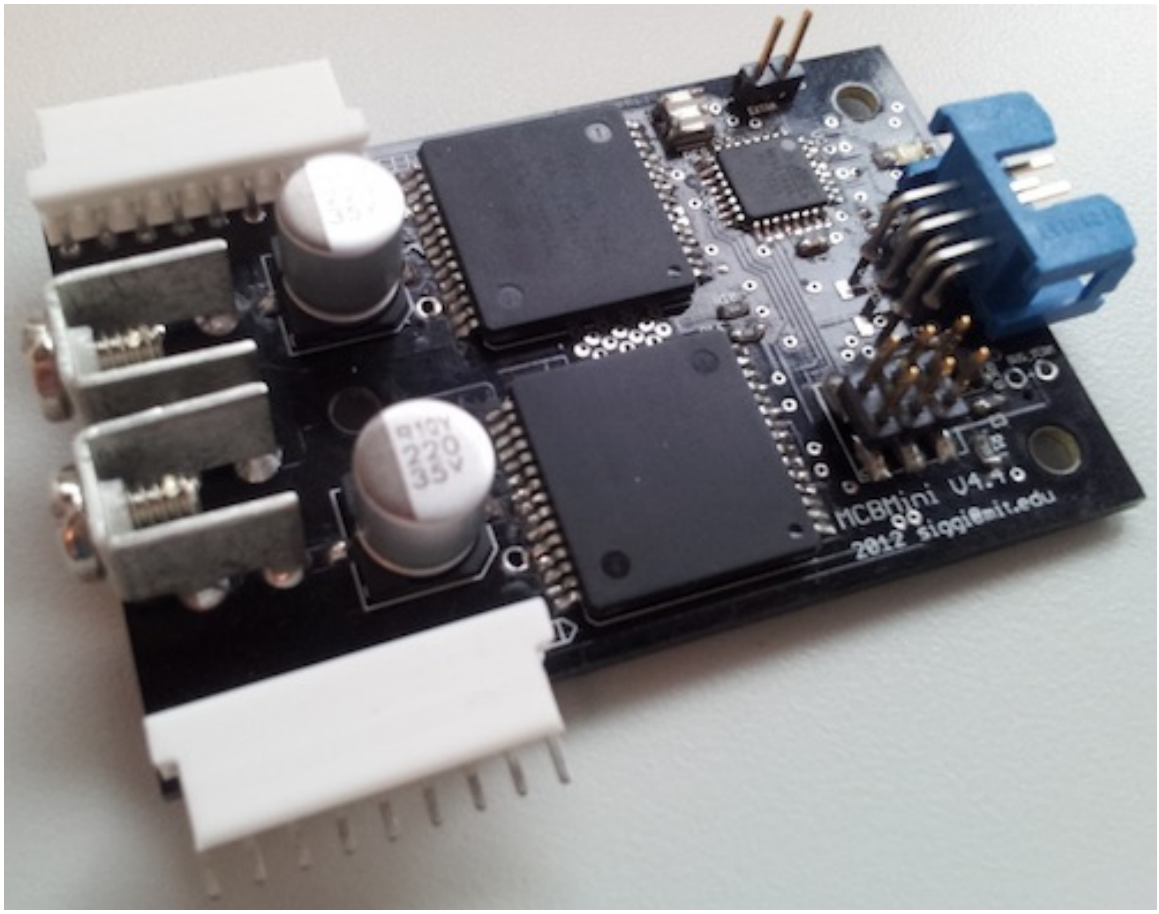
MCBMini V4.4
Manual



January 28th, 2013

siggi@mit.edu

Table of Contents

Features and Specifications

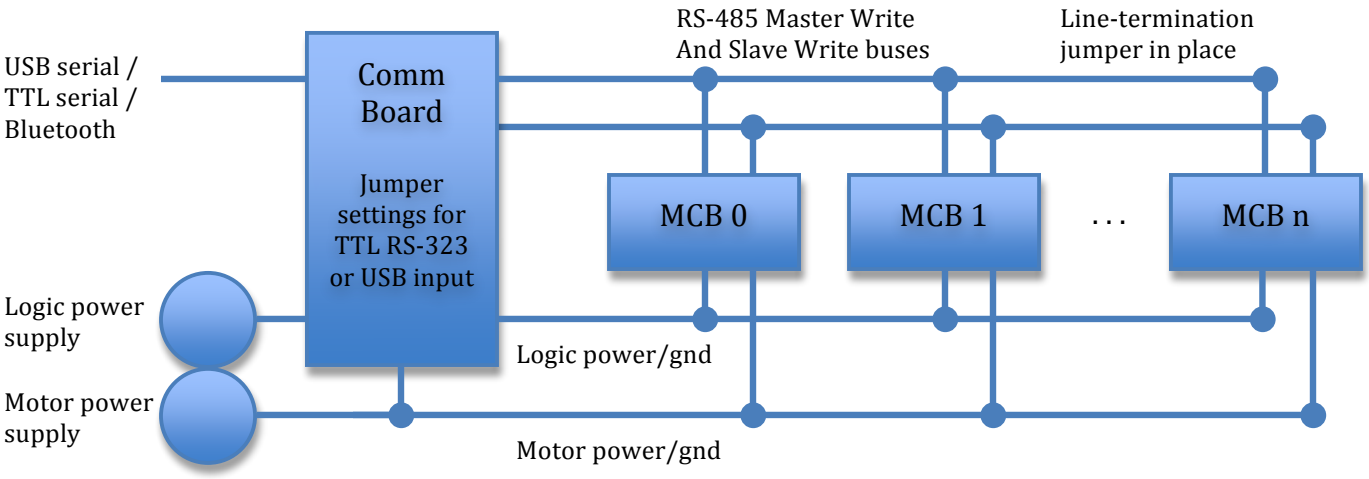
Maximum Specifications

Name	Minimum	Maximum
V_{in} (logic supply)	6 V	24 V
V_{cc} (power supply)	5.5 V	24 V
I_o (single channel current)		30 A continuous *

* This is the maximum output current of the VNH5019A bridge, to run at this current some serious heat considerations need to be made. NOTE The traces and connectors have not been tested at this high current.

