eDVS

The eDVS is a 128 x 128 DVS sensor interfaced with NXP microcontroller on an embedded board

Specifications

DVS RESOLUTION	128 X 128 PIXELS
DVS DYNAMIC RANGE	120 dB
MIN. LATENCY	~ 12 us @ 1 klux with optimized biases
LENS MOUNT	M12 x 0.5 mm
I/O	2 UART connectors up to 12 Mbps (standard firmware 4 Mbps), 200/600 kEPS with/without timestamps, hardware handshake USB 2.0 FTDI serial up to 12 Mbps (shared with UART0) SPI/SSI/Microwire (up to 25/50 Mbit half/full duplex) 6 PWM digital outputs (2.8 V) 4 PWM power outputs (2 motors) 6 analog inputs (2.8 V) 6 digital general purpose I/O (2.8 V) 1 microSD slot (32-bit FAT)
SOFTWARE	DV-Platform, NXP IDE
POWER SOURCE	USB Mini-B or external @ 5 VDC Compatible with single cell LiPo battery (re)charging through USB
DIMENSIONS	L 80 x W 50
WEIGHT	23g (with lens)
HARDWARE MULTI-CAMERA SYNC	Supported (Pinhead connector)
ON-BOARD COMPUTE	32-bit dual-core RM up to 204 MHz Hardware FPU, 136 KB SRAM, 1MB program / data flash
IMU	Yes
SPECIAL FEATURES	Adjustable wake-up on visual activity Hardware real-time clock
CMOS TECHNOLOGY	0.35 um 2P4M
CHIP SIZE	6 x 6.6 [mm]
PIXEL SIZE	40 x 40 [um]
ARRAY SIZE	5.12 x 5.12 [mm]
FILL FACTOR	8.1 %
PIXEL COMPLEXITY	26 transistors, 3 capacitors, 1 photodiode
CHIP VOLTAGES	3.3 V
CHIP POWER CONSUMPTION	23mW (activity dependent)

Specifications not guaranteed. All specifications subject to change without notice

Board layout

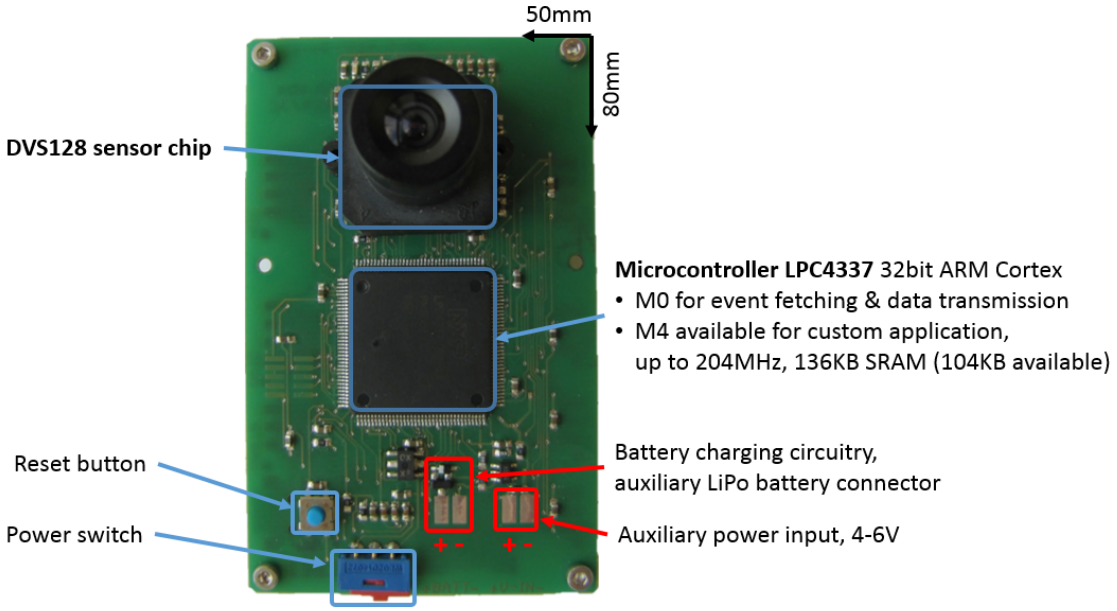


Figure 1 eDVS fron layout

Back:

