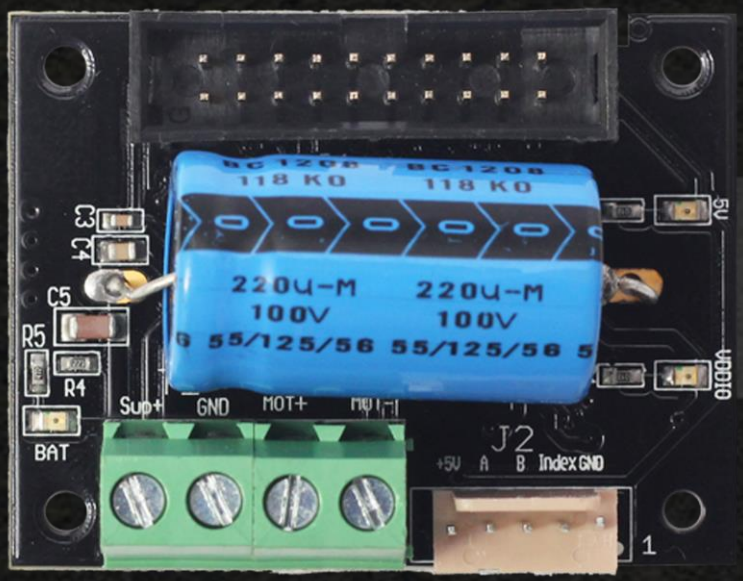
Phi Robotics

Product Manual



Cheetah

Version 1.0

Phi Robotics Research Pvt. Ltd.

www.phi-robotics.com

Table of Contents

1 Introduction

Cheetah is a DC motor driver board; which is interfaced using GPIO and SPI. The board has a L9958 DC motor driver chip, which is a SPI controlled H-Bridge designed for the control of DC motors under extreme condition. The H-Bridge is protected from short circuit and overheating. The board also has a 32-bit quadrature counter for quadrature encoder feedback. The quadrature counter also interfaces with incremental encoders to perform a variety of marker functions.

2 Board Features

- On-board 32 bit quadrature encoder for speed and position feedback chip
- Operating frequency of L9958 is up to 20 kHz
- Overheating and short circuit protection
- Open load detection in on condition
- Enable and disable input
- SPI for configuration
- Daisy chain capability
- X1, X2 or X4 mode of quadrature counting

3 Specifications

- Supply voltage: 12V
- Motor supply voltage: 4 to 28V
- Crystal frequency: 20 MHz
- Current regulation peak threshold: 8.6 A

3.1 PCB Details

- PCB size: 50.8 mm x 38.74 mm
- PCB type: FR4
- Solder mask: Black
- Board thickness: 1.6 mm
- Surface finish: Immersion Gold

.