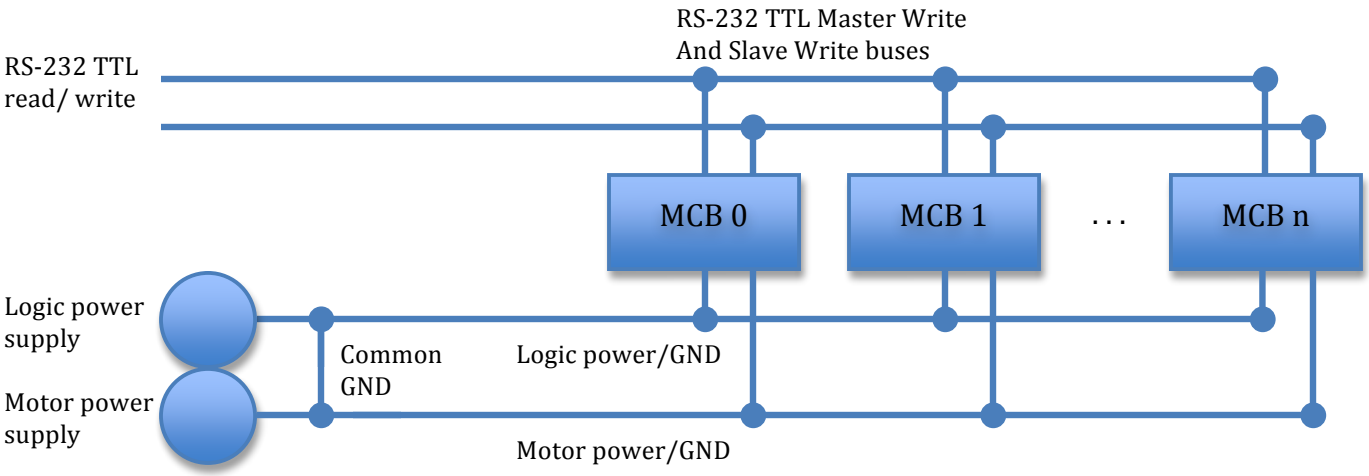
System Overview

RS-485 Mode (with communication board)



In this configuration, the logic and power grounds are only commonly connected through one connection on the communication board to avoid ground loops. The communication board also allows the motor power to be forwarded as logic power to the MCBs via jumper settings. This eliminates the need for two separate power supplies but also runs the risk of having "dirty" power (motors can generate quite a bit of noise) fed to the logic and microcontrollers.

RS-232 Mode (with hardware modification on MCBs)



In this configuration the MCB boards have had a slight hardware modification that changes their bus from being RS-485 differential levels to regular TTL level RS-232 signals. In this mode there are pull-up resistors (within the microcontrollers) enabled on all the slave bus outputs. This might not scale for many boards (>6-8) and using the communication board is recommended.
In this mode the two supplies need to have their grounds manually connected in one place as shown in the figure.

Software Features

MCBMini is a system of RS-485 bus connected two-channel motor controllers. Some of the general features of the motor controller firmware are:

- **Update frequencies**
  - The boards perform their PID update at 100 Hz. This update rate is used to estimate the velocity by taking the difference in the current tick position from the one a certain number of updates ago (specified with the VEL_TIME_DELTA parameter).
  - Their PWM frequency 20kHz
  - The maximum encoder tick frequency is approximately 16.5 kHz if only one channel is using encoder feedback and 12.5 kHz if both are using it simultaneously. This is the frequency of the pulse train on either phase of the encoder
  - All analog values (potentiometer feedback and current sensing) are averaged over several measurements before used in the PID loop or reported on request

- **Control features**

- Position, velocity or mixed mode operation. In position and mixed modes, targets are provided in absolute tick positions. In velocity mode the target is interpreted as the desired difference in actual ticks between a certain number of control updates (specified by the VEL_TIME_DELTA parameter). In mixed mode, the position controller sits "on top of" a velocity controller (the position controller generates target velocities which the velocity controller uses to produce PWM signals).
- In velocity and mixed control modes, a maximum velocity and maximum acceleration can be enforced`
- A streaming mode can be enabled where the controller maintains a look-ahead buffer of target positions that it attempts to "intelligently" navigate smoothly towards.
- A maximum PWM duty cycle can be asserted which allows guaranteeing that a motor is ever only exposed to a portion of the battery voltage.
- Two extra pins on the board can be set to either serve as analog signal readers, switches or analog servo motor controllers

- **Safety features**
  - A communication timeout of approximately half a second will disable both motor channels and notify host on next communication cycle. This effectively makes the boards disable their channels if the host malfunctions or is shut down without the proper shutdown routine of disabling motors
  - A short circuit on any of the motor outputs will generate an error condition, disable the corresponding motor channel and report the event to the host
  - An overheating condition on either of the motor channels will disable the corresponding channel and report the event to the host
  - Reverse battery protection is implemented with an N-channel Mosfet

- **Error reporting**
  - The boards will notify the host when they receive their first packet from startup/reset. This will prompt the host to send parameters to the board in case of a fault reset or initial startup
  - If the boards ever receive a packet with a bad checksum they will discard it but notify the host on the next communication cycle of the event
  - If the boards ever have a buffer overflow (probably caused by error in transmission) then they will discard the message and notify the host of the error on the next successful communication cycle
  - A communication timeout will disable both motor channels. If communication is ever restored, the boards will notify host of the event

- If the boards ever receive a command that is not recognized they will report the event to the host
- A fault condition on either channel (short circuit or overheating) will disable the corresponding channel and generate a fault message

Hardware Features

The H-bridge (ST VNH5019A-E) IC that the MCBMini design uses has various hardware features that can be further explored in the datasheet:
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00234623.pdf

Encoder Resolution Limitation

The boards have a limitation on how frequent the encoder feedback pulses can be for accurately tracking them. This limitation is imposed by the fact that the microprocessor has a lot to do (handle comm. and calculate PID values etc.) and can't be bothered too frequently by the encoder external pin interrupts.