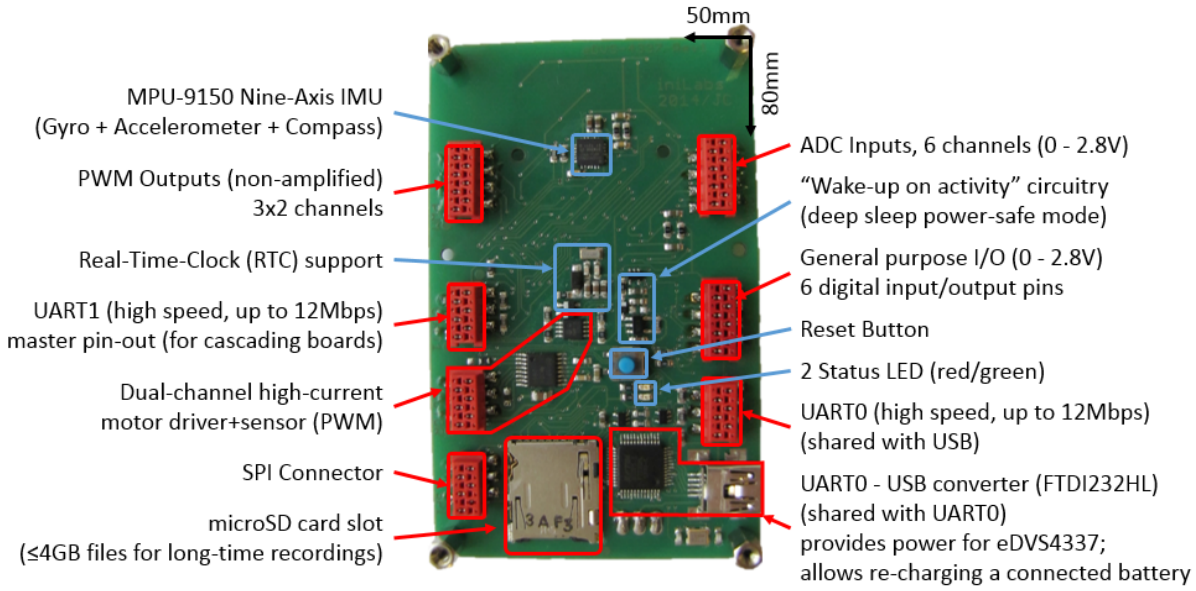


Figure 2 eDVS back layut

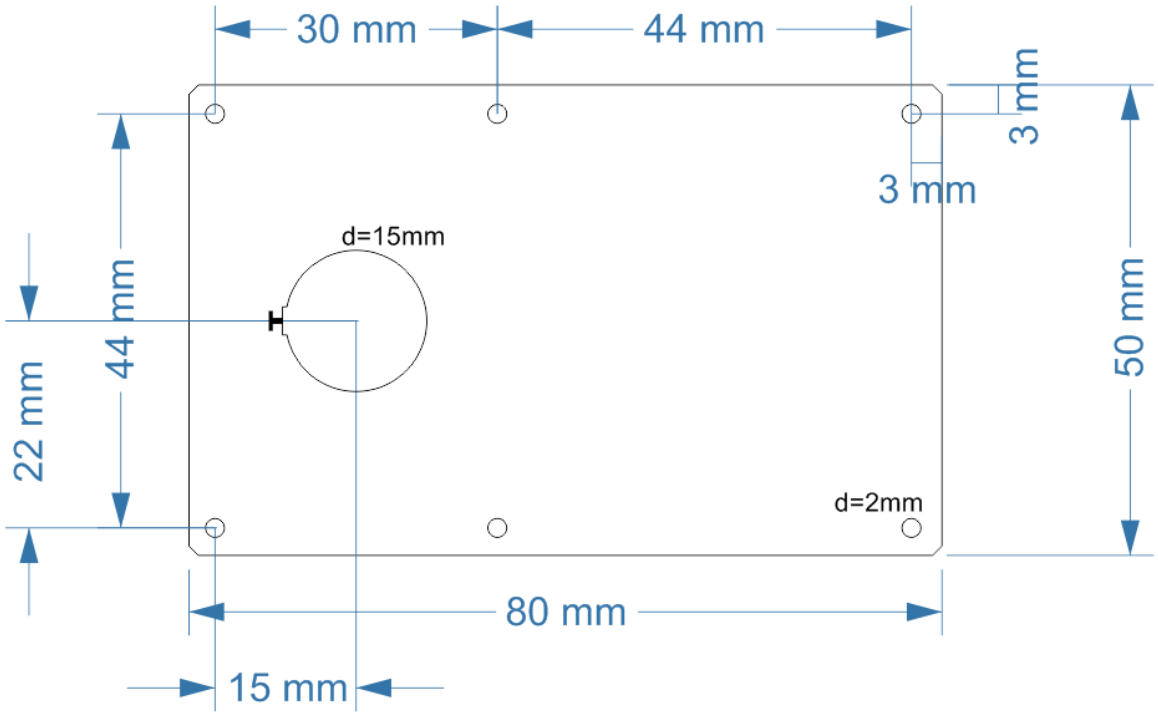


Figure 3 eDVS board dimensions

Getting started

Viewing and processing events

The officially supported software to view and process events coming from the eDVS is DV. Visit the inivation website to get the newest copy of DV.

Once installed, connect the eDVS via USB to your computer. The drivers for USB serial port emulation are already part of all modern operating systems, so no special installation is required at this point. The recently connected devices should show up on your system as a serial communication device, for example as a COM:<x> port on Windows or `/dev/ttyUSB<x>` on Linux. If your OS does not find drivers, download FTDI's latest VCP drivers for FT232HL
Note for Linux users: you may need to ensure you have the correct permissions for accessing /dev/ttyUSBn (where n = 0,1,2, etc). For example, running the following command as root:

```
chmod 666 /dev/ttyUSB0
```

Will give you access to device ttyUSB0.

Custom firmware development

To take advantage of the more advanced features offered by the eDVS4337 boards, you might need to tailor the standard firmware to your own application's needs. The latest version of the firmware can be checked out with Git from:
https://gitlab.com/inivation/devices-bin/tree/master/firmware/eDVS4337
Please follow the instructions within that directories README.txt file to install the LPCXpresso development environment.

Firmware reprogramming

The post build steps of the LPCXpresso build script generate an Intel HEX file for reprogramming through the UART0 port or the USB plug.
You can find already compiled HEX files at
https://gitlab.com/inivation/devices-bin/tree/master/firmware/eDVS4337/Releases

The latest hex files are EDVSBoardOS-4mbps.hex; EDVSBoardOS-6mbps.hex and EDVSBoardOS-12mbps.hex. Choose the hex file denoted with the baud rate you want. If the higher baud rates (6 mbps and 12 mbps) don't work, you can move down to the 4 mbps file.
To enter reprogramming mode, use the programming command from a serial console:

```
P\n
```

Ensure you do not hit any other keys or send further characters after entering reprogramming mode. Just close the terminal window.

Windows & MacOS X

For Windows and Mac OS X, the supported tool is FlashMagic.
The current version (9.51) runs on Windows XP/Vista/7/8 and version 8.50 on MacOS X 10.6+. The following instruction were written for FlashMagic 7.85, but still apply to newer version as well.

