides a simple and flexible way to access and configure GPIO signals through the file system interface (*sysfs*). By default, only the root user is capable of doing so, but it is also possible to grant access to standard users.

To do so, first, it is necessary to give read and write access to the desired user (*staubli* in this case) to the files *export* and *unexport* in folder */sys/class/gpio*. These files are used to create new instances of GPIO signals and delete existing ones respectively. To do so, edit the */etc/rc.local* file using your favorite text editor, and add the following lines at the end:

```
chgrp -R staubli /sys/class/gpio
chmod -R g+rw /sys/class/gpio
```

Then, the udev rule shown in Listing 6 is required to give access to the desired user to the newly created GPIO instances. These instances are dynamically created and destroyed, so the udev mechanism is the only way to modify them so that standard user may access them.

Listing 6: udev rule to give access to the GPIO instances

```
SUBSYSTEM==”gpio”, DEVPATH==”/devices/virtual/gpio/gpio*”, ACTION==”add”,
PROGRAM=”/bin/sh -c 'chown -R staubli:staubli /sys/devices/virtual/gpio/gpio%n'”
SUBSYSTEM==”gpio”, DEVPATH==”/devices/virtual/gpio/gpio*”, ACTION==”add”,
PROGRAM=”/bin/sh -c 'chmod g+w /sys/devices/virtual/gpio/gpio%n/*'”
SUBSYSTEM==”gpio”, DEVPATH==”/devices/virtual/gpio/gpio*”, ACTION==”add”,
PROGRAM=”/bin/sh -c 'chown -R staubli:staubli /sys/class/gpio/gpio%n/*'”
```

A C++ class has been developed around the file system interface to the GPIO signals in order to simplify their use. This class provides the main features listed next, and it is described in more detail in its documentation page (see [5]).

- The desired GPIO port and GPIO pin are selected when the object is created, and can not be modified afterwards.
- Configure any GPIO either as input or output.
- As an output, select the desired output value.
- As an input, read the current input value and also create and configure an event to notify the user on occurrence of a desired transition.

Although simple, this kind of interface to the GPIO signals can not be used when the required toggling frequencies are above a few kilohertz (around $7kHz$). In these cases, other methods may be used, such as the *mmap* (direct memory mapping of processor registers) or the development of a kernel driver. However, all these alternatives are far more complex than the file system interface, and they are not required for this application.

Table 4 summarizes the GPIO pins of the Beaglebone Black used by the expansion board. The direction shown in Table 4 are viewed from the Beaglebone Black side.

## 3.5 Expansion board pinout

Fig. 7 shows the pinout of the two connector used to interface with the expansion headers of the Beaglebone Black (J3 and J6).

Table 4: GPIO used by the expansion board

| Name | Port | Bit | Direction | Description |
|---|---|---|---|---|
| nRESET | GPIO1 | 18 | output | Resets the DAC |
| nLDAC | GPIO1 | 17 | output | Updates the output values of the DAC |
| ADC_START | GPIO2 | 22 | output | Starts a conversion of the ADC |
| ADC_BUSY | GPIO2 | 23 | input | Current state of the ADC conversion |
| INT_A | GPIO2 | 24 | input | Interrupt signal for port A |
| INT_B | GPIO2 | 25 | input | Interrupt signal for port B |
| CELL_CHAIN | GPIO2 | 5 | input | Work-cell operating status |
| ES_CELL | GPIO0 | 27 | input | Work-cell emergency stop button status |
| LIGHT_CURTAIN | GPIO1 | 14 | input | Work-cell light curtain status |
| LEFT_MECH_FUSE | GPIO1 | 12 | input | Left mechanical fuse status |
| RIGHT_MECH_FUSE | GPIO0 | 23 | input | Right mechanical fuse status |
| MODE | GPIO1 | 15 | input | Work-cell operating mode |
| ROBOT_XY_CHAIN | GPIO0 | 26 | input | XY robot operating status |
| LEFT_STAUBLI_CHAIN | GPIO1 | 13 | input | Left Staübli operating status |
| RIGHT_STAUBLI_CHAIN | GPIO1 | 13 | input | Right Staübli operating status |

## 3.6 Cape EEPROM memory

Although it is not mandatory, an $I^2C$ EEPROM can be used to store information identifying the cape and the resources it requires. This EEPROM is read at start-up time, and its information is used by the *cape-manager* program to load the specific Device Tree overlay needed to initialize and reserve the necessary resources (communication ports, software drivers, etc.).

This memory is connected to the second $I^2C$ port of the Beaglebone Black board. Only four $I^2C$ addresses are supported for the cape memory, and in order to allow the use of other capes, the specific $I^2C$ address can be selected by soldering/unsoldering some resistors. Table 5 specifies the required populated resistors to select each of the $I^2C$ addresses.

Table 5: Resistors to be populated to configure the desired $I^2C$ addresses for the cape memory.

| $I^2C$ address | R8 | R9 | R10 | R11 | R12 | R44 |
|---|---|---|---|---|---|---|
| 0x50 | X | X | | | | X |
| 0x51 | | X | | X | | X |
| 0x52 | X | | X | | | X |
| 0x53 | | | X | X | | X |
| 0x54 | X | X | | | X | |
| 0x55 | | X | | | X | |
| 0x56 | X | | X | | X | |
| 0x57 | | | X | X | X | |

Only addresses $0x54$ through $0x57$ are used for the *cape-manager* to load cape information. The other addresses can be used when the memory is only intended to store user non-volatile data.

This memory can not be removed and therefore has to be programmed in-system, but in

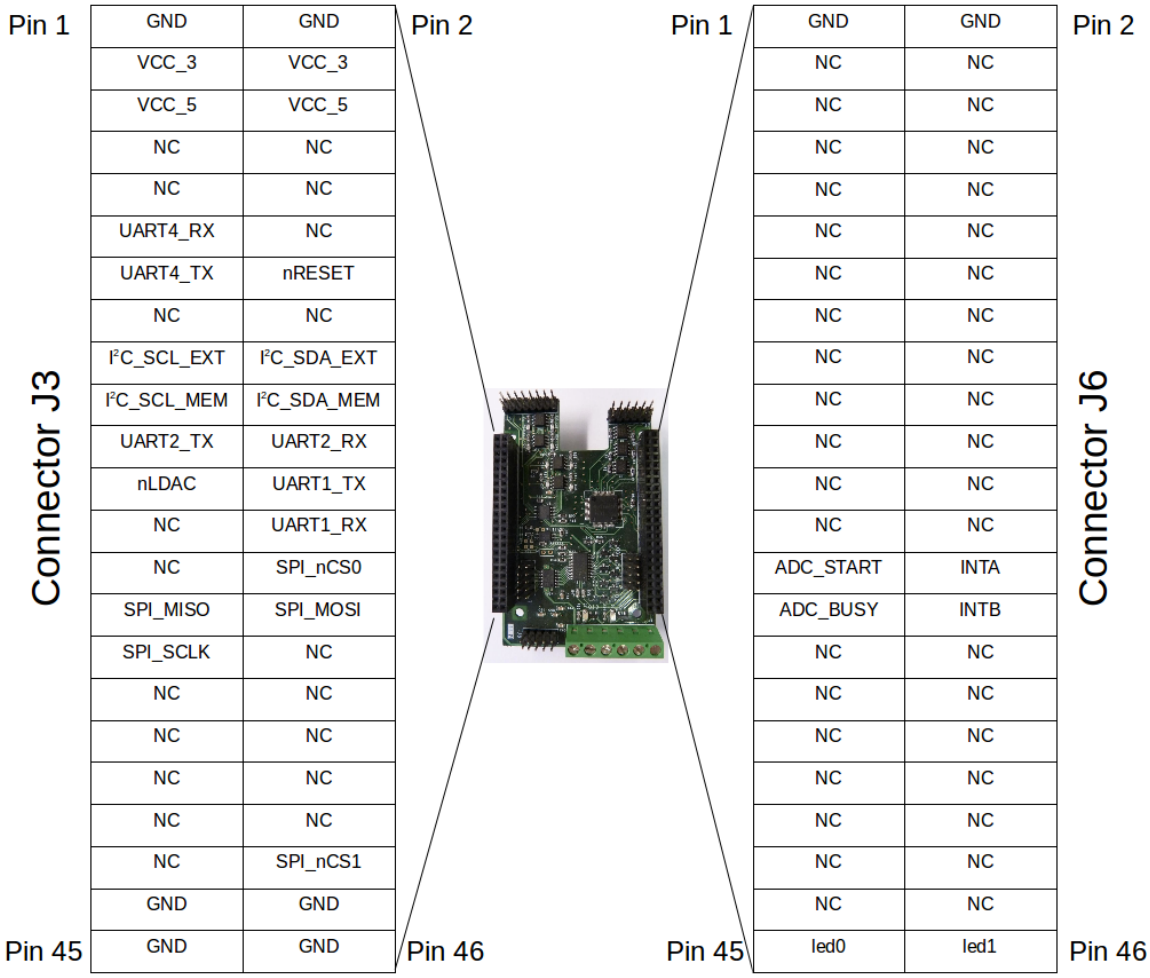



Figure 7: Pinout of the expansion connectors J3 and J6.

order to avoid corruption of its contents during normal operation due to some erroneous write, its write protect feature is enabled by default. A jumper (J4) may be used to temporarily enable it and allow write access to the memory to update its contents.