4 Hardware Connections

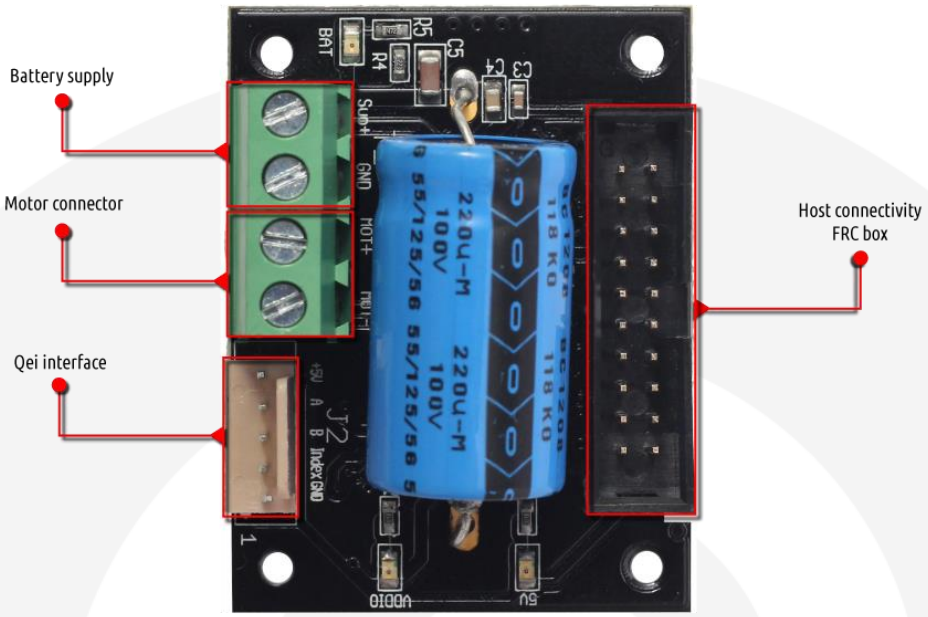


Figure 1 - Cheetah top view

4.1 Host Connectivity

Cheetah has a 10x2 FRC box header (J1) for host microcontroller connectivity. The connector has supply lines, common SPI communication lines for motor driver and quadrature counter, motor driver control lines and a battery voltage monitoring pin. This connector is compatible with Phi Robotics Cheetah-CB board. Figure 2 below shows the pin layout for J1.

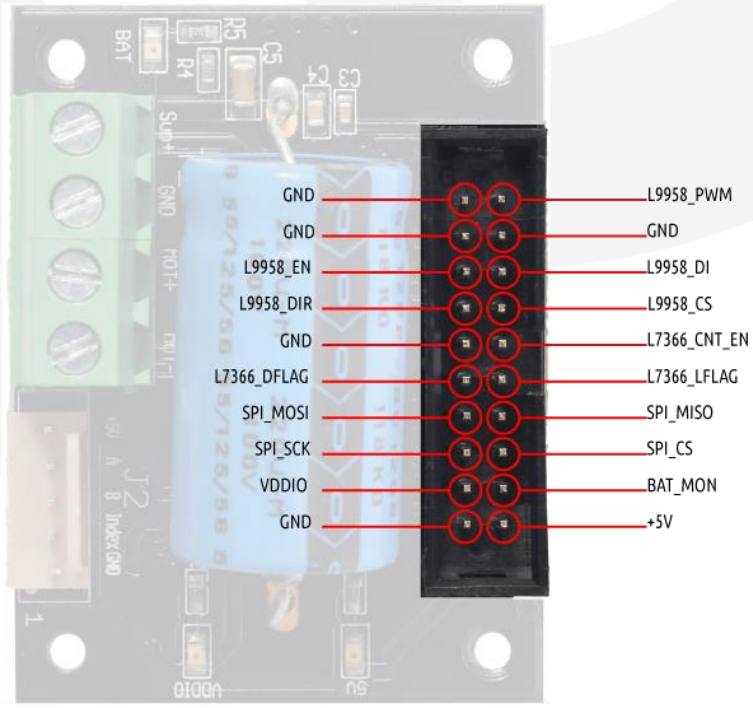


Figure 2 - J1 connector pin layout

4.2 Motor and Battery Connections

Terminal T1 is for motor battery supply and T2 is for motor connections.

4.3 Quadrature Encoder Input

The quadrature counter supports two types of input for motor feedback, quadrature encoder and incremental counter. Figure 3 below shows the connector pin outs for quadrature encoder connector (J2).

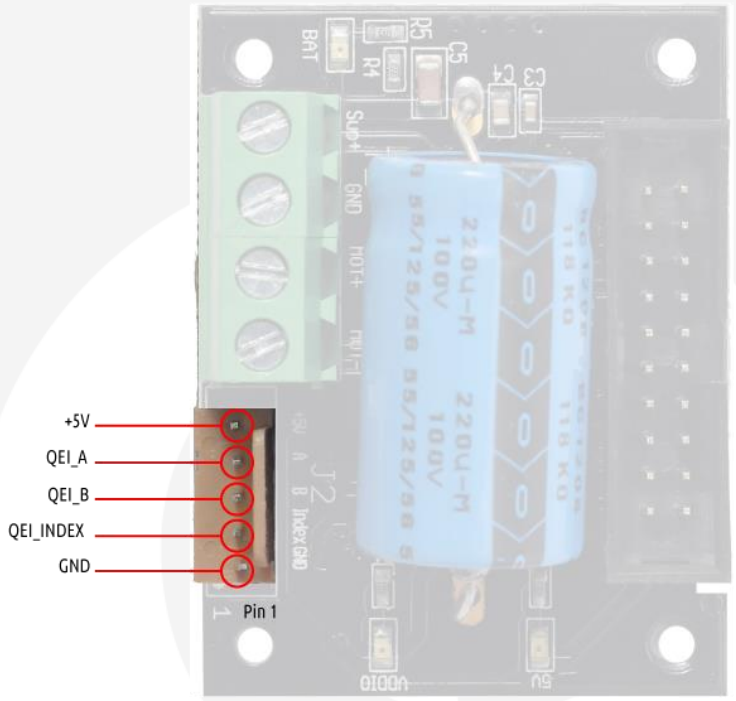


Figure 3 - Quadrature counter connector pin outs

4.4 Indicator LEDs

The Cheetah board has the following indicator LEDs:

