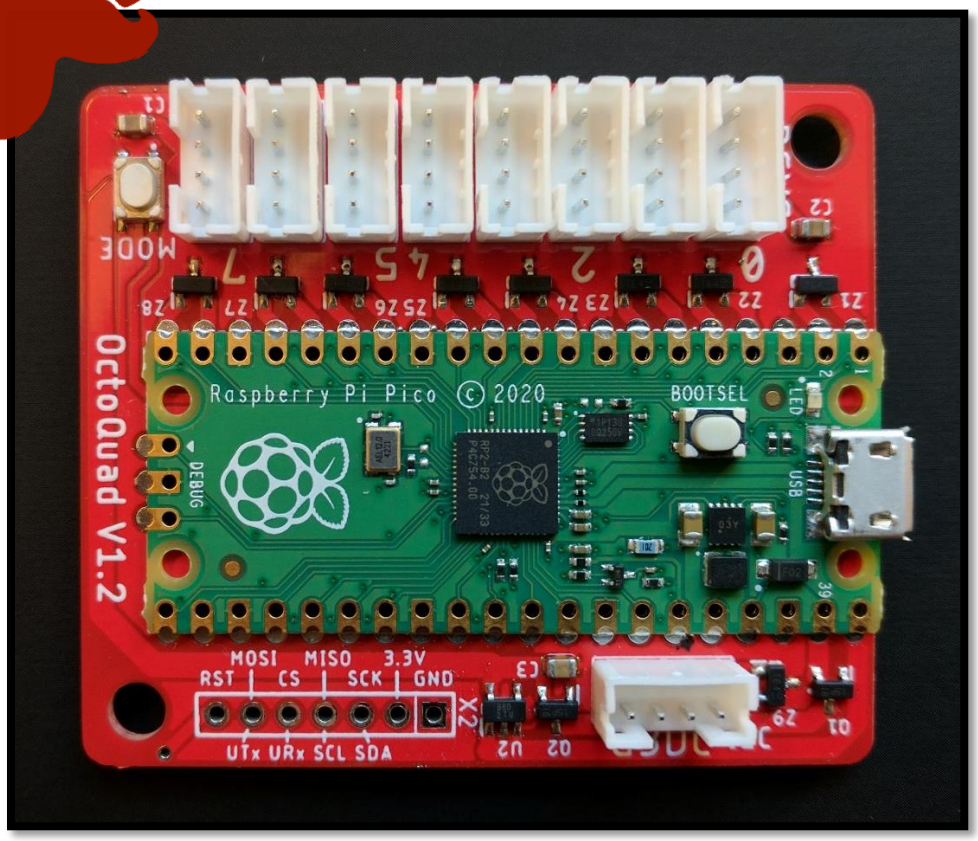
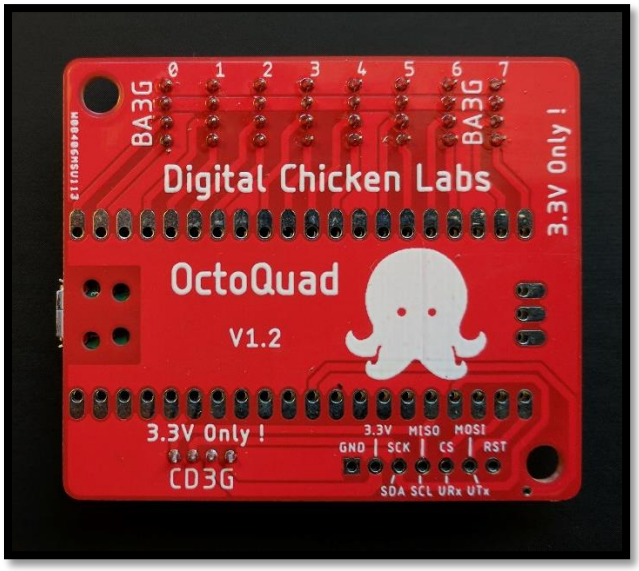
OctoQuad Specification





Spec. Rev. 2.0.4 – 9/1/2022

Table of Contents

1 Introduction

1.1 Overview

This document describes the operation of the OctoQuad module and the programming interface.

1.2 Description

The **OctoQuad** provides a means to read up to eight **quadrature encoders** or **absolute pulse width encoders** on systems where decoding the signals directly using GPIO pins is not feasible. The OctoQuad supports four different interfaces: I2C, SPI, UART, and USB.