During testing we have found out that the maximum frequency of the pulse train on an encoder phase is **16.5 kHz** if only one channel uses quadrature encoder input and **12.5 kHz** if both do.

This value can be calculated from three known values: encoder resolution, gear ratio of motor (if encoder is not on the gear output shaft but on the actual motor shaft) and maximum output shaft velocity.

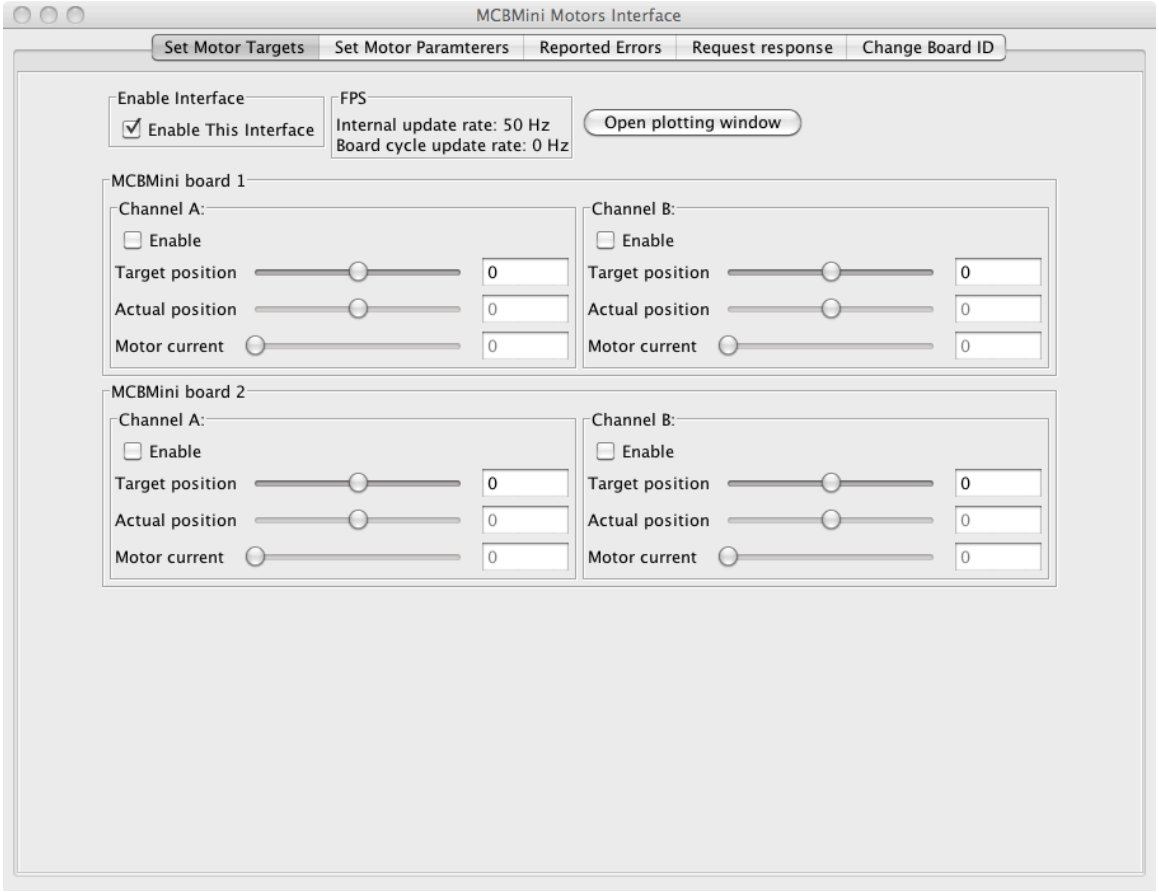
Example: A motor has a maximum velocity of 360 RPM of the output shaft, a 1:30 gear ratio and encoder resolution of 64:

Tick frequency = 30 * 64 * 360 / 60 = 11.5 kHz (note we divide by 60 because it is 360 Revolutions Per Minute)

