Capwheel
Electrical and communications documentation

Jonathan Lock

November 28, 2011

Contents

1 **Introduction** 1
- 1.1 TODO 1

2 **Usage guide** 1
- 2.1 Communication protocol 1
  - 2.1.1 Transmit bytes 2
  - 2.1.2 Receive bytes 2
  - 2.1.3 Example 3

3 **Electrical design** 3

4 **Implementation notes** 3

1 Introduction

Capwheel is a capacitive sensor wheel and button that allows for the low cost replacement of rotary encoders and navigation switches. It can output rotation amounts and button presses either in a polled or automatic mode using an asynchronous serial interface. There is also a configurable RGB led that allows setting a desired hue/saturation/lightness. See the code and/or code documentation document for details as to how to code actually works.

1.1 TODO

The following aspects of the documentation need to be finished;

- Current consumption (baseline, LED off; maximum brightness, average)
- Photographs of finished sensor
- Code snippets and/or library for interfacing the module in C.

2 Usage guide

Using capwheel is meant to be as easy as possible. The sensor consists of six pads for the encoder and one center button. Each of the encoder pads also work as a button, allowing for a total of 7 buttons and rotation. As there is no way for the device to know on a no touch -> touch transition if the user intends to perform a button or rotation action all button actions are active on release. A rotation gives either 6 or 12 pulses per rotation, depending on a flag set at compile time.

All calibration of sensor pads is done on startup and is fully automatic, only a little bit of configuration is required over the serial interface.

2.1 Communication protocol

The communication protocol uses an UART compatible format set up as:

- 8 data bits
- 1 start/stop bit
- 0 parity bits
- 19200 baud

Each message is one byte in length, both in the transmit and receive direction. There is also a chip select line[1] that can be used to make the input on the unit ignore and commands. (This can be used to put this device and another write-only device in parallel using time-division multiplexing). Pull the chip select line high or float it to enable the input stage, ground it to ignore

[1]that may be configured for doing other things based on compile definitions, see section 4.

all input messages. The chip select line status is checked at the end of every received message in the device, so it must be at the correct value when a packet is fully received (there is some delay due to the interrupt taking some time to execute).

The default state is for the unit to output all logged button/rotation events when triggered by sending a specific byte over the serial interface (this is called the polled mode). The unit will then output all the button presses and rotation increments that have happened since boot-up or the last check, with the order preserved. The oldest 128 events are stored, if more events occur they will be ignored/not stored. This can be changed to an automatic mode where a button/rotation event is immediately output.

2.1.1 Transmit bytes

In the transmit direction there are four possible payloads, see table 1. When sending a hue, saturation or lightness command set the correct header bits and then b_5 to b_0 will correspond to the desired value, from 0 to 63 (MSB first). The control register currently only uses two bits;

- b_1 controls the automatic mode setting. If set to 1 a button/rotation event will be immediately sent over the serial interface, if set to 0 (default) events will be internally logged and transmitted when triggered.

- b_0 triggers sending all logged events when in polled mode. Set this bit to 1 to send all the logged button/rotation events.

- b_5–b_2 are reserved for future use. Do not set these to 1 for future compatibility. (These bits are ignored, for now).

To avoid race conditions be sure to only change one bit at at time when changing status (ie. don't set b_1 and b_0 in the same message).

2.1.2 Receive bytes

In the receive direction there are some types of messages as follow;

0x80 The center button was pushed

0x81 to 0x87 Button 1 to 7 was pushed

Table 1: Transmit byte formats.

Function	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Set hue	1	1	Hue control					
Set saturation	1	0	Saturation control					
Set lightness	0	1	Lightness control					
Set control register	0	0	Reserved for future use				Auto mode	Polled mode trigger

0x01 Clockwise rotation, one increment (one byte sent per increment)

0x02 Counterclockwise rotation, one increment (one byte sent per increment)

0xFF Buffer emptied (only sent when in polled mode to signify that all the logged events have been output)

2.1.3 Example

Table 2 shows an example of events that occur and their resulting output (or lack thereof).