For quadrature encoders, counts are tracked using signed 32-bit integers, and the counts for each encoder are individually resettable. Additionally, the velocity of each encoder is tracked using a signed 16-bit integer which represents the delta counts during a user-configurable sampling interval.

For pulse width measurement, pulse width is measured in microseconds and is reported as a 32-bit integer. Velocity is measured as the change in microsecond pulse width during the user configurable sampling interval and reported as a signed 16-bit integer. The velocity calculation requires user-specified minimum and maximum pulse width values.

1.3 Supported Firmware Version

This document supports firmware version 2.0.x

2 Electrical Specifications

2.1 Logic Level

The OctoQuad module uses 3.3v logic and power for SPI/I2C/UART bus communications, as well as 3.3v power and logic for quadrature encoder signals. **The I2C/SPI/UART and encoder connections are *NOT* 5v tolerant!**

2.2 Power

The OctoQuad module may be powered either via USB or via the 3.3v pin on the I2C/SPI/UART bus connection**. If powered via USB, the combined current draw from all 8 encoder ports must not exceed 300ma.** If power is provided to the 3.3v pin and USB is also connected, power will be drawn from the USB host.

2.3 Quadrature signal input

The step rate should not exceed 1 million steps/sec on any individual port (higher rates may work, but have not been tested). Note that the OctoQuad does NOT provide pull-up resistors for the A/B quadrature channels.

2.4 Pulse width signal input

The OctoQuad can measure pulse width signals from 1µs to 65535µs

2.5 ESD Protection

The encoder channels and I2C lines are protected from ESD to +/- 15kV (air)

3 Pinouts

3.1 4-pin JST-PH Encoder Channel Connectors

Each encoder channel connector provides power and A/B quadrature or pulse width input signal connections.

PCB Pin Label	Function
G	Ground
3	3.3v power supply for encoder
A	Quadrature channel A **OR** Pulse Width Input
B	Quadrature channel B

3.2 7-pin 0.1" header

This header provides connections to power the OctoQuad from a 3.3v supply and exposes the I2C, SPI, and UART interface pins**. Note that the pins are muxed and some pins serve multiple functions** (however, only one interface can be active at a time; see section 4).

PCB Pin #	PCB Pin Label	Function
1 (square pad)	GND	Ground
2	3.3v	3.3v power input
3	SCK / SDA	SPI Clock **OR** I2C bus data
4	MISO / SCL	SPI Transmit **OR** I2C bus clock
5	CS / URx	SPI Chip Select **OR** UART receive
6	MOSI / UTx	SPI receive **OR** UART transmit
7	RST	Reset line (active low)

3.3 4-pin JST-PH dedicated I2C Connector

This connector provides connections to power the OctoQuad and exposes the I2C interface data pins. It is pin-compatible with the I2C ports on the REV Robotics Control Hub / Expansion Hub.
NOTE: The I2C lines exposed on this connector are electrically connected to the corresponding pins on the 7-pin header.

PCB Pin Label	Function
G	Ground
3	3.3v power input
D	I2C bus data line
C	I2C bus clock line

4 Interface Selection

4.1 Overview

Only one interface (I2C/SPI/UART/USB) can be active on the OctoQuad at a given time. Your choice of interface is saved to non-volatile storage and is automatically applied at power-up. **The default interface is I2C**. **WARNING: Operating the OctoQuad in a different interface mode than that which it has been electrically wired for in the connection to the host device may cause permanent damage to the OctoQuad or to the host device!**

4.2 Changing the active interface

To change the interface, follow the procedure below:

1. Remove power from the OctoQuad
2. Hold the Mode Select button ('M' on resin printed case) while applying power to the OctoQuad. Upon applying power, the LED should light and stay lit.
3. Release the Mode Select Button
4. The LED will now loop displaying a blink sequence followed by a pause to indicate a mode. Press and release the Mode Select button to cycle through the various modes. The table below lists how many blinks correspond to which interface mode.
5. Once the LED is indicating the desired mode, press and hold the Mode Select button until the LED stays on solid.
6. Release the Mode Select Button
7. After a short time, the LED will begin blinking the sequence for the interface just selected. Your interface choice is now stored in flash and will be applied at all future startups.