Step 1 - Communication

Device: **LPC4337** Flash Bank: A Com Port: Serial port of the FTDI chip, depends on system. Baud Rate: **115200** (try lower baud rates if connection fails; e.g. **19200**) Interface: **None (ISP)** Oscillator (MHz): **12**

Step 2 - Erase

'Erase all Flash' should be ticked

Step 3 - Hex File

Select the Hex file provided or the one you generated. If you build the project with LPCExpresso, take the HEX file in the M4/Release (or M4/Debug) folder. The files in M4 contain the M0 code as well.

Step 4 - Options

'Verify after programming' and 'Activate Flash Bank' must be ticked.

Step 5 - Start

Click the **'Start'** button

Step 6 - Waiting

Please stand by while new firmware is sent through the Serial Port to your board.

Step 7 - Executing

After the programming and verification has ended, restart the device by simply re-plugging the USB cable. Note: at the end of the process you might see a

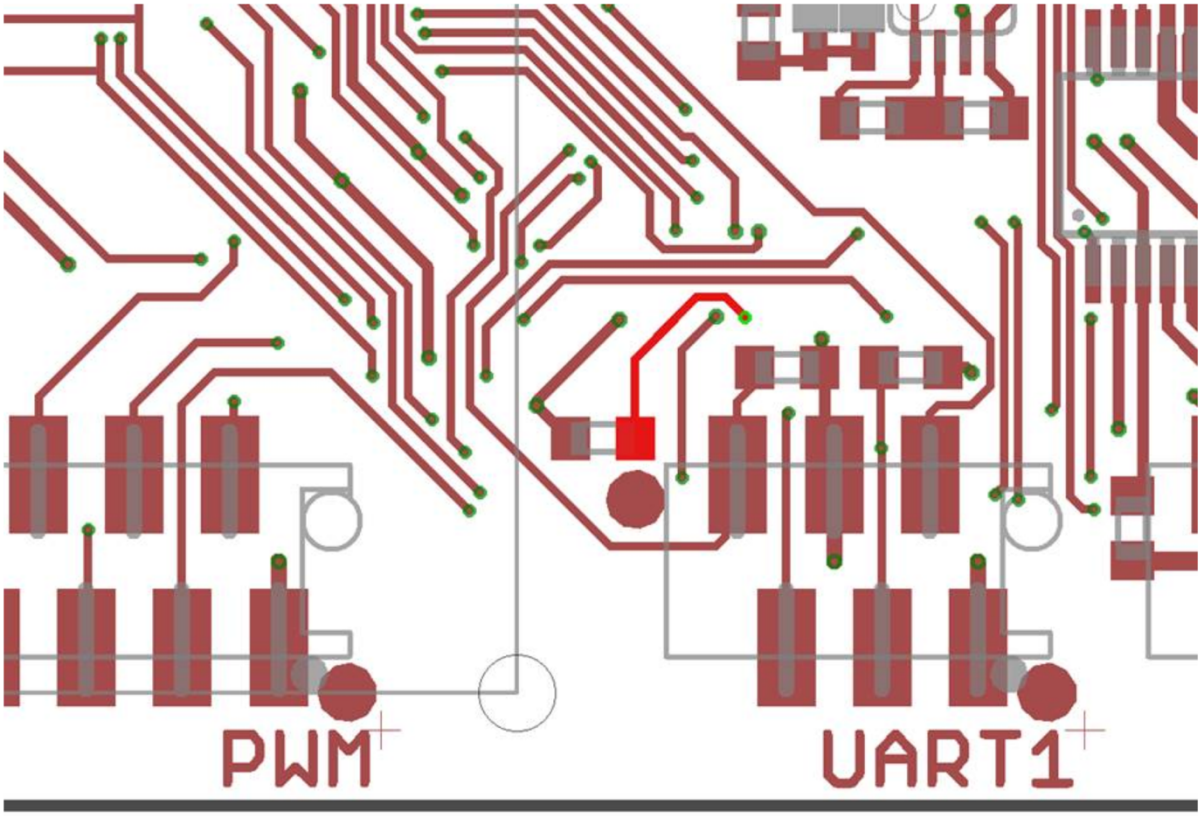
message “Operation Failed. (activating flash bank)”, this is normal, and the programming has taken effect.
If you get the error message ‘Operation Failed. Failed to autobaud - step 1’ this is likely because you haven’t entered programming mode as shown above.