- VDDIO LED: Supply voltage for SPI indicator
- 5V LED: Supply voltage indicator
- BAT: Battery supply indicator

5 Pseudo Code

5.1 QEI Commands and Status Flags

```
// QEI instruction masks
QEI_IR_REG_NONE = 0x00
QEI_IR_REG_MDR0 = 0x08
QEI_IR_REG_MDR1 = 0x10
QEI_IR_REG_DTR = 0x18
QEI_IR_REG_CNTR = 0x20
QEI_IR_REG_OTR = 0x28
QEI_IR_REG_STR = 0x30
QEI_IR_ACT_CLR = 0x00
QEI_IR_ACT_RD = 0x40
QEI_IR_ACT_WR = 0x80
QEI_IR_ACT_LOAD = 0xC0

// QEI status flags
QEI_STATUS_SIGN = 0x01
QEI_STATUS_COUNT_DIR = 0x02
QEI_STATUS_PLS = 0x04
QEI_STATUS_COUNT_EN = 0x08
QEI_STATUS_INDEX = 0x10
QEI_STATUS_COMPARE = 0x20
QEI_STATUS_BORROW = 0x40
QEI_STATUS_CARRY = 0x80
```

5.2 QEI Configuration Parameters

```
// QEI configuration parameters
// quadrature mode options: non-quadrature, 1x, 2x, 4x
QEI_QUAD_MODE = 0x01    // quadrature mode set to 1x
// count limit options: free running, single cycle,
//                      range count, module-n mode
QEI_COUNT_LIMIT = 0x00  // count limit set to free running
// index mode options: disable, load CNTR, reset CNTR, load OTR mode
QEI_INDEX_MODE = 0x00   // index mode disabled
// filter clock divider options: divide by 1, divide by 2
QEI_FILTER_CLK_DIV = 0x00     // filter clock divider set to 1
// index pin synchronous mode: enabled when index input is available
QEI_INDEX_PIN_SYNC = 0x40     // index pin set as asynchronous

// counter register byte size options: 1, 2, 3, 4
QEI_COUNTER_SIZE = 0x00 // counter register byte size set to 4
// flag on index latch options: enable, disable
QEI_IDX_FLAG = 0x00     // disabled
// flag on borrow options: enable, disable
QEI_BW_FLAG = 0x00      // disabled
// flag on compare options: enable, disable
QEI_CMP_FLAG = 0x00     // disabled
// flag on carry options: enable, disable
QEI_CY_FLAG = 0x00      // disabled

// counter register byte size
COUNTER_REG_SIZE = 4
```