Number of blinks	Interface
1	I2C
2	SPI
3	UART
4	USB

5 LED Status indications

5.1 Overview

The status light on the OctoQuad is used to indicate various states of communication with a bus master.

5.2 LED Patterns

5.2.1 Looping sequence of various number of blinks followed by a pause

Indicates that the OctoQuad is powered up and ready to accept communications over the interface corresponding to the number of blinks (see table in *Interface Selection* section).

5.2.2 Rapid flashing (7Hz)

Indicates that there is in-flight or recent communication on the bus

5.2.3 Slow flashing (1Hz)

Indicates that bus communication has occurred since power-up, but no recent communication has occurred.

5.2.4 Very rapid flashing (10Hz)

Internal error; please contact Digital Chicken Labs support (digitalchickenlabs@gmail.com)

6 Field-Upgradable Firmware

6.1 Obtaining firmware files

From time to time, official firmware updates for the OctoQuad may be released. Firmware binaries may be found at https://github.com/DigitalChickenLabs/OctoQuad. **WARNING:** Flashing unofficial firmware may cause permanent damage to the OctoQuad, or to devices to which it is connected.

6.2 Flashing firmware

To flash a firmware image onto the OctoQuad, follow the procedure below:

1. Remove all power and data connections from the OctoQuad.
2. Press and hold the BOOTSEL button ('B' on resin printed case)
3. While holding BOOTSEL, connect the OctoQuad to a computer using the micro-USB port
4. Wait until the emulated USB drive appears on the computer. The LED will remain off.
5. Drag-n-drop the firmware image onto the emulated USB drive
6. The OctoQuad will automatically flash the firmware and reboot. Flashing is complete when the emulated USB drive disappears and the status LED begins blinking an interface code.

7 Registers (I2C / SPI mode)

7.1 Register access

Some registers are read-only, some are write-only, and others are read/write, as indicated in the register map. Writing to a read-only register will have no effect. Data read from a write-only register is undefined.

7.2 Register Map

Address	Type	Access	Contents
0x00	uint8_t	Read-Only	Chip ID (will read 0x51)
0x01	uint8_t	Read-Only	Firmware version (major)
0x02	uint8_t	Read-Only	Firmware version (minor)
0x03	uint8_t	Read-Only	Firmware version (engineering)
0x04	uint8_t	Write-Only	Command Register
0x05	uint8_t	Read / Write	Command Data Register 0
0x06	uint8_t	Read / Write	Command Data Register 1
0x07	uint8_t	Read / Write	Command Data Register 2

0x08	uint8_t	Read / Write	Command Data Register 3
0x09	uint8_t	Read / Write	Command Data Register 4
0x0A	uint8_t	Read / Write	Command Data Register 5
0x0B	uint8_t	Read / Write	Command Data Register 6
0x0C – 0x0F	int32_t	Read-Only	Channel 0 data (quadrature count **OR** µs pulse width)
0x10 – 0x13	int32_t	Read-Only	Channel 1 data (quadrature count **OR** µs pulse width)
0x14 – 0x17	int32_t	Read-Only	Channel 2 data (quadrature count **OR** µs pulse width)
0x18 – 0x1B	int32_t	Read-Only	Channel 3 data (quadrature count **OR** µs pulse width)
0x1C – 0x1F	int32_t	Read-Only	Channel 4 data (quadrature count **OR** µs pulse width)
0x20 – 0x23	int32_t	Read-Only	Channel 5 data (quadrature count **OR** µs pulse width)
0x24 – 0x27	int32_t	Read-Only	Channel 6 data (quadrature count **OR** µs pulse width)
0x28 – 0x2B	int32_t	Read-Only	Channel 7 data (quadrature count **OR** µs pulse width)
0x2C – 0x2D	int16_t	Read-Only	Channel 0 velocity (counts or µs / sampling interval)
0x2E – 0x2F	int16_t	Read-Only	Channel 1 velocity (counts or µs / sampling interval)
0x30 – 0x31	int16_t	Read-Only	Channel 2 velocity (counts or µs / sampling interval)
0x32 – 0x33	int16_t	Read-Only	Channel 3 velocity (counts or µs / sampling interval)
0x34 – 0x35	int16_t	Read-Only	Channel 4 velocity (counts or µs / sampling interval)
0x36 – 0x37	int16_t	Read-Only	Channel 5 velocity (counts or µs / sampling interval)
0x38 – 0x39	int16_t	Read-Only	Channel 6 velocity (counts or µs / sampling interval)
0x3A – 0x3B	int16_t	Read-Only	Channel 7 velocity (counts or µs / sampling interval)

7.3 Register descriptions

7.3.1 Chip ID register

This register will *always* read **0x51** and may be used to confirm a proper bus connection with the OctoQuad.