Linux

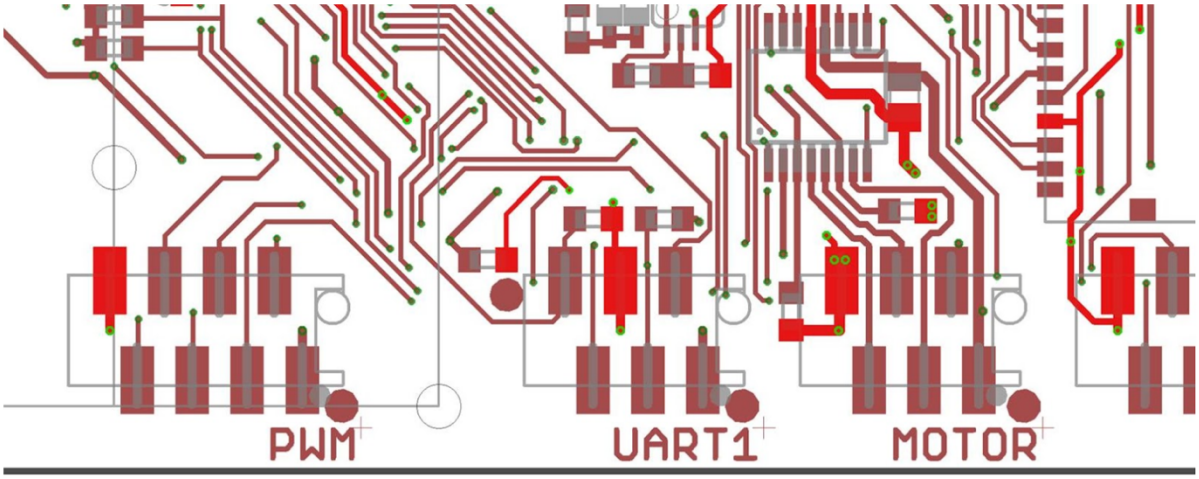
Flash magic is not available for Linux; an open-source alternative is:
http://www.windscooting.com/softy/mxli.html#Latest
Further instructions on request (the process is similar to the above).

Recovering from faulty firmware

It is possible to create faulty firmware that isn’t able to correctly boot on the microprocessor of the eDVS. If this firmware gets flashed to the device, it won’t be able to start correctly and you won’t be able to simply reprogram it as detailed above, since the whole infrastructure to enter programming mode via USB isn’t available then.
To correct this, it’s possible to manually force the microprocessor to enter programming mode directly, by shorting a specific pin to GND (ground). This pin is P2_7, more information can be found in Section 6.2 “Pin description” of the LPC4337 datasheet.
On the PCB, this pin is pulled up to VCC through a pull-up resistor by default, as can be seen here:



The track highlighted in red on the under-side of the board is the pin in question, and to pull it down you can simply short it momentarily with GND.
The following schematic extract shows also all the nearby GND pads:



While the pin is shorted to GND, press the RESET button (blue button on both sides of the board) and release it. If done correctly, the LED should stop blinking. The controller will then start directly in programming mode, so you can then use FlashMagic and the usual procedure to upload new, good firmware.