3 Electrical design

The electrical design is relatively simple, the only active components are an Atmel AVR MEGA88 and a 16MHz crystal. A charge-transfer topology is used for the capacitive sensing which uses the high/low/tristate functionality of the AVR extensively.

All I/O and power supply is done over the 8-pin micromatch connector. See table 3 for a pinout and valid voltage/current ranges. There are also holes for a standard 2x3 pin ISP header as well as traces that can be cut for the LED if jumpers would like to be added.

Figures 1 , 2, 3 and 4 illustrate the schematic and board layout for the design. Note the sandwiching structure for the LED that allows illumination in the center. Mounting holes are 3.2mm in diameter, centered around the sensor and spaced 27.94mm apart (1.1 inches). The PCB has a total size of 35.56mm square (1.4 inches). The most basic hole pattern in an enclosure is 4 3.2mm holes on a 27.94mm grid with a single 28mm hole centered around these.

4 Implementation notes

Some things to keep in mind when implementing the sensor are:

- A nonconductive covering of up to (approx) 0.5-1mm on top of the sensor should be ok.
- Adjust the constants ZONE_IM_S_THR and ZONE_MT_S_THR in the program code to adjust the thresholds for touch and rotation when using a new cover.

Event	RX (input)	TX (output)	Chip select	Internal buffer
Power up	None	None	High	Empty
Touch center button	None	None	High	$\{0x80\}$
Touch button 1	None	None	High	$\{0x80, 0x81\}$
Rotate CCW twice	None	None	High	$\{0x80, 0x81, 0x02, 0x02\}$
Send poll bit ($b_0 = 1$)	$0x01$	$\{0x80, 0x81, 0x02, 0x02, 0xFF\}$	High	Empty
Send poll bit ($b_0 = 1$)	$0x01$	$\{0xFF\}$	High	Empty
Send poll bit ($b_0 = 1$)	$0x01$	None	Low	Empty
Set automatic bit	$0x02$	None	High	Empty
Touch button 2	None	$\{0x82\}$	High	Empty
Rotate CW	None	$\{0x01\}$	High	Empty
Rotate CW	None	$\{0x01\}$	High	Empty
Set LED saturation to max	$0b10111111$	None	High	Empty

Table 2: Example events and output.