7.3.2 Firmware version registers

The firmware version follows the scheme *major.minor.engineering* where each of three numbers is obtained from the corresponding register. For instance, if the registers read {2, 3, 4} then the firmware version is 2.3.4.

7.3.3 Command Register & Command Data Registers 0-6

The Command Register can be used to issue various commands to the OctoQuad, with up to 7 bytes of related data (to be written to the command operand registers). See the *Commands* section.

7.3.4 Channel data registers

These registers contain either quadrature counts or pulse width (in microseconds) for each channel, depending on the channel bank configuration. In either case, the value for each channel is a signed 32-bit integer. (Pulse width will, of course, never be negative).

7.3.5 Channel velocity registers

These registers contain signed 16-bit velocity measurements for each channel.

For quadrature encoders, the velocity is defined as the net change in counts during the velocity sampling interval (see below). For example, if the sampling interval is 100ms and at the beginning of the interval the encoder count is 1234 and at the end of the interval the count is 1200, then the velocity value reported in the register will be -34. This would indicate a velocity of -34 counts/0.1s, or -340 counts/s. To determine the velocity in counts/s, user code must perform the appropriate multiplication factor based on the configured measurement interval.

The velocity sampling interval can be reduced to prevent overflow of the 16-bit counters when using encoders that output a very large number of steps per second, or, it can be increased to provide greater velocity precision on low step-rate encoders.

For pulse width input (absolute encoders) velocity is defined as the net change in microseconds pulse length during the velocity sampling interval. (See discussion of quadrature velocity above). Wrap-around is tracked internally at a much higher speed than the velocity measurement interval, so even if an absolute encoder is rotated more than a full rotation during the velocity measurement interval, the reported velocity will still be correct. **Note, however, that when using an absolute pulse width encoder, the channel pulse width min/max parameter must be set correctly.**

8 Commands

8.1 Description

The Command Register (see register map) may be used to issue various commands to the OctoQuad, with up to 7 bytes of related data (to be written to the command operand registers).

Not all commands require this extra data. For those that do, the operand register(s) *must be written in the same bus transaction in which the Command Register is written*.

8.2 Command List

The following commands are supported:

Command	Description	Operand 0	Operands 1-6
0x00	NO-OP (No command)		
0x01	Set Parameter	Parameter ID	Parameter-dependent
0x02	Get Parameter	Parameter ID	Parameter-dependent
0x03	Save Parameters to flash		
0x14	Reset Everything		

0x15	Reset Channels	8-bit channel bitfield	

8.2.1 Reset Everything Command

This command resets quadrature encoder counts or measured pulse width to zero, and sets all parameters to their factory defaults. NOTE: this does *not* save the newly reset parameters to flash.

8.2.2 Reset Channels Command

This command zeros quadrature count(s) / pulse width measurement for one or more channels. The first and only operand is a bitfield mapping to channel numbers. Each bit in the operand corresponds to a channel, e.g., bit 3 corresponds to channel 3. When issuing this command, for every bit that is set in the operand, the corresponding encoder's count will be reset.

Multiple channels can be reset in one command operation. For example, writing 01000001 as the operand will reset channel 6 and channel 0.

Reset Channel Command – Operand 1								
Bit	7	6	5	4	3	2	1	0
Effect	C7 Reset	C6 Reset	C5 Reset	C4 Reset	C3 Reset	C2 Reset	C1 Reset	C0 Reset

8.2.3 Set Parameter Command

This command is used to set the value for a parameter. See below section on parameters.

8.2.4 Get Parameter Command

This command is used to get the current value of a parameter. See below section on parameters.

8.2.5 Save Parameters to Flash Command

This command may be used to save the current value of all parameters to flash, so that they will be automatically restored after a power cycle.