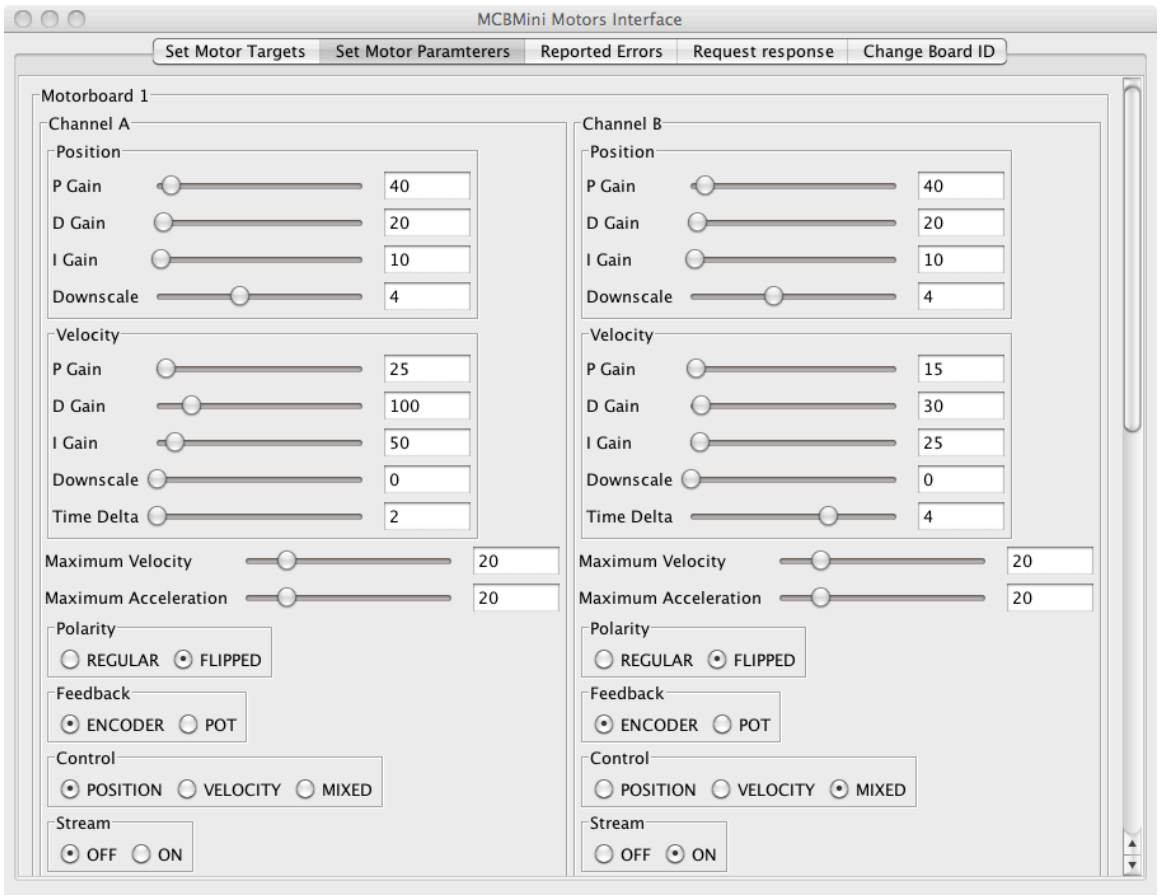
So this is probably a suitable motor/encoder combination to have on both channels without problems. Also note that 360 RPM is maximum speed so mostly the operating speed will be lower.

Software

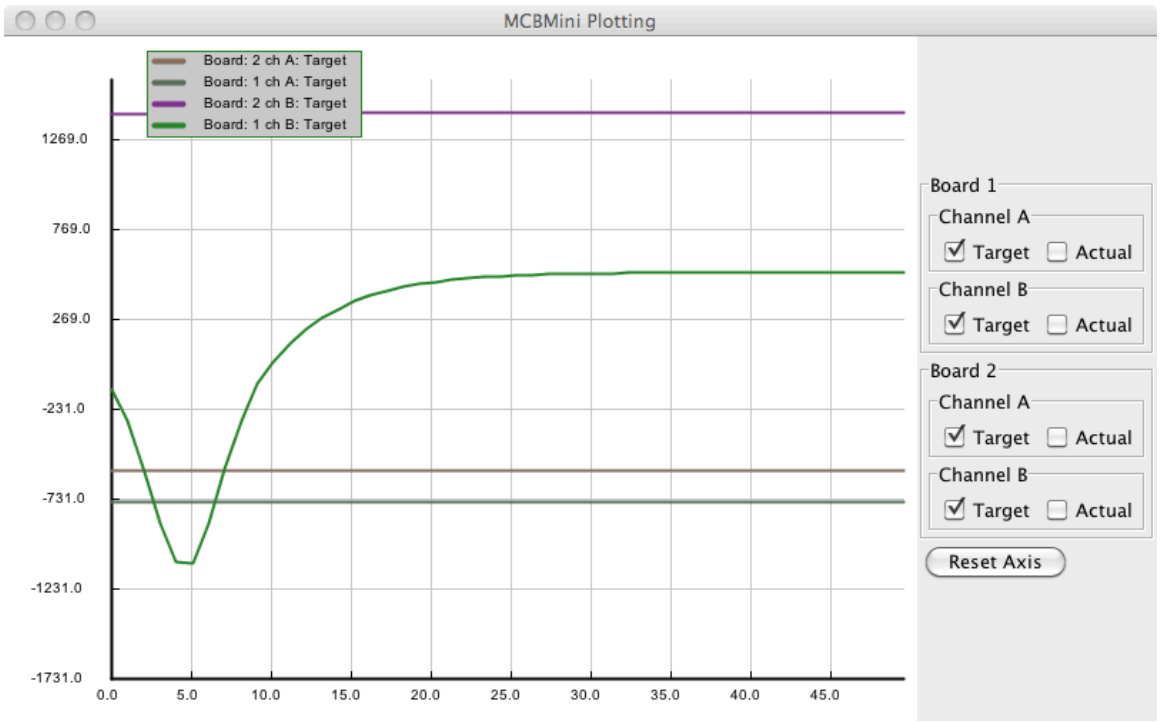
This section needs more work but for now just has screenshots of the GUI.



This is the main motor target window where a user can specify motor target positions or velocities and observe instantaneous feedback values.



In this window the user can specify various parameters for the boards and change them on the fly. These parameters are read from a configuration XML file.



In this window the user can observe target and actual values for all the controllers. This is often crucial to tune the controllers for different setups.

Parameters and Settings

The parameters of every controller are provided in a configuration XML file. The skeleton of the file is provided here and the specific available channel parameters are explained in the following sections.