The manual of the Beaglebone Black (see [2], pages 101 through 105) describes the format of the contents of the EEPROM memory.

# 4 MEG50EC grippers

The grippers used in the Staübli work-cell are the MEG50EC from Schunk. The gripper itself has no embedded controller, so an external one is used. This controller is the MEG-C also from Schunk which allows to control the stroke, force and speed of the jaws by means of analog signals. It also provides digital inputs to control the gripper functions (such as open, close and calibration), and digital outputs to report the status of the gripper. The current position of the jaws is reported also as an analog signal.

The gripper can operate in two different modes: in position mode the jaws move to the desired position at the desired speed until the goal position is reached or the maximum desired force is exceeded. In force mode, the jaws either open or close the the maximum or minim position respectively until the maximum desired force is exceeded.

The gripper does not have any encoders (it uses a stepper motor) which make it necessary to perform a calibration procedure before it can be used in position mode. Otherwise, the behavior of the gripper will change depending on the initial position. However, it can be used in force mode without calibrating. The calibration procedure consists on moving the jaws to the maximum or minimum position until the maximum force is exceeded.

The main feature of the MEG50EC grippers are shown in Table 6. See the product manual for more detailed information ([8]).

Table 6: Main features of the MEG50EC gripper.

| **Feature** | **Value** |
|---|---|
| Power supply | $24\,V$ |
| Position range | $8\,mm$ |
| Force range | $110\,N$ |
| Speed range | $6\,mm/s$ to $32\,mm/s$ |

The need to deal with analog signals increases the complexity of the system. The first approach was to use National Instruments DAQ cards: the NI PCI 6220 for analog input and the NI PCI-6722 for analog output and digital input and output. Although its use simplified the hardware design, the software integration was much more difficult. Only the former card is supported by the Comedi [4] project, which provides open-source drivers, tools, and libraries for data acquisition in Linux.

Therefore, it was necessary to use the Linux drivers provided by the manufacturer, which, at the time of developing this project, were only available for an obsolete Suse Linux distribution. The defacto software platform used at the Institute is an Ubuntu Linux distribution with the ROS framework, and the task of either using the manufacturer drivers on Ubuntu or using ROS on the Suse distribution proved to be too complex.

Given the relatively low resolution required for the stroke, force and speed signals for the target applications, a simpler and cheaper custom data acquisition system was proposed. This new data acquisition system, could not reach the levels of performance of the National Instruments cards, but it fulfills all the requirements, and also it has been easily integrated into the Ubuntu and ROS software framework.

First, the hardware design of the different modules is presented in section 4.1, and an overview of the low level C++ software drivers and the corresponding ROS wrappers are presented later in section 4.2.

## 4.1 Hardware design

Mainly, the resources needed to fully control each gripper are:

- 4 digital outputs with a maximum output level of $24\,V$ to reset the controller, and start the the opening, closing and calibration motions.
- 3 digital inputs with a maximum input level of $24\,V$ to monitor the status of the gripper (position reached, calibration done and gripper stopped).
- 1 analog input signal with a dynamic range of $5\,V$ for the position feedback signal.
- 3 analog output signals with a dynamic range of $10\,V$ to control the stroke, force and speed of the jaws.

### 4.1.1 GPIO

The two grippers require a total of 8 digital outputs and 6 digital inputs. The Beaglebone Black board has more than enough GPIO pins available, but an external GPIO expander device was used instead. The reason behind this decision was that a GPIO pin of the Beaglebone Board, if damaged, can not be easily replaced, and also, exposing pins of the processor to external signals may cause damage to other parts of the board, rendering it useless. An external GPIO expander can be easily replaced if damaged, and the processor may remain intact in case of any problem.

The latency of all inputs and outputs is not crucial for the correct operation of the grippers, so a low speed communication interface may be chosen. The MCP23017 with an $I^2C$ interface and two 8 bit independent ports has been selected. As presented before in section 3.1, this GPIO expander is connected to the second $I^2C$ port of the Beaglebone Black and has a fixed $I^2C$ address of 0x40.

Each pin of each port can be configured as either input or output, but in order to simplify the level translation stage required to interface with the digital signals of the gripper, PORT A can only be used for outputs, and PORT B can only be used for inputs.

The GPIO expander used also provides two signals (INTA and INTB) to notify an external device of any activity on the input ports. Both signals are connected to the Beaglebone Black to avoid having to continuously poll the inputs over the $I^2C$ bus to detect changes. See Table 4 for the specific GPIO pins used.

As presented in section 4.1, the digital inputs and outputs need to operate at least in the $24\,V$ voltage range. Most digital systems use logic levels from $2.5\,V$ to $5\,V$, so a level translator will be needed in both inputs and outputs.

The ADG3123 from Analog devices has been chosen for the output level translator because it provides a low impedance outputs for both high and low logic levels, which make it possible to easily connect them to almost any device. The external side power supply is connected to the VCC_EXT pin of connector J1 shown in Fig. 3. The maximum supply voltage for the external side of the level translator is $35\,V$.

For the input level translator a simple MOS transistor in inverter configuration has been used for each pin (see Fig. 8). With this circuit, the logic level read at the gpio expander will be complementary of the real digital input, but this can be easily handled by software.

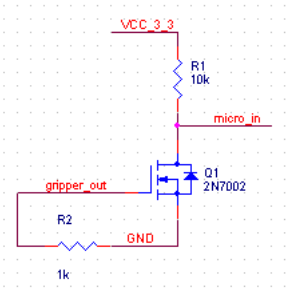



Figure 8: Schematic of the digital input interface

This input interface is independent of the The VCC_EXT power supply used, but the maximum voltage at the input is limited by the maximum gate-source voltage supported by the 2N7002 transistors, which is $30\,V$.

Resistor R2 in Fig. 8 is used to force a default input voltage at the input of the transistor when it is left unconnected. Otherwise, interferences and capacitive effects from other parts of the circuit or the environment itself may arbitrary change the input value readings, and generate spurious interrupts to the Beaglebone Black board processor.

The value of this resistor is quite important because it will create a voltage divider with the output impedance of the connected device, which may degrade the logic levels at the input. Since the outputs of the gripper controller are isolated by an open collector opto-coupler with a low value pull-up resistor (470 Ω), the value of the R2 resistor is set to 1 $k\Omega$ to ensure valid logic levels.

Fig. 9 shows the position of the input (J6) and the output (J7) connectors on the expansion board, as well as the pinout of each connector for the gripper control. If used for an other application, Fig. 9 also shows the GPIO expander pin assigned to each output pin. In this case, the only constraint is that connector J7 can only be used for outputs, and connector J6 can only be used for inputs.

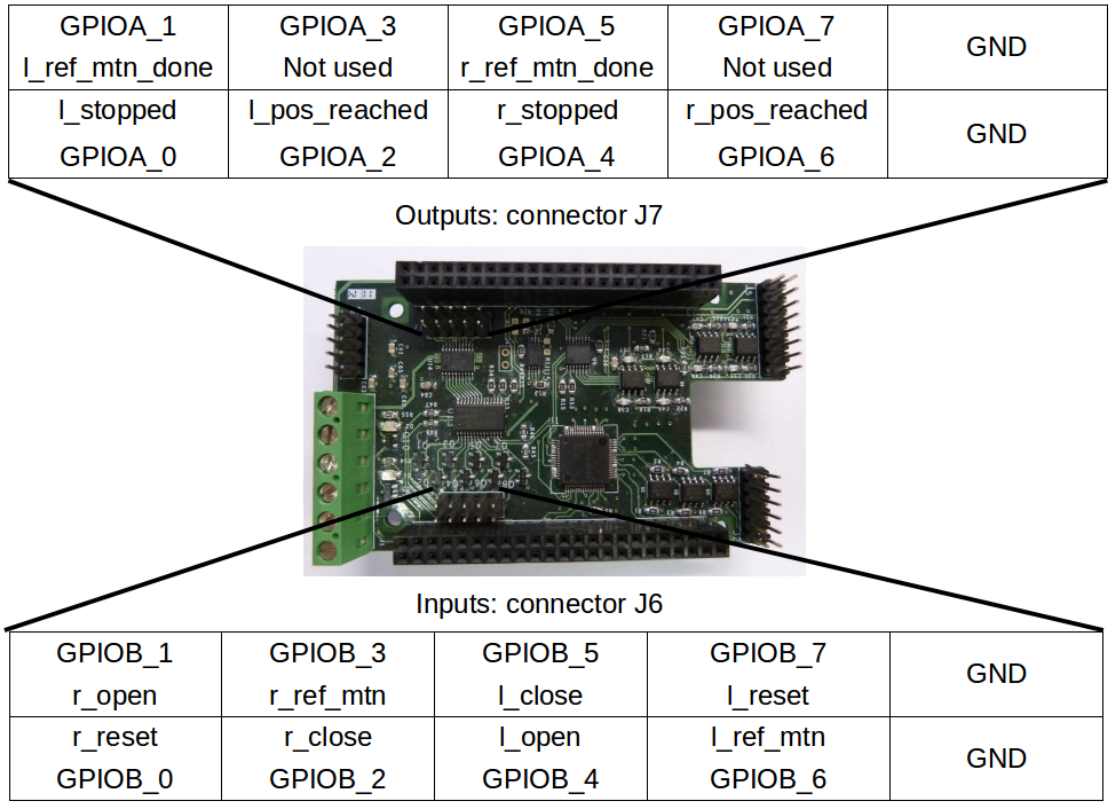



Figure 9: Pinout of the input (J6) and output (J7) connectors of the expansion board.