5.3 QEI Register Access and Initialization

```
void qeiInit()
{
	uint32_t mdr0_val, mdr1_val;
	// disable qei counter before initialisation
	gpioClear(QEI_COUNT_EN);

	mdr0_val = QEI_QUAD_MODE | QEI_COUNT_LIMIT
			| QEI_INDEX_MODE | QEI_FILTER_CLK_DIV
			| QEI_INDEX_PIN_SYNC;

	mdr1_val = QEI_COUNTER_SIZE | QEI_IDX_FLAG
			| QEI_BW_FLAG | QEI_CMP_FLAG | QEI_CY_FLAG;

	gpioSet(QEI_COUNT_EN);				// enable counter
	qeiRegWrite(QEI_IR_REG_MDR0, mdr0); // setting MDR0 value
	qeiRegWrite(QEI_IR_REG_MDR1, mdr1); // setting MDR1 value
	// reset count
	spiWriteByte(QEI_IR_ACT_CLR | QEI_IR_REG_CNTR);
}

void qeiRegLoad(uint8_t reg)
{
	uint8_t cmd;
	// if reg = CNTR, DTR value will be loaded to CNTR
	// if reg = OTR, CNTR value will be loaded to OTR
	cmd = LS7366_IR_ACT_LOAD | reg;
	spiChipSelect(0);
	spiWriteByte(cmd);
	spiChipSelect(1);
}

void qeiRegWrite(uint8_t reg, uint32_t data)
{
	uint8_t writeSize, writeBuff[] ;
	uint8_t cmd = QEI_IR_ACT_WR | reg;

	if((reg == QEI_IR_REG_MDR0) || (reg == QEI_IR_REG_MDR1)
		|| (reg == QEI_IR_REG_STR))
	{
		// MDR0, MD1 and STR are 8 bit register
		writeSize = 1;
		writeBuff[0] = data;
	}
	else
	{
		// DTR, OTR and CNTR register size is configurable
		writeSize = COUNTER_REG_SIZE;
		loop(index 0 to writeSize)
		{
			// copying uint32_t value to uint8_t array
			// higher byte needs to sent first
			writeBuff[index] = data >> ((writeSize - (index+1))*8);
		}
	}
	spiChipSelect(0);
	spiWriteByte(cmd);					// send command
	spiWrite(writeBuff, writeSize);		// send data
	spiChipSelect(1);
}
```

```
uint32_t qeiRegRead(uint8_t reg)
{
	uint8_t cmd, readSize, index;
	uint8_t readBuff[4] = {0};

	cmd = QEI_IR_ACT_RD | reg;
	if((reg == QEI_IR_REG_MDR0) || (reg == QEI_IR_REG_MDR1)
		|| (reg == QEI_IR_REG_STR))
		readSize = 1;
	else
		readSize = COUNTER_REG_SIZE;

	spiChipSelect(0);
	spiWriteByte(cmd);
	spiRead(readBuff, readSize);
	spiChipSelect(1);

	if((reg == QEI_IR_REG_MDR0) || (reg == QEI_IR_REG_MDR1)
		|| (reg == QEI_IR_REG_STR))
	{
		data = readBuff[0];
	}
	else
	{
		data = 0;
		for(index = 0; index < readSize; index++)
		{
			// higher byte is read first
			data |= readBuff[index] << ((readSize - (index+1))*8);
		}
	}
	return data;
}
```

5.4 Reading Encoder Count

```
uint32_t qeiReadCount(void)
{
	uint32_t count;
	// load count in OTR register
	qeiRegLoad(LS7366_IR_REG_OTR);
	// read output register (OTR) to get count
	count = qeiRegRead(LS7366_IR_REG_OTR);
}
```

5.5 Enable and Disable Cheetah

```
void cheetahEnable(void)
{
	gpioSet(MOTOR_DRV_EN);		// set the motor driver enable pin
	gpioClear(MOTOR_DRV_DIS);	// clear the motor driver disable pin
	gpioSet(QEI_COUNT_EN);		// set qei enable pin
	spiWriteByte( QEI_IR_ACT_CLR | QEI_IR_REG_CNTR); // reset count
}

void cheetahDisable(void)
{
	gpioClear(MOTOR_DRV_EN);	// clear the motor driver enable pin
	gpioSet(MOTOR_DRV_DIS);		// set the motor driver disable pin
	gpioClear(QEI_COUNT_EN);	// clear qei enable pin
}
```

5.6 Rotating the Motor

```
void cheetahSetDirection(uint8_t dir)
{
	if(dir == clockwise)
		gpioClear(MOTOR_DRV_DIR);
	else
		gpioSet(MOTOR_DRV_DIR);
}

void cheetahMove(uint8_t dir, uint32_t speed)
{
	uint32_t pwmVal
	cheetahSetDirection(dir);

	// user should calculate the value to update in the PWM register
	// from given speed, according to the PWM configuration
	pwmVal = calcPwmVal(speed);
	// set PWM value
	pwmUpdate(pwmVal);
}
```

5.7 Battery Voltage Monitoring

```
uint32_t cheetahGetBatteryVoltage(void)
{
	uint32_t adcVtg_mv, battVtg_mv;

	// get the value from ADC in millivolts
	adcVtg_mv = adcGetVoltage();

	// convert sampled ADC voltage value into battery voltage
	// battery voltage is sampled through resistor divider of 10K-1K
	// since we are measuring voltage across 1K resistor,
	// we need to multiply ADC voltage value by 11
	battVtg_mv = adcVtg_mv * 11;

	return battVtg_mv;
}
```

6 References

LS7366R-S Datasheet: http://www.lsicsi.com/counters.htm

L9958 Datasheet: http://www.st.com/web/catalog/sense_power/FM1965/SC1039/PF246497

Phi Robotics Research Pvt. Ltd.

www.phi-robotics.com