Skeleton Configuration File

```
<Root>
        <port>/dev/tty.usbserial-AH016UG3</port>
        <firmware_version>16</firmware_version>
        <MiniBoards>
                <MiniBoard>
                        <id>1</id>
                        <Channels>
                                <Motor>
                                        <channel>A</channel>
                                        ...parameters...
                                </Motor>
                                <Motor>
                                        <channel>B</channel>
                                        ...parameters...
                                </Motor>
                        </Channels>
                </MiniBoard>
```

```
        </MiniBoards>
</Root>
```

Channel Parameters

Every channel parameter has a name, a command ID byte, and a value. The value is either a continuous integer and has a range or a named enumerated constant.

Name	ID	Range	Description
pos_p_gain	1	X > 0	The proportional gain for the position PID controller
pos_d_gain	2	X > 0	The differential gain for the position PID controller
pos_i_gain	3	X > 0	The integral gain for the position PID controller
pos_downscale	4	X > 0	This is an output divisor which provides a larger range in the PID constants: output = pid_output / 2^X
vel_p_gain	36	X > 0	The proportional gain for the velocity PID controller
vel_d_gain	37	X > 0	The differential gain for the velocity PID controller
vel_i_gain	38	X > 0	The integral gain for the velocity PID controller
vel_downscale	39	X > 0	This is an output divisor which provides a larger range in the PID constants: output = pid_output / 2^X
vel_time_delta	41	X > 1	The velocity is calculated as the difference between the actual position at the current update cycle and at the update cycle delta updates earlier. This can be used to
max_velocity	4	X > 0	Only used in the velocity and mixed control modes. Sets the maximum velocity of the controller, measured in the tick difference between updates
max_acceleration	40	X > 0	Only used in the velocity and mixed control modes. Sets the maximum acceleration of the controller, measured in increments/decrements in tick difference between updates
slow_enable_const	26	X > 0	On enable, the PID output ramped up over a period of X * 32 * duration_of_update
polarity	7	Enum	See MotorPolarity enumerator description below
feedback	8	Enum	See FeedbackMode enumerator description below
control	9	Enum	See ControlMode enumerator description below
extra_pin	32	Enum	See ExtraPinMode enumerator description below
stream	42	Enum	See StreamMode enumerator description below

Enumerator Value Definitions

These define the values of the parameters that take enumerated constant values.

MotorPolarity	ID	Description
REGULAR	0	Motor phases have same polarity as the quadrature encoder phases
FLIPPED	1	Motor phases have opposite polarity ot the quadrature encoder phases

FeedbackMode	ID	Description
ENCODER	0	A quadrature encoder is used for control feedback
POT	1	A potentiometer is used for control feedback

ControlMode	ID	Description
POSITION	0	A regular feedback position PID controller
VELOCITY	1	A velocity feedback controller and uses the velocity PID constants as well as the time_step_delta parameter to calculate the actual velocity in a difference

		of tick position. Is subject to maximum velocity and acceleration
MIXED	2	Operates as a position PID controller "on top of" the velocity controller used in velocity mode. Targets are set in absolute positions but this allows velocity and acceleration limits to be applied

ExtraPinMode	ID	Description
OFF	0	No extra mode
SWITCH	1	The extra pin now serves as a switch with a pull-up resistor. If it gets pulled to GND, an EXTRA_VALUE message gets sent to the server.
ANALOG	2	The extra pin is used to sense an analog voltage signal. There are resistor pads on the board to allow this to be hardwired to the logic and motor batteries to monitor their voltage levels.
SERVO	3	The extra pin can now be used to control a servo motor. The signal should be provided with EXTRA_SERVO messages.

StreamMode	ID	Description
OFF	0	Stream mode is off
ON	1	Target positions get buffered so the controller can look at the next three target values. It can then "intelligently" decide how strictly to commit to the next target value vs. choose the next one after once it passes the first one.

Communication

Synchronous / Asynchronous

We can also communicate with the MCBs in asynchronous mode where the master just sends commands to each MCB without regard to the responses. The only risk is that the master might have sent a message to MCB n+1 before MCB n is done responding so now we have two MCBs using the Slave bus. A solution to this problem is to limit the longest reply an MCB could ever give to be shorter than the shortest package ever sent by the host. This means that in the time it takes the host to send any message to MCB n+1, MCB n would have finished sending its response, leaving the bus free for MCB n+1 to respond.

This can be simply implemented on the host side by checking every message as it is going out and making sure that it is at least of size M (size of longest reply an MCB could give, plus overhead) and if not, padding it with zeros (at the end of the buffer so they won't affect the message or the checksum). At 115000 baud rate, it has been found that if all host messages are at least 20 bytes then there will be no contention on the slave bus in asynchronous mode.

RS-485 vs. RS-232

The MCBs board layout assumes two RS-485 line drivers that communicate with the host via the MCBMini Comm board. When the microcontroller on the MCB wants to transmit a message, it **enables** the write RS-485 driver and then transmits the message. When it is done transmitting, it **disables** the driver but it also turns off its TX pin and sets it into Hi-Z state. This means that if the TX pin were directly wired to

the bus, bypassing the RS-485 drivers, it would still be able to share a bus with multiple slaves.

It is recommended that the MCBs be used using the RS-485 drivers and the MCBMini Comm board but it is also possible through a slight hardware modification to fully bypass RS-485 and control the bus using TTL level serial commands (either from another microcontroller, Arduino or FTDI board for example). This might be handy if a minimal implementation of the motor control system is required and not a lot of noise on the bus is expected (RS-485 will be much more robust than TTL level RS-232).

The only hardware modification needed is to simply add jumpers (zero ohm resistors or just solderbridges) to resistor parts JR0, JR1, JR2 and JR3. No software modification is needed for this feature to work, the firmware already supports both modes of operation. Now the slave TX line is connected to the SLAVE_WR_A pin on the comm. connector and the slave RX line is connected to the MASTER_WR_A pin.