Appendix B summarizes the overall electrical and timing specifications of the expansion board.

### 4.1.2 Digital to Analog Converter

The two grippers require a total of 6 analog outputs with a dynamic range of 10 $V$ as presented in section 4.1. To generate analog signal from a digital circuit, a Digital to Analog Converter is needed. Such devices, normally use an SPI interface because it offers a better data bandwidth, and so does the AD5668 from Analog devices selected for this application. The main features of this converter are listed in Table 7.

Both the resolution and the bandwidth provided by this converters are more than enough to

Table 7: Main features of the AD5668 Digital to Analog Converter.

| **Feature** | **Value** |
|---|---|
| Power supply | 2.7 $V$ - 5.5 $V$ |
| Internal reference | 1.25 $V$ |
| Interface | SPI |
| Resolution | 16 bits |
| Bandwidth | 320 kHz |
| Max. output voltage | $2V_{ref}$ |
| Num. channels | 8 |

handle the position, speed and force control signals of the grippers, but these parameters allow to widen the scope of use of the designed expansion board.

Although the desired output data values are transmitted using the SPI interface, to actually get the desired voltage at the outputs, it is necessary to use a control signal (nLDAC). Check Table 4 for the specific Beaglebone Black GPIO pin used. As shown in table 4, another GPIO pin is used to reset the converter to its default configuration (nRESET).

In order to achieve the desired output voltage range, a variable gain, non-inverting amplifier is used at the output stage (shown in Fig. 10. This amplifier also includes a first order low pass filter to reduce as much as possible the output voltage noise.

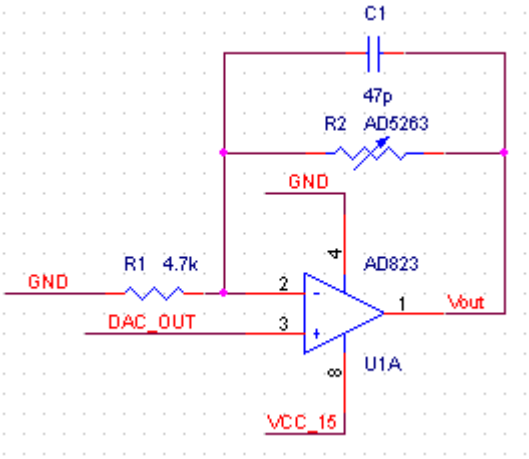



Figure 10: Schematic of the circuit used for each of the analog outputs of the expansion board.

The variable gain is implemented using digital potentiometers which allow to change the gain of the amplifier programatically, instead of manually adjusting a traditional potentiometer. The main features of the AD5263 digital potentiometer from Analog Devices selected for this application are shown in Table 8.

Despite its advantages, digital potentiometers present some limitations compared to their analog counterparts. First, their bandwidth is quite limited, normally below the 1 $MHz$ value. However, in this case the bandwidth of both the Digital to Analog Converter and the digital potentiometer match (as shown in tables 7 and 8), which means that no device will limit the other.

The second main limitation, is that the range of voltages in which digital potentiometers can operate is limited by their power supply. Again, in this case, the maximum desired output voltage is 15 $V$, which coincides with the maximum power supply for the digital potentiometer,

Table 8: Main features of the AD5263 digital potentiometer.

| **Feature** | **Value** |
|---|---|
| Interface | $I^2C$ |
| Max. Power supply | $15\,V$ |
| Num. Potentiometers | 4 |
| Resolution | 8 bits ($78.125\,\Omega$) |
| Max. Resistance | $20\,k\Omega$ |
| Bandwidth | $300\,kHz$ |

as shown in Table 8.

In order to be able to independently adjust the gain on each of the 8 available analog outputs, two of these digital potentiometers are needed, with addresses 0x59 and 0x5A as shown in Fig. 5.

The operational amplifier used is the AD823 from Analog Devices. The main features of this amplifier are summarized in Table 9.

Table 9: Main features of the AD823 operational amplifier.

| **Feature** | **Value** |
|---|---|
| Bandwidth | $16\,MHz$ |
| Slew rate | $22\,\mu V/s$ |
| Max. Power supply | $36\,V$ |

The output voltage of the amplifier stage as a function of the Digital to Analog Converter output and the value of the digital potentiometer (R2) is:

$$V_{out} = \frac{\frac{1}{R_1||R_2C_1} + s}{\frac{1}{R_2C_1} + s} V_{DAC}. \tag{1}$$

From Eq. 1 is simple to get the DC gain and the zero and pole frequencies as:

$$G_{DC} = 1 + \frac{R_2}{R_1}, \omega_{pole} = R_2C_1, \omega_{zero} = R_1||R_2C_1 \tag{2}$$

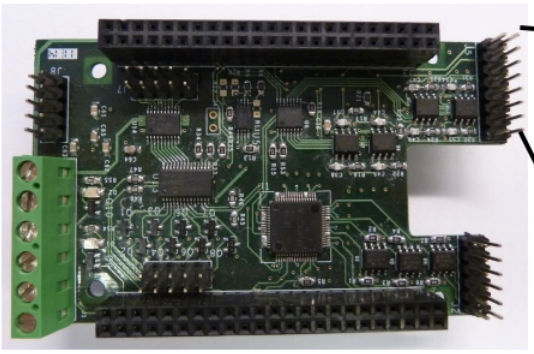
It is easy to see that, with the values shown in Fig. 10 and Table 8, the output voltage range can be modified from $2.5\,V$ to a little over $13\,V$, which is enough for the desired application.

Since the operational amplifier used has both an output and input rail-to-rail features, it is not necessary to use a symmetric power supply to get a $0\,V$ output value. The gain-bandwidth product and the slew rate features of the amplifier are also suitable for the gripper control application.

As shown in Eq. 1, the filter implemented by this circuit actually has a zero in addition to the desired pole, which limits the attenuation at high frequencies. Also, both the pole and the zero frequencies depend on the value of the digital potentiometer, which will change the frequency response of the amplifying stage depending on the gain.

Fig. 11 shows the position of the analog output connector (J5) on the expansion board, as well as its pinout for the gripper control. If used for an other application, Fig. 11 also shows the Digital to Analog Converter channels assigned to each output pin.

Appendix B summarizes the overall electrical and timing specifications of the expansion board.



| Connector J5 | | |
|---|---|---|
| DAC_CH0 | l_speed | GND |
| DAC_CH1 | l_position | GND |
| DAC_CH4 | r_speed | GND |
| DAC_CH6 | r_force | GND |
| DAC_CH2 | l_force | GND |
| DAC_CH3 | Not used | GND |
| DAC_CH7 | Not used | GND |
| DAC_CH5 | r_position | GND |

Figure 11: Pinout of the analog output connector (J5) of the expansion board.

### 4.1.3 Analog to Digital Converter

The two grippers require a total of 2 analog inputs with a dynamic range of $5\,V$ as presented in section 4.1. The main features of the Analog to Digital Converter used are shown in Table 10.

Table 10: Main features of the ADS8558 Analog to Digital Converter.

| **Feature** | **Value** |
|---|---|
| Power supply | $2.7\,V$ - $5.5\,V$ |
| Internal reference | $0.5\,V$ to $3\,V$ |
| Interface | SPI |
| Sampling Freq. | 530 kSPS |
| Bandwidth | 320 kHz |
| Max. input voltage | $\pm 12\,V$ |
| Num. channels | 6 |

The input voltage range of the converter can be configured by software to adjust it to the desired input signal, making it possible to maximize the resolution of the converter depending on the application. This Analog to Digital converter has 3 independent SAR converters with a selectable single differential or two single ended inputs. Each converter can be configured independently.

In order to isolate the converter from the external environment, an operational amplifier in voltage follower configuration is used at each input, as shown in Fig. 12. Also a first order low pass filter is inserted between the amplifier and the converter to limit the bandwidth of the input signal and avoid aliasing problems.

The operational amplifier used is the AD823, and its main features are shown in Table 9. In this case the amplifier uses a symmetric power supply to allow bipolar signals to reach the Analog to Digital converter. This first stage provides a high input impedance to the input signals, and also isolates the converter from the output impedance of the connected device, which may negatively affect quality of the conversion.

The cut-off frequency of the low pass filter of Fig. 12 is:

$$f_c = \frac{1}{2\pi R_1 C_1}, \tag{3}$$

which is the one recommended by the manufacturer.