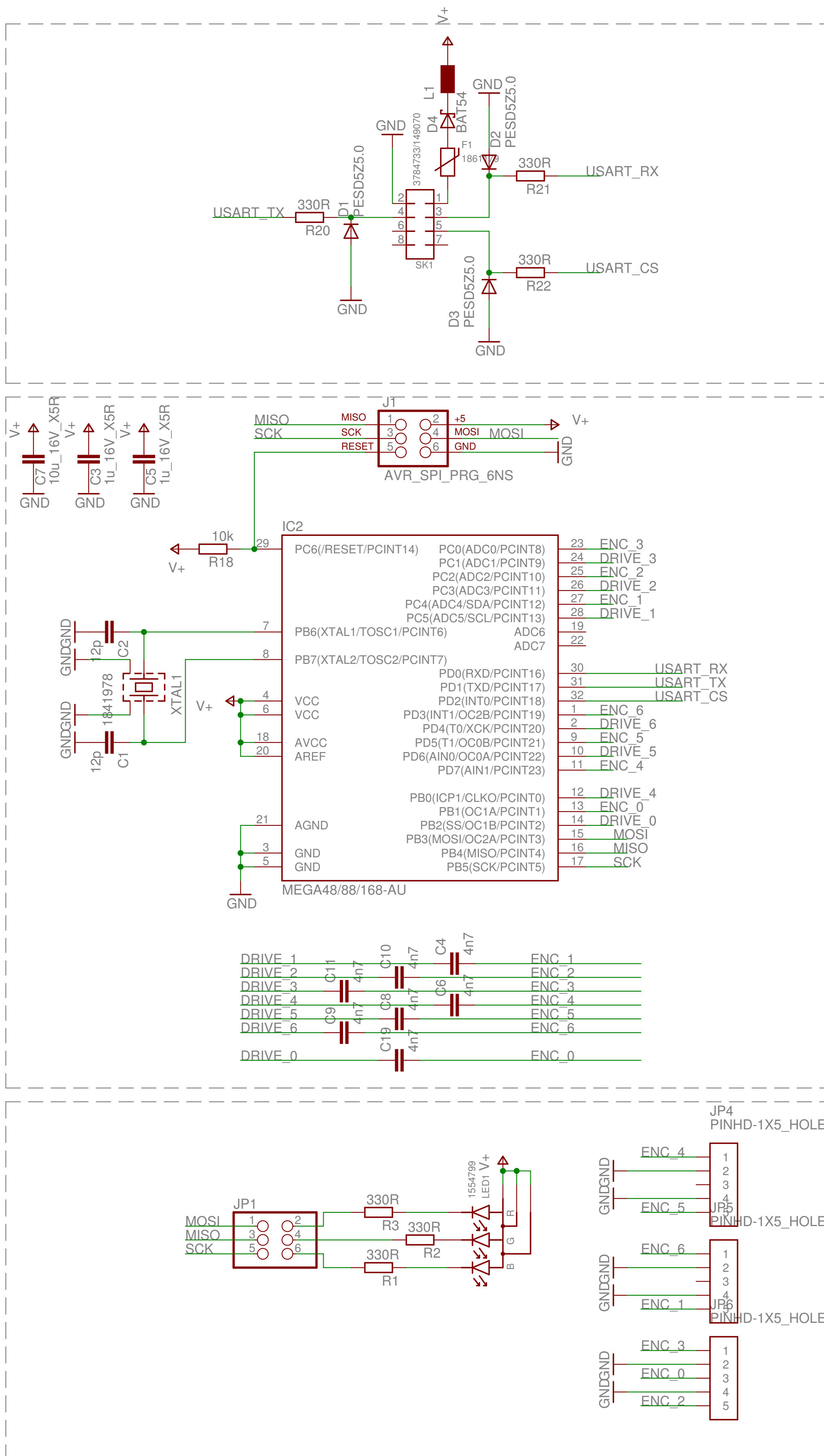

Figure 1: Schematic of bottom PCB

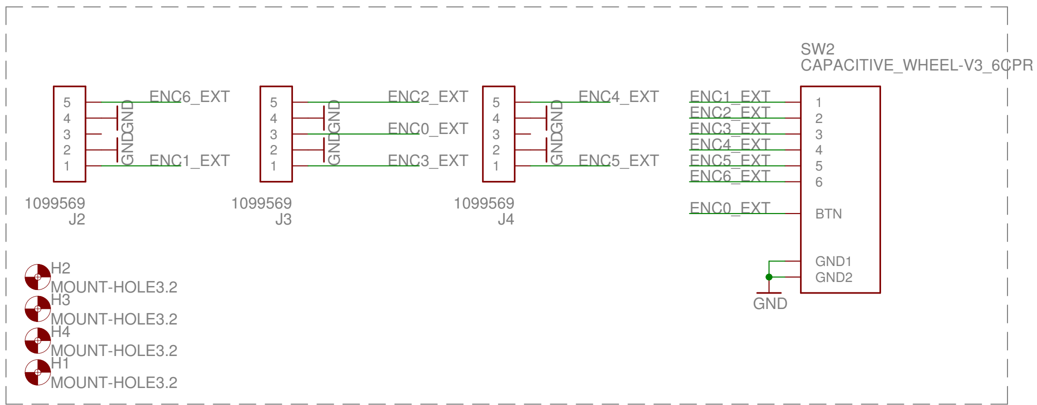

Figure 2: Schematic of top PCB

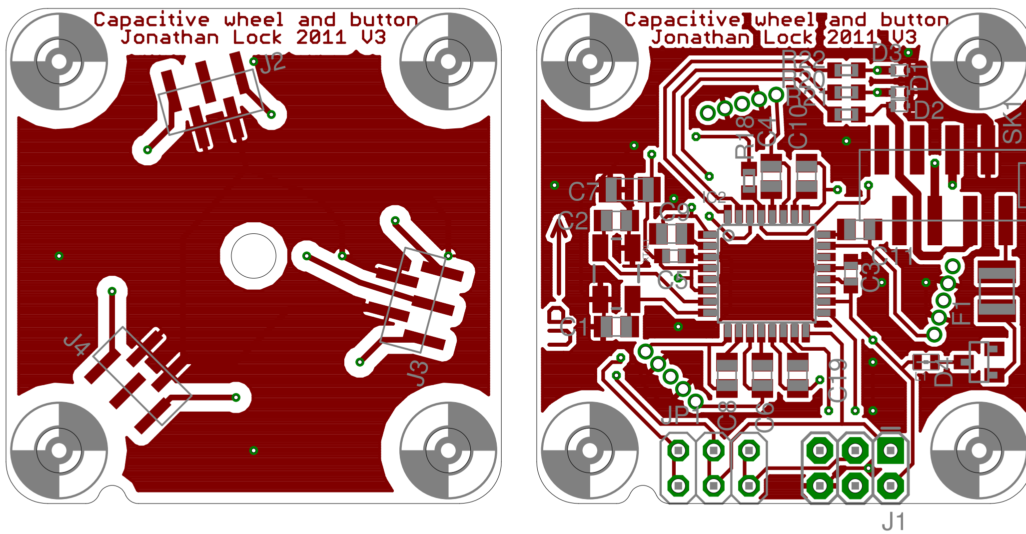

Figure 3: Top layers of PCBs

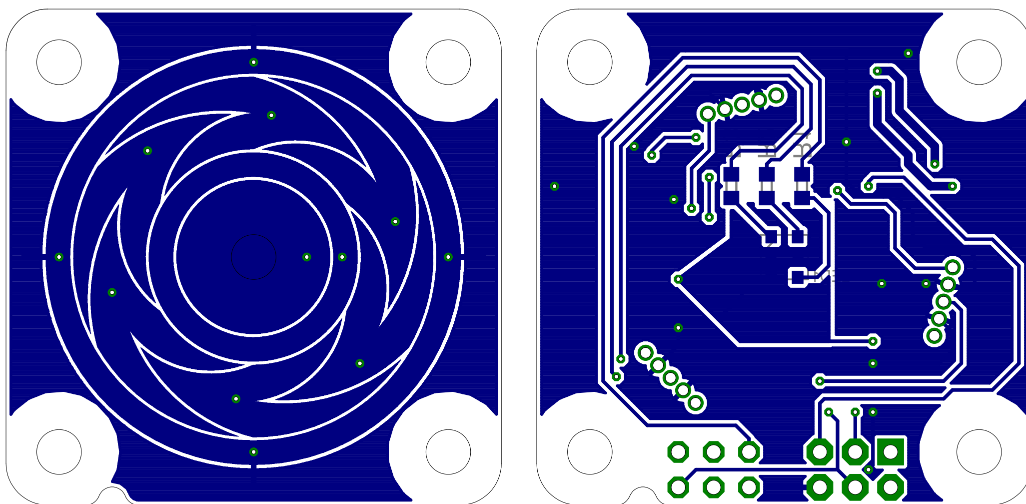

Figure 4: Bottom layer of PCBs

Pin number	Use	Nominal Voltage	Notes	Limit
1	Positive supply (V^+)	4.0-5.5V	Decoupled on board (esd insensitive)	6V
2	Ground	-	All grounds are common	-
3	USART RX (input)	0-V^+	ESD protected and with a 330R resistor in series[a]	±10mA
4	USART TX (output)	0-V^+	ESD protected and with a 330R resistor in series[b]	±10mA
5	USART CS (input)	0-V^+	ESD protected and with a 330R resistor in series[c]	±10mA
6-8	NC	-	-	-

Table 3: Pinout and voltage/current specifications.

[a] clamps to V^+ and GND
[b] clamps to V^+ and GND
[c] clamps to V^+ and GND

- Grounding the sensor is nice as it gives a larger change in capacitance.
- The chip select line may also be configured to perform other things based on the compile-time configuration. It can be disabled or configured to trigger a system reset on a low->high transition.