Serial Synchronization

The method we use to synchronize data on the bus is to define two **special** bytes, a HEADER (0xAA) and an ESCAPE (0x55). When we want to send a stream of bytes over the serial link, we start by iterating through the transmission bytes, if we ever see HEADER or ESCAPE in the data, then we insert the ESCAPE character in front of that byte in the stream and then convert that byte by XORing it with 1. When we want to send the stream to an MCB we add the HEADER at the end of the package.

This guarantees that whenever the MCB receives a HEADER, it has received a whole package. The MCB just has to make sure that whenever it sees an ESCAPE byte, not to keep it but to take the next byte in the stream and apply the XOR-with-1 transformation on it to revert it to its original state.

Our convention is to only calculate checksums using the data **before** transformation, that means that the checksum knows nothing of inserted ESCAPE bytes or transformations.

Package Description

Serial packets are oriented such that received zeros on the bus before the transmission of any packets will not affect the parsing of a successful packet.

The host can set an R bit in the CMD byte which will mean that we don't mean to send the value for that particular CMD, rather we are requesting a response with the current value corresponding to that CMD. If the MCB sets the R bit in it's response to the host, it is communicating that it has a message waiting to be polled by the host.

MCB IDs can range from 0 to 125. ID 126 is reserved for boards that have never been given an ID (have just been programmed) and ID 127 is reserved for broadcast.

Byte Name	Description	Sequence
Zeros	An arbitrary number of padded zeros can precede any packet without affecting its reception or parsing	first
Byte data	Packet can contain any length of data bytes or integers, depending on what is expected given the CMD	
Int >> 0	Integers are sent with low byte first	
Int >> 8		
Int >> 16		
Int >> 24		
R | CMD	First bit is the request response bit, the rest are CMD bits	
M | ID	First bit is the motor channel bit, last seven bits are the target/source board ID	
CHCKSUM	Checksum of everything in packet up to this byte (not including) modulo 256	
HEADER	For packetizing	last

Command Bytes

These are the command bytes that have not already been listed under the Channel Parameter section. Most of these parameters can either be set or requested through the R bit as described in the packet description above.

ID	CMD Name	Data Size	Description
0	ID	Byte*	The identification number of the MCB. 126 means the board is new and hasn't received an ID before, 127 is the broadcast ID
6	ENABLE	Byte	Enables the output of the channel
10	TARGET_TICK	Integer	The target value (position or velocity depending on the control mode) of the PID controller
11	ACTUAL_TICK	Integer	The actual value of the feedback controller. These are encoder or potentiometer ticks or velocity depending on control mode
12	MOTOR_CURRENT	Integer	The analog motor electrical current signal
22	ENCODER_VALUE	Integer	The current quadrature encoder tick counter
23	POT_VALUE	Integer	The current analog potentiometer value
35	ACTUAL_VEL	Integer	The difference in tick values now and VEL_TIME_DELTA updates ago
28	ACTUAL_ENCODER_OFFS	Integer	This can be used to add a constant value to the current encoder tick counter
17	EMPTY_RESPONSE	Integer	If an MCB was not asked for a response and has no message in the buffer, it will reply with an EMPTY message
18	ERROR	Byte	This is an error message that contains an ERROR code which are listed in a table below
19	PID_OUTPUT	Integer	The current control signal being applied as a PWM value
24	FIRMWARE_VERSION	Integer	The firmware version of the board
25	MAX_PWM_DUTY_CYCLE	Integer	The maximum PWM duty cycle (1100 is 100%)
27	DEBUG	Integer	A way for the boards to report a debug value to the host (only used in development)
29	SATURATION	Byte	If the PID controller is at either its low or high limits, it will not add to its error integrator
30	I_COMPONENT	Integer	The current integrative component of the PID controller
31	REQUEST_MESSAGE	None	This is the message that a host sends to an MCB once it has indicated through the R bit that it has messages
33	EXTRA_PIN_VALUE	Integer	This is how the analog or switch (1/0) value of the extra pin is reported. This is also how to specify the servo position value

* When requesting the ID, usually the packet would be addressed to the broadcast ID (127) only with one MCB on the bus. When setting the ID, one needs to put the following bytes into the packet: [id+10, id, 3, 2, 1]. This is for added packet security, as the ID gets written to eeprom and persisted.

The following commands are specifically made for fast and efficient transmission of streaming target positions and receiving actual values back. These commands set the target values of both channels in one command and request a feedback value for the specified channel

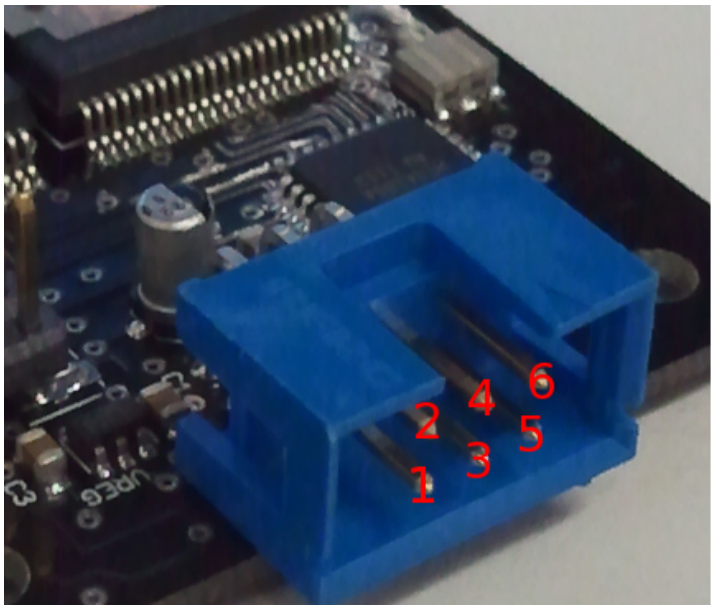
ID	CMD Name	Description
13	TWO_TARGET_TICK_ MOTOR_CURRENT	Sets the target values of both channels and requests the analog electrical current feedback value for one channel
13	TWO_TARGET_TICK_ ACTUAL	Sets the target values of both channels and requests the actual position tick value for one channel (either encoder or potentiometer)
34	TWO_TARGET_TICK_ VELOCITY	Sets the target values of both channels and requests the actual velocity signal of one channel