Accessing the device manually

It is possible to manually access the eDVS and send commands to it directly. This can be done by using any serial console emulation program and connecting it to the device (such as “hyperterm” or “putty” on Windows OS or “minicom” in Linux).
Here is an example using the popular Putty program on Windows, which can be downloaded at https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html. Once you’ve downloaded Putty, just run its executable. In the Session settings (top left), change the Connection type to Serial and then write the correct COM port (in our case COM3) into the Serial line textbox, and set the Speed to 4000000 for devices with serial number 613xx or to 12000000 for devices with serial numbers 126xx. Then switch to the Serial settings (bottom left) and verify that they match what you just entered. Further, change Flow control to RTS/CTS (i.e. enable hardware handshaking) and ensure Data bits is 8, Stop bits is 1 and Parity is None (often referred to as 8N1). Then switch to the Terminal settings (third from the top left), and tick Implicit CR in every LF, as well as set to Force off both Local echo and Local line editing.
After pressing the Open button, you’ll be able to send commands to the device. The list of commands is available in the next section.

UART Protocol (PC->Board)

Supported Commands (all commands need to be terminated by ‘\n’; i.e. return):

```
E+/-                    - enable/disable event sending
!ER+/-                  - enable/disable event recording (SD card)
!Ex                     - specify event data format, ??E to show options see
below for more details)
!ETx                    - set current timestamp to x (default: 0)
!ETM+                   - synch timestamp, master mode, output active
!ETM0                   - synch timestamp, master mode, output stopped
```

```
!ETS                    - synch timestamp, slave mode
!ETI                    - single retina, no external synch mode
!B[0-11]=x              - set bias register to value
!BF                     - send bias settings to DVS (flush)
!BDx                    - select and flush predefined bias set x
?Bx                     - get bias register x current value
!L[0,1,2]               - LED off/on/blinking
!U=x                    - set baud rate to x
!U[0,1,2]               - UART echo mode (none, cmd-reply, all)
!S+b,p                  - enable sensors streaming, ??S to show options
!S-[b]                  - disable sensors streaming, ??S to show options
further explanation below)
?Sb                     - get sensor readouts according to bitmap b
??S                     - bitmap b options
R                       - reset board
P                       - enter reprogramming mode
!M+/-                   - enable/disable motor driver
?MC[0,1]                - get motor PID controller gains
!MC[0,1]=p,i,d          - set motor PID controller gains
!MP[0,1]=x              - set motor PWM period in microseconds
!M[0,1]=[%]x            - set motor duty width in microseconds [% 0..100]
!MV[0,1]=[0-100]        - set motor velocity (internal P-controller for
PushBot)
!MD[0,1]=[%]            - set motor duty width, slow decay [% 0..100]
!MVD[0,1]=x             - set motor duty velocity, slow decay
!P[A,B,C]=x             - set timer base period in microseconds
!P[A,B,C][0,1]=[%]x     - set timer channel width in microseconds [% 0..100]
!T+/-                   - enable/disable Real Time Clock (RTC)
!Tyyyy-mm-dd hh:mm:ss - set RTC time
?T                      - get RTC time
??                      - display help menu
```

Enabling / Disabling Data Streaming

Use the !S command to enable or disable data streaming. Format:

!Snb,p

where n = '-' disables streaming and n = '+' enables streaming; p is the period (in milliseconds) and b is a bitmask (see below). Example:

!S+10,8

will stream the ADC channel 0 and channel 2 (10=0b1010) readings at 125Hz (8ms).

upcoming (example) reply:

```
-S1 1000\n
-S3 0250\n
```

List of available sensory data:

Bit	Decimal	Name	#	Description	Format
0	1	BATTERY	1	battery voltage (in mVolt)	up to 4 digits (0..9999)