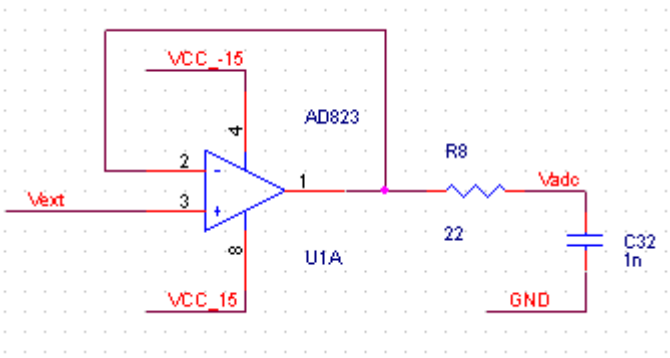
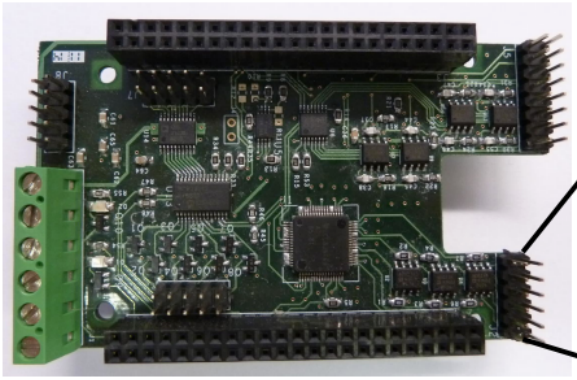



Figure 12: Schematic of the circuit used for each of the analog inputs of the expansion board.

Although the configuration of the converter and the transmission of the data is done through the SPI interface, the Analog to Digital Converter requires an external signal to start the conversion (ADC_START). The converter also provides a second external signal (ADC_BUSY) to indicate its internal state, either busy converting data or idle. These two signals are directly handled by GPIO's of the Beaglebone Black (See Table 4 for the specific GPIO pins used).

Fig. 13 shows the position of the analog input connector (J2) on the expansion board, as well as its pinout for the gripper control. If used for an other application, Fig. 11 also shows the Analog to Digital Converter channels assigned to each input pin.



| Connector J2 | | |
|---|---|---|
| ADC_A0 | Not used | GND |
| ADC_A1 | l_cur_pos | GND |
| ADC_B0 | r_cur_pos | GND |
| ADC_B1 | Not used | GND |
| ADC_C0 | Not used | GND |
| ADC_C1 | Not used | GND |

Figure 13: Pinout of the analog input connector (J2) of the expansion board.

Appendix B summarizes the overall electrical and timing specifications of the expansion board.

## 4.2 Software design

Fig. 14 shows the general software structure and dependencies for the gripper driver. As shown in Fig. 14, the software is mainly divided in three layers: the lower most layer deals with the hardware itself, implementing the communication protocols needed to configure, control and exchange information with the expansion board devices.

The middle layer uses the modules on the previous layer to actually implement the gripper control by binding together the operation of all the low level drivers. Finally, the upper most layer creates a wrapper around the gripper driver to easily interface it to a ROS network.

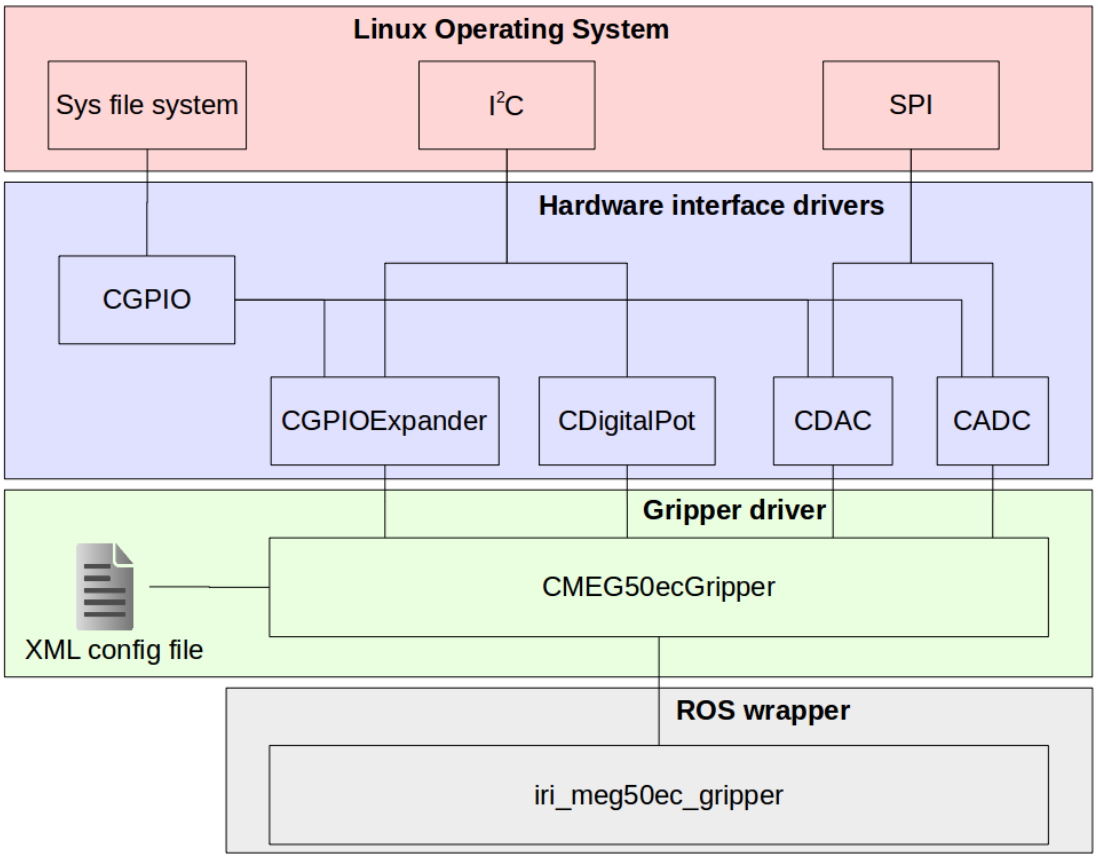



Figure 14: General software structure and dependencies for the gripper. From low level Linux drivers to the ROS framework.

Due to fact that both grippers share most of the hardware resources (analog to digital and digital to analog converters, digital potentiometers and the GPIO expander) the gripper driver in the middle layer, and also the ROS wrapper, controls both grippers, providing an interface that allow the user to choose on which gripper to perform the desired action. Nonetheless, the operation of both grippers is completely independent from each other.

### 4.2.1 Hardware interface drivers

As shown in Fig. 14, there exist a separate Linux low level driver for each of the main hardware modules presented in section 4.1. Although not complete, these modules try to expose as much of the features of the underlying device as possible through their API's, so that they can be used in a more general setup.

In this section, only the main features of these modules are presented. The detailed description for the CGPIO modules has already been presented in section 3.4. For the CGPIOExpander modules see [5], for the CDAC module see [5], for the CADC module see [5] and for the CDigitalPot modules see [5].

The main features provided by the public API of the CGPIOExpander module are listed below. The features provided by this module are quite similar to the one provided by the CGPIO class.

- The GPIO ports and the GPIO pins used by the two external interrupt signals of the device

can be selected when the object is created, but they can not be modified afterwards.

- Configure the general operation of the device, which includes the external interrupt polarity and output type, the $I^2C$ signals slew rate control and the internal registers addressing mode.
- Configure each GPIO pin as either an input or an output. The polarity of each signal and an optional pull up resistor can also be configured for each GPIO pin.
- Create and delete events associated to specific transitions on the input signal for each of the GPIO pins.
- Set the output value when configured as an output pin, and read the input value when configured as an input pin.

The CGPIOExpander class has an internal thread that continuously monitors the external interrupt signal of the device and access its internal registers to generate the desired internal events.

The main features provided by the public API of the CDAC module are listed below:

- Configure the operation of the device, which includes the voltage reference used to perform the conversion and the default value of the outputs at power up.
- Enable and disable the output channels. Each channel can be individually enabled, but it is only possible to disable all the channels at once.
- Update the output value for each channel independently. The update process is split in two: a first stage where the desired output value is stored in internal memory, and a second stage where the value is actually loaded into the output. A separate function exists for each stage, and also a single function which performs all the operations, depending on the case of use.
- Get the current output voltage present at each channel.

The main features provided by the public API of the CDigitalPot module are listed below:

- Set the desired resistance between the two terminals. The maximum resistance allowed is configured when the object is created and can not be modified afterwards.
- Shutdown unused potentiometers to reduce power requirements.

The main features provided by the public API of the CADC module are listed below:

- The GPIO ports and GPIO pins used for the START and BUSY signals can be selected when the object is created, but they can not be modified afterwards.
- Configure the desired input voltage range for each of the internal SAR converters independently and the global voltage reference used for the conversion.
- Enable or disable the operation of each of the internal SAR converters individually.
- Start and stop the execution of an internal thread that periodically gets the current voltage at the inputs. The sampling period is set to 10 $Hz$ and can not be modified. It is only possible to get new samples when the internal thread is running, otherwise, the last converted value will be always returned.