Error Codes

ID	Error Name	Data	Description
0	BAD_CHECKSUM	None	MCB received a bad checksum on last comm. cycle.
1	BAD_CMD_RECEIVED	Byte	MCB received an unknown CMD on this comm. cycle. Data contains the unknown CMD byte
2	UNINITIALIZED	None	MCB reports that it is receiving a target position but has never received parameters (PID constants etc.) from host.
3	BUF_OVF	None	MCB reports that on the last comm. cycle an RX buffer overflow happened.
4	TIMEOUT_DISABLE	None	MCB reports that it had not received comm. from the host for more than a second so it disabled both channels
5	FAULT	None	Overheating or short circuit on either channel
6	BAD_ID_RCV	None	An ID packet was received that doesn't contain the correct redundancy structure
7	PACKAGE_OVF	None	An overflow of package buffers has occurred
9	PARAM_DUR_EN	None	Board has received parameters while enabled
10	MSG_BUF_OVF	None	The board's message buffer has been overrun. The server is not polling it's messages as fast as it is generating them

Hardware

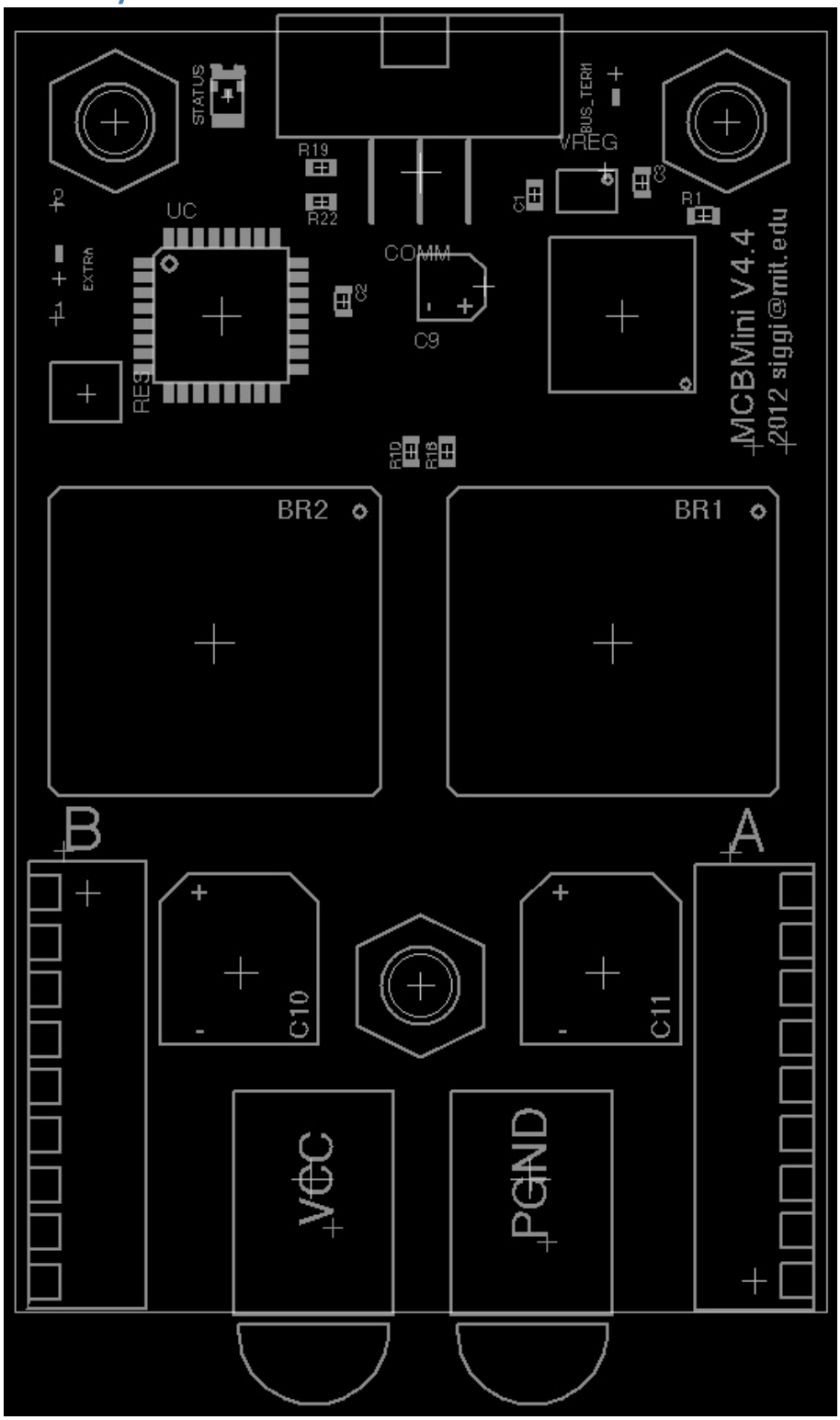
Communication Header Pin-Out



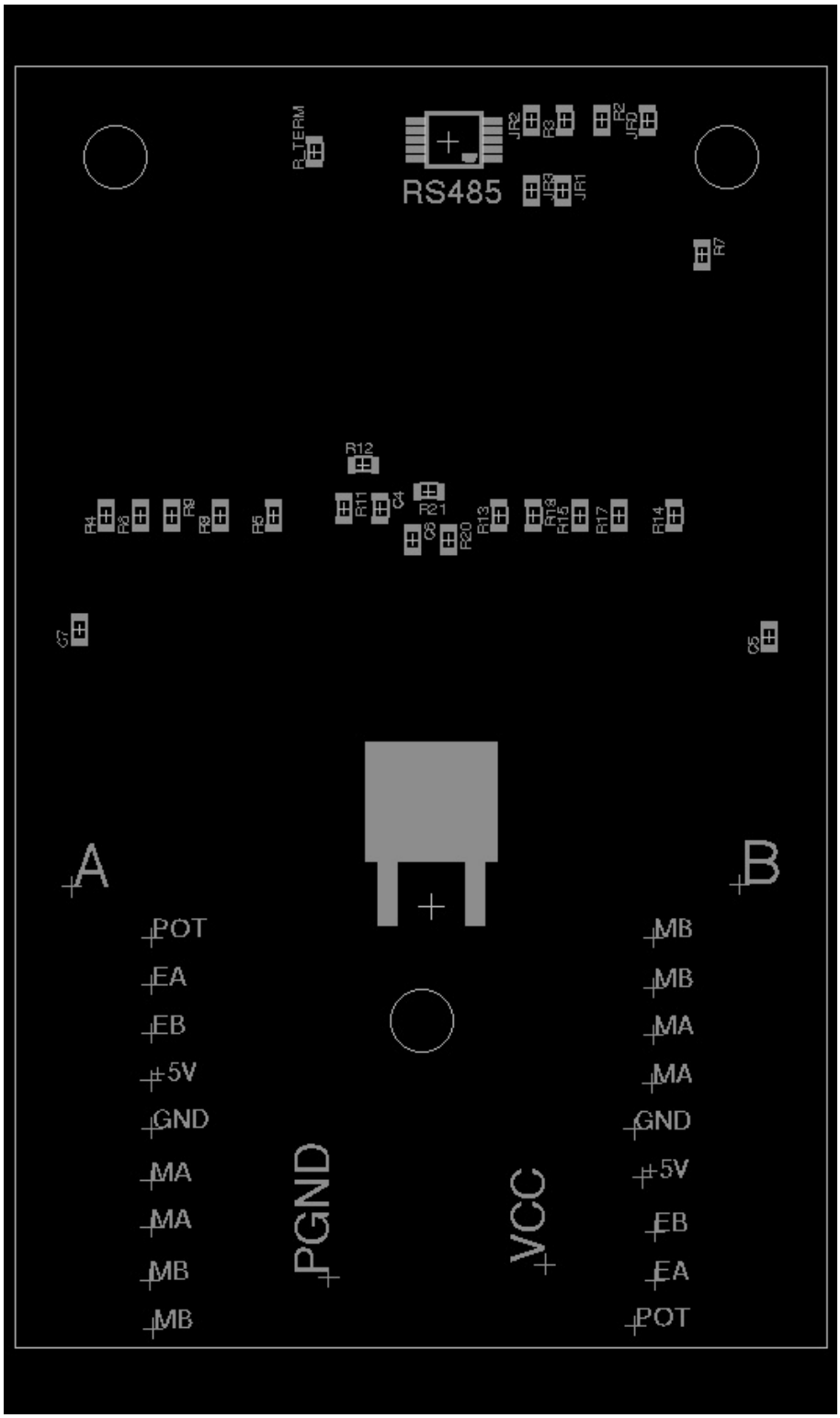
1 Vin
2 GND
3 SLAVE WR A
4 SLAVE WR B
5 MASTER WR A
6 MASTER WR B

Pin number refers to position on ribbon cable