9 Parameters

9.1 Description

The OctoQuad supports various user-configurable options ("parameters") which affect its operation. *Parameters are not directly mapped to registers*. Parameters may optionally be saved to flash so that they are automatically restored after a power cycle. (See *Save Parameters to Flash* command).

9.2 Parameter List

Parameter ID	Name	Values
0x00	Channel directions	Channel bitfield (uint8_t)
0x01	I2C Recovery Mode	I2C Recovery mode (uint8_t)
0x02	Channel Bank Config	Channel Bank Mode (uint8_t)

0x03	Channel Velocity Interval	Interval_ms (uint8_t)
0x20	Channel Pulse Width min/max	Min_µs (uint16_t) Max_µs (uint16_t)

9.3 Setting Parameters

A Parameter may be set by writing the *Set Parameter* command ID to the command register and filling the command data registers (sequentially) with the parameter ID, followed by the value(s) for the parameter. **For parameter names in red the parameter values must be preceded by an 8-bit integer corresponding to the channel index.** (I.e. the first command data register filled after the parameter ID must be the desired channel index, then the parameter value(s) follow in subsequent command data registers). The general format for setting parameters is as follows:

Setting a Parameter (write to these registers)			
Register	Command (0x04)	Cmd Data 0 (0x05)	Cmd Data 1-6 (0x06 – 0x0B)
Data	Set Param (0x01)	Param Number	Parameter Vals. (1st may be ch idx)

9.4 Reading Parameters

Reading the current value of a parameter is accomplished in two steps. First, write the *Read Parameter* command ID to the command register and fill the command data register 0 with the parameter ID to be read. **If reading a parameter name in red, then data register 1 must be filled with a channel index.** Once the *Read Parameter* command has been issued, the current parameter value will be filled into the command data registers (starting with command data 0) which can be retrieved with a subsequent read.

9.5 Parameter Descriptions

9.5.1 Channel Directions Parameter

This parameter is used to set the quadrature encoder count direction on a per-port basis. It has no effect on the channel if the channel is operating in pulse width input mode. The first and only argument is a bitfield mapping to channel numbers. Each bit in the argument corresponds to a channel, e.g., bit 3 corresponds to channel 3. When issuing this command, for every bit that is set in the operand, the corresponding encoder channel will be negated.

Multiple channels can be configured one write to this register. For example, writing 01000001 to the operand will set channel 6 and channel 0 to be negated.

Encoder Directions Parameter – Argument 0								
Bit	7	6	5	4	3	2	1	0
Effect	E7 DIR	E6 DIR	E5 DIR	E4 DIR	E3 DIR	E2 DIR	E1 DIR	E0 DIR

9.5.2 I2C Recovery Mode Parameter

This parameter is used to set how aggressively the OctoQuad will attempt to un-wedge a hung I2C bus. It has no effect if the OctoQuad is operating in any interface mode other than I2C. Three modes are supported:

- 0: The OctoQuad will not attempt to perform any type of recovery for a stuck I2C bus
- 1: An inter-byte timeout is used for I2C transactions: successive byte transfers must occur within 50ms of each other in order to prevent the timeout from expiring. If the timeout expires, the firmware will assume that the bus has become wedged and will reset the I2C peripheral in an attempt to recover the bus.
- 2: Inter-byte timeout from Mode 1, plus pulling clock low for a small period of time if 1500ms elapses with no communications. May help to un-wedge master-side I2C hardware on an incredibly glitch/noisy bus.

9.5.3 Channel Bank Mode Parameter

The OctoQuad contains two channel banks, covering channels 0-3 and 4-7. This parameter may be used to set which mode (quadrature or pulse width measurement) each channel bank is configured for. Possible values are:

- 0: All quadrature
- 1: All pulse width
- 2: First bank quadrature; second bank pulse width

9.5.4 Channel Velocity Measurement Interval Parameter

This parameter is used to set the time interval at which the velocity is calculated for each encoder. The value is interpreted directly as milliseconds. For example, setting the value of this parameter for a channel to the decimal value “40” means that the velocity for the respective channel will be measured at 40ms intervals. The default interval is 50ms. Setting the sampling interval to 0 will be disregarded.