Although the hardware device allows it, this class does not provide the possibility of working with differential signals. It can still be achieved by computing the difference between the two single ended channels of a single SAR converter.

### 4.2.2 Gripper driver

The gripper driver CMEG50ecGripper combines most of the features of the low level driver modules presented in the previous section to generate the control signals and monitor the status signals of the gripper in order to effectively manage it. In this section, only the main features of the gripper driver are presented. For a more detailed description see [5].

Due to the great many parameters needed to properly configure the gripper operation (device names, $I^2C$ addresses, GPIO ports and pins, DAC and ADC channels, etc), an XML file is used to provide a simple an easy way to configure the grippers. The format of this XML file is presented in Appendix C, together with the specific configuration values required to operate the gripper on the expansion board.

As introduced in section 4, the gripper con operate in either position or force mode. Although most of the public functions are common to both modes, a specific API is provided for each one. The main features provided by the public API of the CMEG50ecGripper module are listed below:

- Load the configuration parameters either from an XML file or from a properly initialized C data structure.
- Monitor the current state of the jaws. Whether they are opening, closing or moving in general.
- Set the desired maximum force (in N) the jaws would apply before stopping and the desired speed of the jaws (in mm/s).
- Stop the motion of the jaws at any time.
- Get the current position of the gripper. This position is half the distance between the two jaws.
- Configure the gripper operation in either position or force mode.
- In position mode, set the desired position of each gripper (mm). This has no effect in force mode.
- In force mode, command the gripper to open or close to the maximum or minimum position respectively, or until the configured maximum force is exceeded. This has no effect in position mode.

As introduced in section 4.2, a single driver operates the two grippers in the Staübli work-cell, so most of the API functions need a parameter to specify on which gripper to perform the desired operation.

The CMEG50ecGripper driver has two internal threads that are managed internally. The first thread waits for changes on the digital outputs of the gripper to generate three internal events: when the jaws have stopped due to force limits, when the reference motion has completed to calibrate the gripper and when the target position has been reached. The second thread, periodically monitors the analog output of the gripper which encodes the actual position of the jaws, and updates the internal values.

None of the command functions of the gripper are blocking, except for the calibration function that wait for the reference motion to complete. There exist two different mechanisms to know the current state of the gripper: continuously poll the status functions (is_opening(), is_closing() or is_moving()) or use asynchronous notification through events.

### 4.2.3 ROS wrapper

Fig. 15 shows the topics, services and actions implemented in the ROS wrapper of the gripper driver.

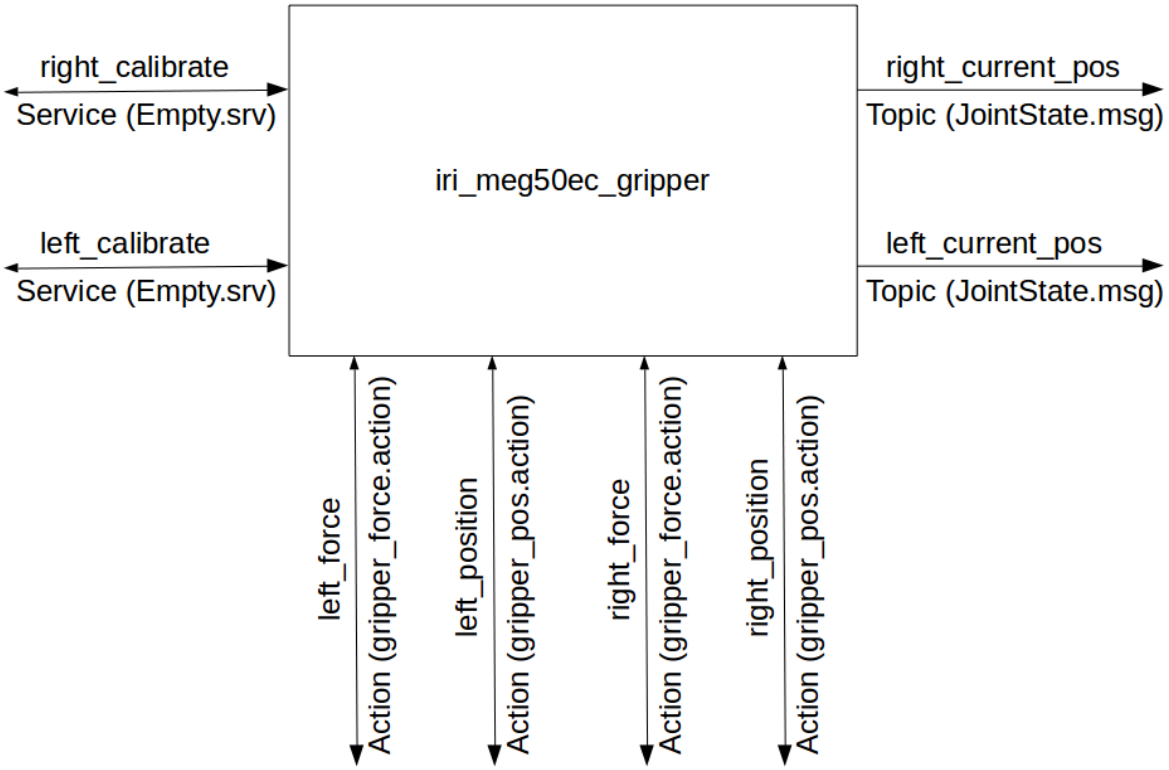



Figure 15: ROS communication interface of the ROS wrapper of the gripper driver

#### Published topics: right_current_pos and left_current_pos

These topics publish a JointState.msg type message with the current position of the grippers so that the robot state publisher can update the gripper model in rviz. The joint type is linear and the position is encoded in meters. The values published by these topics are not valid until the grippers have been calibrated.

#### Service servers: right_calibrate and left_calibrate

These are servers for an Empty.srv type service. When the service request is received, the gripper starts the calibration procedure to set the gripper in a known position. The service does not return until the calibration process has finished.

#### Action servers: right_force and left_force

These are servers for a gripper_force.action type action. The goal, feedback and result fields for this action are shown in Listing 7.

Listing 7: gripper_force.action action definition

```
#goal definition
```

```
bool close
float32 speed
float32 force_limit
---
#result definition
float32 final_position
---
#feedback
float32 current_position
```

When received, this action sets the corresponding gripper in force mode, and performs the action specified by the goal (either open or close) until the gripper reaches a motion limit or the specified force limit is exceeded.

When the action is active, the current position of the jaws is periodically published thought the feedback topic, and when the action finishes, it reports the actual final position reached by the gripper.

#### Action servers: right_position and left_position

These are servers for a gripper_pos.action type action. The goal, feedback and result fields for this action are shown in Listing 8.

Listing 8: gripper_pos.action action definition

```
#goal definition
float32 position
float32 speed
float32 force_limit
---
#result definition
float32 final_position
bool force_limit
---
#feedback
float32 current_position
```

When received, this action sets the corresponding gripper in position mode, and starts moving the grippers to the desired position until it is reached or the specified force limit is exceeded.

When the action is active, the current position of the jaws is periodically published thought the feedback topic, and when the action finishes, it reports the actual final position reached by the gripper, and a flag indicating if the maximum force was exceeded.

#### Parameters

The node has only two parameters, to specify the path and filename of the XML file needed to properly configure the gripper.

- config_path(string,default: "./"): Path to the desired configuration file.
- config_file(string,default: "meg50ec_config.xml"): Name of the desired configuration file.

# 5 Work-cell monitor

The Staübli work-cell has several safety features. A few of the safety features are common to the whole work-cell, and halt the normal operation of all the robots. These general safety features include a laser curtain that surrounds the whole perimeter of the work-cell and a general emergency stop button.

The other safety features are associated to one of the robots. Each of the Staübli manipulators have a mechanical fuse at the last joint, just before the end effector, and an emergency stop button in the control pendant. Regarding the XY robot, the only safety feature of this robot is an emergency stop button.

Check the technical report [6] for detailed information on how these safety features are connected together to generate the safety chain of the whole work-cell and the ones for each particular robot. All these safety features generate 24 $V$ signals that should be monitored in order to know the current state of each of the elements of the work-cell at any time.

It is possible to operate the work-cell in two different modes: one (called *cell mode*) in which all robots work together in cooperation and another one (called *independent mode*) in which each robot can be operated independently from the others.

The selected operation mode changes somewhat how the safety features of the work-cell affect the operation of each robot. While in *cell mode* the activation of any safety feature halt the normal operation of all the robots, in *independent mode*, only the safety features associated to each robot halt the normal operation of the corresponding robot.

## 5.1 Hardware design

This part of the design was a later addition that was included after the board had been designed and built. Therefore it has been implemented as a second expansion board, on top of the one presented in this document. In this case, instead of using a GPIO expander integrated circuit as has been done for the gripper control, the Beaglebone Black GPIO pins have been used.

The Staübli work-cell safety signals monitored by this modules, together with a brief description, are presented in Table 11.

All these signals have a high voltage value of 24 $V$ when the corresponding device is in operation condition, and a value of 0 $V$ otherwise. To translate the voltage level to the 3.3 $V$ range supported by the Beaglebone Black GPIO pins, a simple resistive voltage divider is used as shown in Fig. 16.