Board Layout



Front side



Back side (actual view, not mirrored)

MCBMini Bill of Materials

Design Reference	Value	Package	Digikey	Count
Resistors				
R_TERM	100R	"0603"	P100GCT-ND	1
R1,R11,R20	10k	"0603"	P10KGCT-ND	3
R12,R21	1k5	"0603"	P1.5KGCT-ND	2
R2,R3,R7,R9,R10,R18,R19	3k3	"0603"	P3.3KGCT-ND	7
R4,R5,R6,R8,R13,R14,R15, R16,R17,R22	1k	"0603"	P1.0KGCT-ND	10
JR0,JR1,JR2,JR3	0R	"0603"	P0.0GCT-ND	4
ICs				
RS485		MSOP-10	296-21412-1-ND	1
UC		TQFP32-08	ATMEGA168A-AU-ND	1
BR1,BR2	VNH5019A	MULTIPOWE RSO-30	497-13073-1-ND	2
VREG		SOT-23-5	576-2785-1-ND	1
Misc. Discrete				
FET	FDD8896	TO-252	FDD8896CT-ND	1
STATUS	LED	"1206"	754-1439-1-ND	1
RES		AWSCR-CV	535-10011-1-ND	1
Board Connectors				
COMM			609-2846-ND	1
MCON_A, MCON_B			A19483-ND	2
POWER_GND, VCC			7700K-ND	2
PROGRAMMING			WM17457-ND	1
BUS_TERM, EXTRA			A26509-40-ND	0.1
Capacitors				
C1	1uF @50V	"0603"	587-2400-1-ND	1
C10,C11	220uF	HAO	565-2478-1-ND	2
C3	2.2uF @10V	"0603"	587-1253-1-ND	1
C4,C6	33nF @50V	"0603"	445-5084-1-ND	2
C2,C5,C7	0.1uF @50V	"0603"	311-1343-1-ND	3
C9	22uF	alum	PCE3865CT-ND	1
Non-Board Components				
(COMM-MATE)			609-2841-ND	1
(MCON-MATE)			A30993-ND	2
Jumper	cap		S9001-ND	1
plastic nut	_4-40		H616-ND	1
plastic standoff	_4-40		4799K-ND	3