9.5.5 Channel Pulse Width min/max Parameter

This parameter is used to inform the firmware of the minimum/maximum pulse lengths that an absolute encoder will output, to enable accurate velocity calculation. This will default to 1µs/1024µs

10 I2C Interface

10.1 Overview

The OctoQuad supports operating as a slave on the standard I2C interface, using the register model. Bus clock rates of up to 400KHz are supported. **The OctoQuad’s I2C address is 0x30.**

10.2 Registers

Please refer to the register map in section 7.2

10.3 I2C Wedged Bus Recovery

The OctoQuad can be configured to attempt recovery of a stuck I2C bus in certain scenarios. See section 9.5.2 for more details.

11 SPI Interface

11.1 Bus Specifications

The OctoQuad can be configured to operate in SPI interface mode, at up to 1MHz clock rate. The SPI framing format used is Motorola format 3 **(Clock high when idle, data latched on rising edge). The slave select line is high when idle.** *The slave select line must remain asserted for the entirety of the transaction. Additionally, the slave select line must be asserted for at least 50µs before the first clock cycle and asserted for at least 50µs after the last clock cycle.*

11.2 Data Protocol

The general format for SPI communication with the OctoQuad bears some degree of similarity to communicating with a register-based I2C device, but nonetheless is quite different.

11.2.1 Flag Byte

All data frames sent from the SPI bus master to the OctoQuad (including not only data frames used to write, but also those used to perform a read) **must** begin with a special flag byte. This flag byte is what distinguishes a read from a write. Since SPI is a full-duplex bus (that is, data is transferred from master to slave and from slave to master simultaneously on every clock) this flag byte serves as a simple way to differentiate writes and reads.

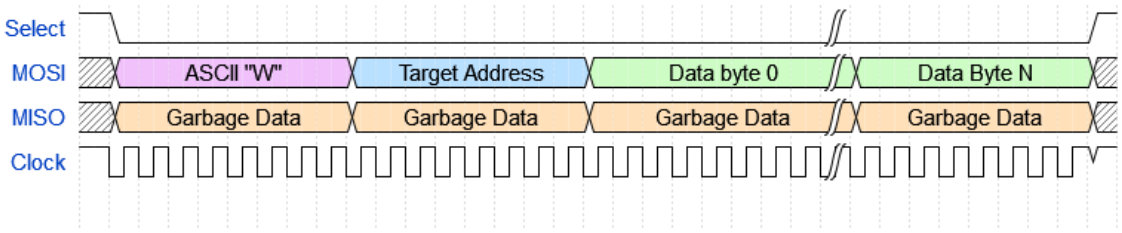
Flag	Meaning
0x57 (ASCII 'W')	Master is writing
0x53 (ASCII 'S')	Master is writing "sticky" *Source Address* (see below)
0x52 (ASCII 'R')	Master is reading

11.2.2 Writing

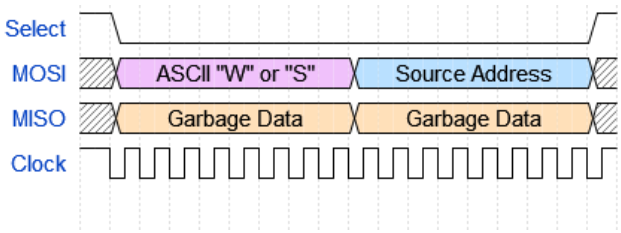
There are two different types of write operations that the master may perform, and **these operations are implicitly distinguished by the length of the data frame:** (a) Write Operation and (b) Write *Source Address* operation. Moreover, while the 'W' flag may be used for either operation, the 'S' flag may only be used with the Write *Source Address* operation.

In a Write Operation, the *Target Address* must immediately follow the 'W' flag byte. Data bytes to be written into memory starting at the *Target Address* directly follow the *Target Address* in the transaction. A Write Operation must provide at least one data byte following the *Target Address.* Otherwise, it will be interpreted as a Write *Source Address* operation.