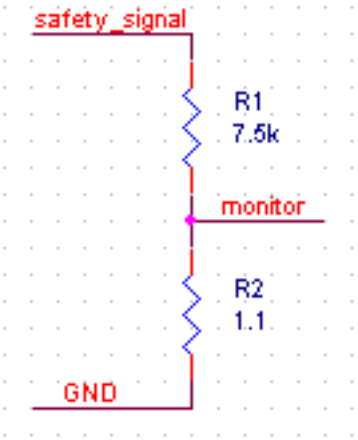



Figure 16: Resistive voltage divider used to translate the safety features voltage level.

Table 11: Safety features monitored by the expansion board presented in this document.

| **Feature** | **Description** |
|---|---|
| cell_chain | This signal reports the state of the whole cell. A high level indicates the cell is in working condition, and a low level indicates that one or more of the individual safety features may not be active. |
| cell_emergency_stop | This signal reports the state of the emergency stop button for the whole cell. It will stop all robots in the work-cell whatever the operating mode selected. |
| light_curtain | This signal reports the state of the laser light curtain surrounding the work cell. It will stop all robots in the work-cell whatever the operating mode selected. |
| left_mechanical_fuse | This signal reports the state of the mechanical fuse at the end effector of the left Staübli robot. Its behavior depends on the operating mode selected. |
| right_mechanical_fuse | This signal reports the state of the mechanical fuse at the end effector of the right Staübli robot. Its behavior depends on the operating mode selected. |
| operation_mode | This signal reports the current operating mode selected in the control pendant. A high level indicates all robots work cooperatively and a low level indicates that each robot works independently. |
| robot_xy_chain | This signal reports the state of the emergency stop button for the XY robot. Its behavior depends on the operating mode selected. |
| left_staubli_chain | This signal reports the state of the emergency stop button for the left Staübli robot. Its behavior depends on the operating mode selected. |
| right_staubli_chain | This signal reports the state of the emergency stop button for the right Staübli robot. Its behavior depends on the operating mode selected. |

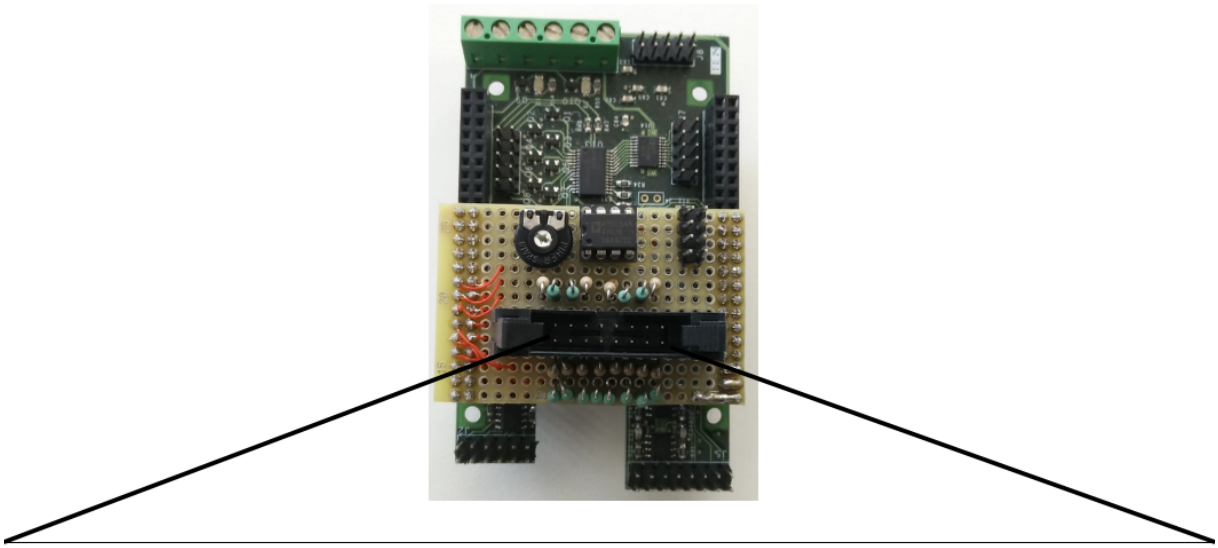
Note that, with the resistor values shown in Fig. 16, the logic high level voltage at the output of the voltage divider is just over $3\,V$. This has been done intentionally to protect the Beaglebone Black GPIO pins from possible voltage overshots present at the safety signals when they switch.

In the future, clamping diodes to $3.3\,V$ may be added to further protect the GPIO pins of the Beaglebone Black board against voltage overshots. Also, a more complete and robust circuit such as the one presented in section 4.1.1 for the gripper outputs may be used instead.

Fig. 17 shows the position of the monitor signals connector on the expansion board, as well as its pinout for the safety features monitor application. If used for an other application, Fig. 17 also shows the Beaglebone Black GPIO pins assigned to each input pin.

## 5.2 Software design

Fig. 18 shows the general software structure and dependencies for the gripper driver. As shown in Fig. 18, the software is mainly divided in three layers: the lower most layer deals with the hardware itself, implementing the interface to the GPIO puns through the CGPIO class presented in section 3.4.



**Monitor Connector**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| GPIO2_1 | Not | GPIO0_27 | Not | GPIO1_14 | Not | Not | GPIO1_15 |
| Not used | connected | cell_es | connected | light_crtn | connected | connected | op_mode |
| Not used | Not used | cell_chain | r_staubli | l_staubli | l_mech_f | r_mech_f | xy_chain |
| GPIO2_2 | GPIO2_3 | GPIO2_5 | GPIO2_4 | GPIO1_13 | GPIO1_12 | GPIO0_23 | GPIO0_26 |

Figure 17: Pinout of the safety features monitor connector of the expansion board.

The middle layer periodically checks the state of the each of the safety features signals and reports their status. Finally, the upper most layer creates a wrapper around the gripper driver to easily interface it to a ROS network.

### 5.2.1 Monitor driver

The CMonitor driver uses several CGPIO class objects to check the status of the Staübli work-cell safety features. In this section, only the main features of the monitor driver are presented. For a more detailed description see [5].

To simplify the configuration process, an XML file is used to provide the GPIO port and GPIO pin value pairs for each of the monitored safety features signals. The format of this XML file is presented in Appendix D, together with the specific configuration values required to operate the monitor module on the expansion board.

The main features provided by the public API of the CMEG50ecGripper module are listed below:

- Load the configuration parameters either from an XML file or from a properly initialized C data structure.
- Configure the rate at which the safety features are monitored.
- Get the current status of all the monitored safety features.

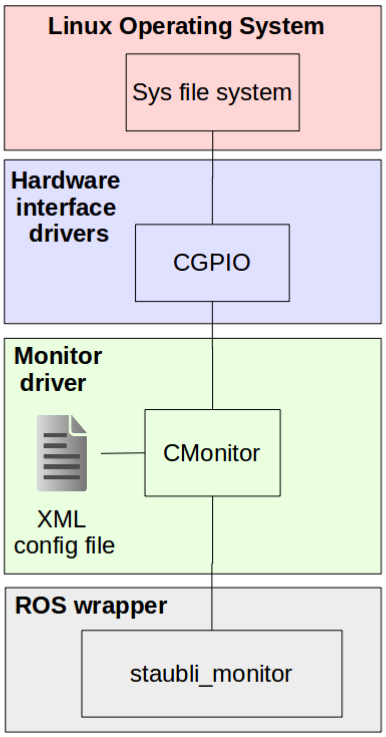



Figure 18: General software structure and dependencies for the monitor. From low level Linux drivers to the ROS framework.

The CMonitor driver has an internal thread which is started only when the object has been properly configured with the appropriate GPIO ports and pins. This thread gets the current value of all GPIO pins (a high level means the safety feature is operational and a low level means something is wrong), and stores them in internal memory until the information is requested by some other application.

### 5.2.2 ROS wrapper

The ROS wrapper for the Staübli work-cell monitor driver is very simple as shown in Fig. 19. The name of the node is taken from the name of the work-cell, which is the blacksmith character of the Asterix comics.

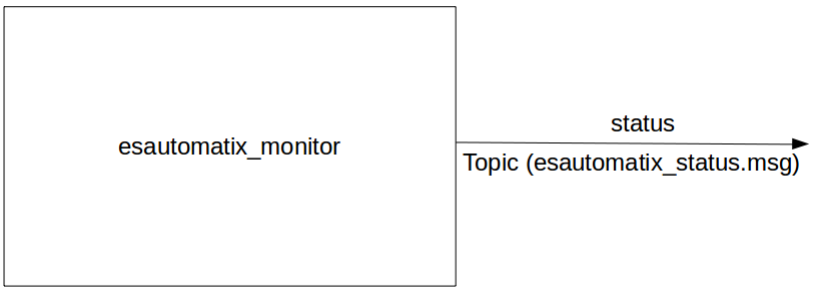



Figure 19: ROS communication interface of the ROS wrapper of work-cell monitor driver.