Bit	Decimal	Name	#	Description	Format
1	2	ADC CHANNEL 0	1	raw ADC reading from pin 2	up to 4 digits (0..1023)
2	4	ADC CHANNEL 1	1	raw ADC reading from pin 3	up to 4 digits (0..1023)
3	8	ADC CHANNEL 2	1	raw ADC reading from pin 4	up to 4 digits (0..1023)
4	16	ADC CHANNEL 3	1	raw ADC reading from pin 5	up to 4 digits (0..1023)
5	32	ADC CHANNEL 4	1	raw ADC reading from pin 6	up to 4 digits (0..1023)
6	64	ADC CHANNEL 5	1	raw ADC reading from pin 7	up to 4 digits (0..1023)
7	128	RAW GYRO	3	raw gyroscope data (3 axes) (+/- 2000º/s)	+/- up to 5 digits (+/- 32767)
8	256	RAW ACC	3	raw accelerometer data (3 axes) (+/- 2g)	+/- up to 5 digits (+/- 32767)
9	512	RAW COMP	3	raw magnetic values (3 axes) (+/- 1229 uT)	+/- up to 4 digits (+/- 4095)

Bit	Decimal	Name	#	Description	Format
10	1024	CAL GYRO	3	calibrated gyroscope data in dps	up to 8 hexadecimal digits (Q16)
11	2048	CAL ACC	3	calibrated accelerometer data in g's	up to 8 hexadecimal digits (Q16)
12	4096	CAL COMP	3	calibrated magnetic values in microtesla	up to 8 hexadecimal digits (Q16)
13	8192	QUARTERNION	4	9 axis quarternion	up to 8 hexadecimal digits (Q30)
14	16384	EULER ANGLES	4	Euler angles in degrees	up to 8 hexadecimal digits (Q30)
15	32768	ROTATION MATRIX	9	rotation matrix	up to 8 hexadecimal digits (Q30)
16	65536	HEADING	1	heading in degrees	up to 8 hexadecimal digits (Q16)
17	131072	LINEAR ACC	3	linear acceleration in m/s^2	up to 8 hexadecimal digits (Float)
18	262144	IMU STATUS	2	IMU status: temperature in milliC, time in milliSec	+/- up to 5 digits (+/99999)and (+/- 31 bits)
19	524288	PWM_SIGNALS	4	currently set PWM duty	+/- up to 3 digits (+/-

Bit	Decimal	Name	#	Description	Format
				cycles (in uS and %)	1000000 and +/-100)
20	1048576	MOTOR_CURRENTS	2	motor currents from the motor driver (in mA)	up to 4 digits (0..9999)
21	2097152	EVENT_RATE	1	event rate (events per second)	up to 7 digits (0..1000000)
28	268435456	MOTOR_SENSORS	2	wheel tick counter (only present in PushBot)	+/- up to 11 digits(+/- 31 bits)

Custom application sensors should use bits 28 - 31; bits 22 - 27 are reserved for future applications.

Event recording formats

Streaming

You can specify the following formats for data streaming:

```
!E0 - 2 bytes per event, binary: 1yyyyyyy.pxxxxxxx (default) (p = polarity)
!E1 - 3-6 bytes per event; the above address format followed by 1-4 bytes
delta-timestamp (7 bits each)
!E2 - 4 bytes per event (as !E0 followed by 16 bit absolute timestamp)
!E3 - 5 bytes per event (as !E0 followed by 24 bit absolute timestamp)
!E4 - 6 bytes per event (as !E0 followed by 32 bit absolute timestamp)
```

Every timestamp has 1 us resolution.

Examples:

Format	Data packet
!E0 will result in data packets	`1yyyyyyy.pxxxxxxx`

!E1 will result in data packets	`1yyyyyyy.pxxxxxxx.1ttttttt`
	(time stamp wrap-around after 2^7 us = 128 us)
	`1yyyyyyy.pxxxxxxx.0ttttttt.1ttttttt`
	(time stamp wrap-around after 2^14 us = 16 384 =~ 16 ms)