The general format for a Write Operation is shown below:



In a Write *Source Address* operation, the new *Source Address* must immediately follow either the 'W' or 'S' flag bytes. The master must not send any more bytes following the *Source Address*; otherwise, the operation will be interpreted as a Write Operation. The *Source Address* is the address in memory from which read operations will begin returning data. The general format for a Write *Source Address* operation is shown below:



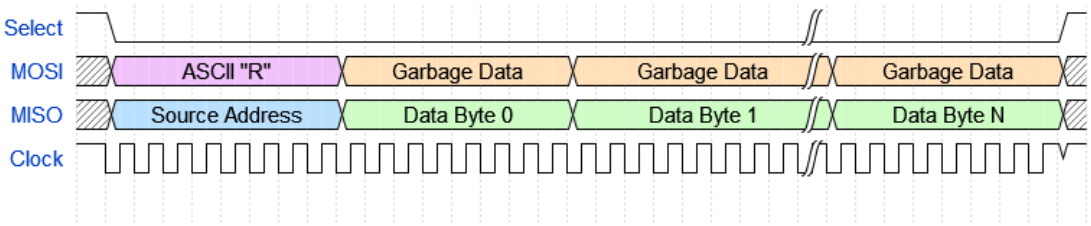
When performing either type of write operation, the master must ignore all received data from the OctoQuad.

11.2.3 Reading

All Read Operations begin with the master sending the 'R' flag, after which it may continue to perform N more byte transactions on the bus. All data sent by the master after the 'R' flag will be ignored by the OctoQuad. All Read Operations will begin sending data from the current *Source Address*. The first received byte from the OctoQuad (that is, the byte received while the master is transmitting the 'R' flag) will be the *Source Address* from which the data came. This means if the master wishes to read N bytes from the OctoQuad memory, it must actually perform N+1 byte transfer operations on the bus.

What happens after the master has finished performing a Read Operation is determined by whether the *Source Address* was set in sticky mode or not. If the *Source Address* was set in "sticky" mode, then the *Source Address* will remain **unchanged.** If the *Source Address* was not set in "sticky" mode, then the *Source Address* will be incremented by the number of bytes read during the Read Operation. Using "sticky" mode is helpful should the master wish to repeatedly read the same block of registers without reading from other locations in-between those reads. By using "sticky" mode, the master may simply perform the same read sequence repeatedly without performing a Write *Source Address* operation.

The general format for a read sequence is shown below:



11.3 Register Map

Please refer to the register map in section 7.2

12 USB Serial Interface

12.1 Overview

The OctoQuad module can be configured to provide a USB virtual serial port interface, supporting the USB CDC ACM protocol. No baud rate configuration is necessary, because the USB interface is not bridging to a physical UART.

12.2 Protocol

The USB serial protocol is a simple ASCII text-based format. On power-up, the OctoQuad will begin streaming a CSV string of all quadrature encoder counts to the host at 10Hz. The host can issue various one-character commands to the OctoQuad to adjust behavior.

12.2.1 Data Format

For quadrature count / pulse width only readings, the string will take the format
`“enc0,enc1,enc2,enc3,enc4,enc5,enc6,enc7\r\n”`

For count / pulse width & velocity readings, the string will take the format
`“enc0,enc1,enc2,enc3,enc4,enc5,enc6,enc7,vel0,vel1,vel2,vel3,vel4,vel5,vel6,vel7\r\n”`

The values reported are in base 10 (decimal). An example string with velocity reporting might look like:

`“5897,0,0,3974,0,0,0,0,121,0,0,-230,0,0,0,0\r\n”`

In this case, encoder 0 count is 5897, encoder 3 count is 3974, encoder 0 velocity is 121, and encoder 3 velocity is -230

12.3 Command Table

ASCII Char	Effect
‘R’	Reset all encoder counts
‘0’	Reset encoder 0 count
‘1’	Reset encoder 1 count
‘2’	Reset encoder 2 count
‘3’	Reset encoder 3 count
‘4’	Reset encoder 4 count
‘5’	Reset encoder 5 count
‘6’	Reset encoder 6 count
‘7’	Reset encoder 7 count
‘V’	Enable velocity reporting (following count reporting)
‘v’	Disable velocity reporting
‘Q’	Set channel bank mode 0 (all quadrature)
‘P’	Set channel bank mode 1 (all pulse in)

‘H’	Set channel bank mode 2 (1st bank Q , 2nd bank P)
‘F’	Set streaming rate to “fast” (60Hz)
‘M’	Set streaming rate to “medium” (30Hz)
‘S’	Set streaming rate to “slow” (10Hz)

13 UART Interface

13.1 Overview

The UART interface runs at 115200 baud and mirrors the USB serial interface protocol.

Special Thanks To

- Chris Johannesen: hardware testing
- Laina Galayde: OctoQuad logo artwork
- Uday Vidyadharan: help with Python sample code