#### Published topics: status

This topic publishes an esautomatix_status.msg message (see Listing 9 for its format), which reports the status of the whole work-cell at the desired rate. A value of 0 in one of the *chain* fields of the message indicates that the associated robot is not is operating condition. The safety feature field or fields that have a value of 0 will indicate the cause. Any value different from 0 indicate that everything is okay.

Listing 9: esautomatix_status.msg message definition

```
uint8 cell_chain
uint8 cell_emergency_stop
uint8 light_curtain
uint8 left_mechanical_fuse
uint8 right_mechanical_fuse
uint8 operation_mode
uint8 robot_xy_chain
uint8 left_staubli_chain
uint8 right_staubli_chain
```

A value of 0 in the *operating_mode* field indicates that each robots operate independently from each other, and any other value indicates indicates that all robots work cooperatively.

#### Parameters

The node has only two parameters, to specify the path and filename of the XML file needed to properly configure the gripper.

- config_path(string,default: "./"): Path to the desired configuration file.
- config_file(string,default: "monitor_config.xml"): Name of the desired configuration file.
- sampling_rate(int,default: 100): Desired publish rate in Hertz.

## 6 XY robot and force sensor

The force and torque sensor used is the FTC50L from Schunk, which has three degrees of freedom to measure forces and three more to measure torques, and also, it is capable of measuring the deformation of the sensor due to external action. Both data acquisition and sensor configuration is done through a high speed serial port interface with RS-232 logic levels. The maximum speed of this interface is just under 1 $Mbps$ (921600 $bps$ to be exact), so special care should be taken to guarantee the maximum bandwidth.

The XY robot is a custom robot build at IRI which has two independent degrees of freedom, each controlled by a DC motor. Each motor has an optical incremental encoder of 500 pulses per revolution to close the control loop, and forward, reverse and home limit switches in order to initialize and control the motion of each axis. Each axis is controlled by an MCDC2805 motor controller which integrates the motor power stage, the encoder and limit switch interfaces and the PID control loop in a single device. This controller provides a standard low speed serial interface (maximum 19200 $bps$) to configure its operation and also to send motion commands and get information about its behavior.

Both systems require a serial interfaces with RS-232 voltage levels. Since the serial ports provided by the Beaglebone Black use a 3.3 $V$ logic levels, a level translator is required. Care

should be taken when choosing the RS-232 transceiver because not all available devices support the relatively high transmission speeds required by the Force and Torque sensor.

The MAX3237 from Maxim Semiconductors is used, which provides 3 drivers and 5 receivers and it is normally used in applications when a complete serial port is required. It also operates in the $3.3\,V$ range. This level translator has a dedicated pin to enable a special operating mode in which speeds of up to 1 $Mbps$ are possible.

Fig. 20 shows the position of the serial interface connector (J8) on the expansion board, as well as its pinout for the force and torque sensor and the two degrees of freedom XY robot. If used for an other application, Fig. 20 also shows the Beaglebone Black serial devices assigned to each output pin.

| Connector J8 | | | |
|---|---|---|---|
| x-axis | ttyO1_rx | ttyO1_tx | x-axis |
| GND | | GND | |
| y-axis | ttyO2_rx | ttyO2_tx | y-axis |
| GND | | GND | |
| FT-sensor | ttyO4_rx | ttyO4_tx | FT-sensor |

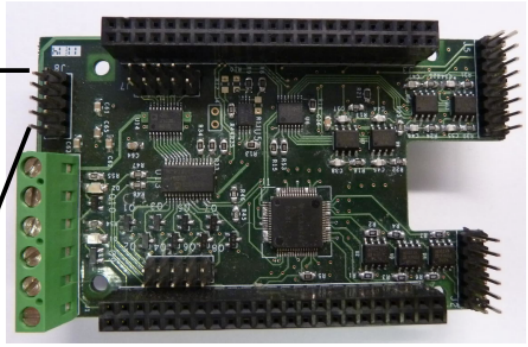


Figure 20: Pinout of the serial interface connector (J8) of the expansion board.

The serial ports can be used as regular Linux devices and no special driver has been developed.

# A Device Tree Overlay

Listing 10 shows the complete Device Tree Overlay used for the expansion board presented in this technical report.

Listing 10: Complete Device Tree overlay for the expansion board

```
/dts-v1/;
/plugin/;

/ {
   compatible = "ti,beaglebone", "ti,beaglebone-black";

   /* identification */
   part-number = "BB-AD-DA-GPIO";
   version = "00A0";

   /* state the resources this cape uses */
   exclusive-use =
      /* SPI pins */
      "P9.42", /* SPI1_CS1 */
      "P9.28", /* SPI1_CS0 */
      "P9.29", /* SPI1_MISO */
```

```
    "P9.30", /* SPI1_MOSI */
    "P9.31", /* SPI1_SCLK */
    /* uart 1 pins */
    "P9.24", /* uart1_txd */
    "P9.26", /* uart1_rxd */
    /* uart 2 pins */
    "P9.21", /* uart2_txd */
    "P9.22", /* uart2_rxd */
    /* uart 4 pins */
    "P9.13", /* uart4_txd */
    "P9.11", /* uart4_rxd */
    /* i2c 1 pins */
    "P9.18", /* i2c1_sda */
    "P9.17", /* i2c1_scl */
    /* Hardware IP cores in use */
    "spi1",
    "uart1",
    "uart2",
    "uart4",
    "i2c1";

  fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {
      bb_spi1_pins: pinmux_bb_spi1_pins {
        pinctrl-single,pins = <
          0x190 0x33/* mcasp0_spi1_sclk INPUT_PULLUP */
                    /*                   MODE3 */
          0x194 0x33/* mcasp0_spi1_d0 INPUT_PULLUP */
                    /*                MODE3 */
          0x198 0x13/* mcasp0_spi1_d1 OUTPUT_PULLUP */
                    /*                MODE3 */
          0x19c 0x13/* mcasp0_spi1_cs0 OUTPUT_PULLUP */
                    /*                 MODE3 */
          0x164 0x12/* mcasp0_spi1_cs1 OUTPUT_PULLUP */
                    /*                 MODE2 */
        >;
      };
      bb_uart1_pins: pinmux_bb_uart1_pins {
        pinctrl-single,pins = <
          0x184 0x00
          0x180 0x20
        >;
      };
      bb_uart2_pins: pinmux_bb_uart2_pins {
        pinctrl-single,pins = <
          0x154 0x01
          0x150 0x21
        >;
      };
```

```
      bb_uart4_pins: pinmux_bb_uart4_pins {
        pinctrl-single,pins = <
          0x070 0x26
          0x074 0x06
        >;
      };
      bb_i2c1_pins: pinmux_bb_i2c1_pins {
        pinctrl-single,pins = <
          0x158 0x72 /* i2c1_sda, SLEWCTRL_SLOW */
                     /*          INPUT_PULLUP */
                     /*          MODE2 */
          0x15c 0x72 /* i2c1_scl, SLEWCTRL_SLOW */
                     /*          INPUT_PULLUP */
                     /*          MODE2 */
        >;
      };
    };
};

fragment@1 {
    target = <&spi1>;
    __overlay__ {
      #address-cells     = <1>;
      #size-cells        = <0>;
      status             = "okay";
      pinctrl-names      = "default";
      pinctrl-0          = <&bb_spi1_pins>;
      cs-gpios           = <&gpio4 17 0>, <&gpio1 7 0>;

      channel@0{
        #address-cells     = <1>;
        #size-cells        = <0>;
        compatible         = "spidev";
        reg                = <0>;
        spi-max-frequency  = <1000000>;
      };
      channel@1{
        #address-cells     = <1>;
        #size-cells        = <0>;
        compatible         = "spidev";
        reg                = <1>;
        spi-max-frequency  = <1000000>;
      };
    };
};

fragment@2 {
    target = <&uart2>;  /* really uart1 */
    __overlay__ {
      status = "okay";
```

```
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart1_pins>;
      };
    };

    fragment@3 {
      target = <&uart3>;  /* really uart2 */
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart2_pins>;
      };
    };

    fragment@4 {
      target = <&i2c1>;
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_i2c1_pins>;

        /* this is the configuration part */
        clock-frequency = <100000>;
        #address-cells = <1>;
        #size-cells = <0>;
      };
    };

    fragment@5 {
      target = <&uart5>;       /* really uart4 */
      __overlay__ {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&bb_uart4_pins>;
      };
    };
};
```

This Device Tree overlay, stored as a .dts files, must be compiled before it can be used by the operating system. At the time of writing this technical report, the Device Tree compiler available in Ubuntu distributions does not support all the necessary features. Therefore, it is necessary to download and build a newer version. To upgrade to the newer version, execute the following commands.

```
wget -c https://raw.githubusercontent.com/RobertCNelson/tools/master/pkgs/dtc.sh
chmod +x dtc.sh ./dtc.sh
```

Once upgraded, the following command will compile the Device Tree overlay, where <*device_tree_overlay*> is the name given to the overlay file. An arbitrary name can be used if the overlay is to be loaded manually. However, i

```
dtc -O dtb -o <device_tree_overlay>.dtbo -b 0 -@ <device_tree_overlay>.dts
```

# B Specifications

Table 12 summarizes the overall specifications of the expansion board.