Format	Data packet
	1yyyyyyy.pxxxxxxx.0ttttttt.0ttttttt.1ttttttt (time stamp wrap-around after 2^21 us = 2 097 152 us =~ 2 sec) 1yyyyyyy.pxxxxxxx.0ttttttt.0ttttttt.0ttttttt.1ttttttt (time stamp wrap-around after 2^28 us = 268 435 456 us =~ 4.5 min)
!E2 will result in data packets:	1yyyyyyy.pxxxxxxx.tttttttt.tttttttt (time stamp wrap-around after 2^16 us = 65 535 us =~ 65ms)
!E3 will result in data packets:	1yyyyyyy.pxxxxxxx.tttttttt.tttttttt.tttttttt (time stamp wrap-around after 2^24 = 16 777 216 us =~ 16 sec)
!E3 will result in data packets:	1yyyyyyy.pxxxxxxx.tttttttt.tttttttt.tttttttt.tttttttt (time stamp wrap-around after 2^24 = 4 294 967 296 us =~ 72 min)

Recording

For event recording on SD-card, the format is always as !E1 above, i.e. 1yyyyyyy.pxxxxxxx (p = polarity), followed by 1-4 bytes delta-timestamp (7 bits each); this size depends on how much time has passed (see table below):

Time	Data packet
< 128us:	1yyyyyyy.pxxxxxxx.1ttttttt
< 16384 us:	1yyyyyyy.pxxxxxxx.0ttttttt.1ttttttt
< 2097152 us:	1yyyyyyy.pxxxxxxx.0ttttttt.0ttttttt.1ttttttt
else	1yyyyyyy.pxxxxxxx.0ttttttt.0ttttttt.0ttttttt.1ttttttt

(a leading 1 in a time stamp byte indicates the final byte of time-stamp)

Connectors

The input voltage available in the connectors is the one present before the voltage regulator for the board. Most pins available have secondary functions that can be used if needed.
I2C is available in the PWM outputs connector.

GPIO

Pin	Primary Function	Secondary Function(s)
1	Supply Voltage	
2	GPIO2[5]	PWM Channel A 2
3	GPIO2[6]	PWM Channel A 2
4	GPIO2[2]	PWM Channel C 0
5	GPIO5[12]	Timer2 Capture 2
6	GPIO5[13]	Timer3 Capture 1
7	GPIO5[14]	Timer0 Capture 2
8	GND	

ADC

Pin	Function
1	Analog Supply
2	ADC 0
3	ADC 1
4	ADC 2
5	ADC 3
6	ADC 4
7	ADC 5
8	Analog GND

PWM Outputs

Pin	Primary Function	Secondary Function(s)	
1	Supply Voltage		
2	PWM Channel A 0	TWI SDA	GPIO5[3]
3	PWM Channel A 1	TWI SCK	GPIO5[4]
4	PWM Channel B 0	GPIO5[15]	
5	PWM Channel B 1	GPIO5[16]	
6	PWM Channel C 0	GPIO5[7]	
7	PWM Channel C 1	GPIO1[10]	
8	GND		

SPI

Pin	Primary Function	Secondary Function(s)
1	Supply Voltage	
2	MOSI	GPIO0[9]
3	MISO	GPIO0[8]
4	SCK	
5	SSEL	GPIO0[4]
6	GND	

UART0 (Slave)

Pin	Primary Function	Secondary Function(s)
1	Supply Voltage	

Pin	Primary Function	Secondary Function(s)
2	RXD	GPIO5[1]
3	TXD	GPIO5[0]
4	GND	
5	CTS	GPIO1[8]
6	RTS	GPIO0[10]

UART is already set up in the default firmware. The UART0 port can be accessed from outside and it supports exactly the same command structure that USB supports, since the USB connection is in the end provided by an FTDI chip connected to UART0. This means you can get events from it easily.

UART1 (Master)

Pin	Primary Function	Secondary Function(s)
1	Supply Voltage	
2	TXD	GPIO1[14]
3	RXD	GPIO1[7]
4	GND	
5	RTS	GPIO2[11]
6	CTS	GPIO2[133]