Table 12: Electrical and timing specifications of all the modules of the expansion board.

| Power Supply | | | |
|---|---|---|---|
| **Feature** | **Min** | **Typical** | **Max** |
| VCC_5 | | $5\,V$ | |
| VCC_5 current | $1\,A$ | | |
| VCC_15 | | $15\,V$ | |
| VCC_15 current | $10\,mA$ | | |
| VCC_-15 | | $-15\,V$ | |
| VCC_-15 current | $70\,mA$ | | |
| VCC_EXT | $12\,V$ | | $35\,V$ |
| Digital Outputs | | | |
| **Feature** | **Min** | **Typical** | **Max** |
| Number of ports | | | 8 |
| Output voltage | | | VCC_EXT |
| Output impedance | | $10\,\Omega$ | |
| Digital inputs | | | |
| **Feature** | **Min** | **Typical** | **Max** |
| Number of ports | | | 8 |
| Input voltage | | | $30\,V$ |
| Input impedance | | $1\,k\Omega$ | |
| Analog Outputs | | | |
| **Feature** | **Min** | **Typical** | **Max** |
| Number of ports | | | 8 |
| Bandwidth | | | $300\,kHz$ |
| Output voltage | $0\,V$ | | $13\,V$ |
| Analog inputs | | | |
| **Feature** | **Min** | **Typical** | **Max** |
| Number of ports | | | 6 |
| Bandwidth | | | $320\,kHz$ |
| Input voltage | $-12\,V$ | | $12\,V$ |
| Serial ports | | | |
| **Feature** | **Min** | **Typical** | **Max** |
| Number of ports | | | 3 |
| Baudrate | | | $1\,Mbps$ |

# C Gripper XML Configuration file

Listing 11 shows the XML template used to configure the CMEG50ecGripper driver. This file has a common part *devices_config_t* which provides information on the $I^2C$ and SPI Linux devices, and also the Beaglebone Black GPIO signals to handle the hardware modules. The *gripper_config_t* data structure has all the information required to configure each gripper (left and right), and a boolean value specifies whether the configuration for each gripper should be loaded or not.

Listing 11: Gripper XML configuration file template

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="port_t">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="GPIO0"/>
        <xsd:enumeration value="GPIO1"/>
        <xsd:enumeration value="GPIO2"/>
        <xsd:enumeration value="GPIO3"/>
     </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="port_exp_t">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="PORTA"/>
        <xsd:enumeration value="PORTB"/>
     </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="digital_pot_t">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="DPOT1"/>
        <xsd:enumeration value="DPOT2"/>
        <xsd:enumeration value="DPOT3"/>
        <xsd:enumeration value="DPOT4"/>
     </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="dac_t">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="DAC_A"/>
        <xsd:enumeration value="DAC_B"/>
        <xsd:enumeration value="DAC_C"/>
        <xsd:enumeration value="DAC_D"/>
        <xsd:enumeration value="DAC_E"/>
        <xsd:enumeration value="DAC_F"/>
        <xsd:enumeration value="DAC_G"/>
        <xsd:enumeration value="DAC_H"/>
     </xsd:restriction>
```

```
</xsd:simpleType>

<xsd:simpleType name="adc_t">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ADC_A0"/>
      <xsd:enumeration value="ADC_A1"/>
      <xsd:enumeration value="ADC_B0"/>
      <xsd:enumeration value="ADC_B1"/>
      <xsd:enumeration value="ADC_C0"/>
      <xsd:enumeration value="ADC_C1"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="gpio_t">
  <xsd:sequence>
    <xsd:element name="port" type="port_t">
    </xsd:element>
    <xsd:element name="pin" type="xsd:unsignedByte">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="gpio_exp_t">
  <xsd:sequence>
    <xsd:element name="port" type="port_exp_t">
    </xsd:element>
    <xsd:element name="pin" type="xsd:unsignedByte">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="pot_t">
  <xsd:sequence>
    <xsd:element name="pot" type="digital_pot_t">
    </xsd:element>
    <xsd:element name="device" type="xsd:unsignedByte">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="i2c_dev_t">
  <xsd:sequence>
    <xsd:element name="device" type="xsd:string">
    </xsd:element>
    <xsd:element name="address" type="xsd:unsignedByte">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

  <xsd:complexType name="gripper_config_t">
```

```
    <xsd:sequence>
      <xsd:element name="stopped" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="ref_motion_done" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="target_pos_reached" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="reset" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="ref_motion" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="open" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="close" type="gpio_exp_t">
      </xsd:element>
      <xsd:element name="speed_pot" type="pot_t">
      </xsd:element>
      <xsd:element name="force_pot" type="pot_t">
      </xsd:element>
      <xsd:element name="position_pot" type="pot_t">
      </xsd:element>
      <xsd:element name="speed_dac" type="dac_t">
      </xsd:element>
      <xsd:element name="force_dac" type="dac_t">
      </xsd:element>
      <xsd:element name="position_dac" type="dac_t">
      </xsd:element>
      <xsd:element name="current_pos_adc" type="adc_t">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="devices_config_t">
    <xsd:sequence>
      <xsd:element name="gpio_exp" type="i2c_dev_t">
      </xsd:element>
      <xsd:element name="gpio_exp_int_a" type="gpio_t">
      </xsd:element>
      <xsd:element name="gpio_exp_int_b" type="gpio_t">
      </xsd:element>
      <xsd:element name="digital_pot1" type="i2c_dev_t">
      </xsd:element>
      <xsd:element name="digital_pot2" type="i2c_dev_t">
      </xsd:element>
      <xsd:element name="dac" type="xsd:string">
      </xsd:element>
      <xsd:element name="adc" type="xsd:string">
      </xsd:element>
      <xsd:element name="adc_start" type="gpio_t">
      </xsd:element>
```

```
      <xsd:element name="adc_busy" type="gpio_t">
      </xsd:element>
      <xsd:element name="heartbeat" type="gpio_t">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="meg50ec_bbb_config_t">
    <xsd:sequence>
      <xsd:element name="devices" type="devices_config_t">
      </xsd:element>
      <xsd:element name="left_gripper_enabled" type="xsd:boolean">
      </xsd:element>
      <xsd:element name="left_config" type="gripper_config_t">
      </xsd:element>
      <xsd:element name="right_gripper_enabled" type="xsd:boolean">
      </xsd:element>
      <xsd:element name="right_config" type="gripper_config_t">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="meg50ec_bbb_config" type="meg50ec_bbb_config_t">
  </xsd:element>

</xsd:schema>
```

# D Monitor XML Configuration file

Listing 12 shows the XML template used to configure the CMonitor driver.

Listing 12: Monitor XML configuration file template

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="port_t">
     <xsd:restriction base="xsd:string">
        <xsd:enumeration value="GPIO0"/>
        <xsd:enumeration value="GPIO1"/>
        <xsd:enumeration value="GPIO2"/>
        <xsd:enumeration value="GPIO3"/>
     </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="gpio_t">
    <xsd:sequence>
      <xsd:element name="port" type="port_t">
      </xsd:element>
```

```
      <xsd:element name="pin" type="xsd:unsignedByte">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="monitor_bbb_config_t">
    <xsd:sequence>
      <xsd:element name="cell_chain" type="gpio_t">
      </xsd:element>
      <xsd:element name="cell_emergency_stop" type="gpio_t">
      </xsd:element>
      <xsd:element name="light_curtain" type="gpio_t">
      </xsd:element>
      <xsd:element name="left_mechanical_fuse" type="gpio_t">
      </xsd:element>
      <xsd:element name="right_mechanical_fuse" type="gpio_t">
      </xsd:element>
      <xsd:element name="operation_mode" type="gpio_t">
      </xsd:element>
      <xsd:element name="robot_xy_chain" type="gpio_t">
      </xsd:element>
      <xsd:element name="left_staubli_chain" type="gpio_t">
      </xsd:element>
      <xsd:element name="right_staubli_chain" type="gpio_t">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="monitor_bbb_config" type="monitor_bbb_config_t">
  </xsd:element>

</xsd:schema>
```

## References


[1] Device tree wiki page. `http://www.devicetree.org/Main_Page`, Janaury 2014.

[2] beagleboard.org. Beaglebone black manual. `https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true`, May 2015.

[3] beagleboard.org. Beaglebone black porduct webpage. `http://beagleboard.org/BLACK`, March 2015.

[4] Frank Mori Hess and Frank Mori Hess. Linux comedi project webpage. `http://www.comedi.org/`.

[5] IRI. Cgpio class reference documentation. `http://devel.iri.upc.edu/docs/labrobotica/drivers/meg50ec_gripper_bbb`, March 2015.

[6] Sergi Hernandez Juan. Staübli work-cell: General description and operation. Technical report, IRI, 2015.

[7] ros.org. Main ros webpage. http://www.ros.org/.

[8] SCHUNK. Meg50ec gripper manual. http://www.schunk.com/schunk_files/attachments/OM_AU_MEG50-EC__EN.pdf.

[9] Wikipedia. I2c wikipedia page. http://en.wikipedia.org/wiki/I%C2%B2C, March 2015.

[10] Wikipedia. Spi wikipedia page. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus, March 2015.

## IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.