E-Bike Li-Ion Akku durch LiFePO4 Akku ersetzen

Inhaltsverzeichnis

1 Vorwort

Leider hat mein TranzX BL03 Akku im Sept. 2023 angefangen zu schwächeln.
Symptome: Reichweite zurückgegangen und dann war er aus nicht nachvollziehbaren Gründen tiefentladen.
Ich habe ihn dann mal kurz geladen und nach einigen Minuten waren wieder ca. 30V Spannung da.
BMS ausgelesen → meldet eine Restkapazität von 60%.
Nochmal ein bisschen geladen bis Akkuspannung 35V. → BMS meldet Restkapazität von 9%. OK jetzt hat das BMS gemerkt dass der Akku fast leer ist!
Laden funktioniert noch, fahren geht auch wieder, aber ich habe Angst dass es zu einem Akkubrand kommen könnte wenn ich länger auflade. Der Akku ist auch schon ‚alt': ca. 8Jahre.

Das zeigt mir dass ich weg möchte von diesen brandgefährlichen Li-Ionen Akkus!
Als Alternative bin ich auf LiFePO4 Akku Technologie gestoßen. Diese hat folgende Vorteile gegenüber Li-Ion Technik:

- LiFePO4 Akkus **gehen nicht thermisch durch**, d.h. **keine Brandgefahr**
- sie erlauben deutlich mehr Ladezyklen
- längere Lebensdauer
- unempfindlicher gegen Hitze und Kälte (Entladetemperatur -20 .. 60°C)
- weniger Gefahrstoffe
- günstigerer Preis

Nachteile sind:

- mehr Gewicht / Volumen
- es gibt keinen passenden Ebike Akku zu kaufen

Also selber was bauen!

1.1 Herausforderungen

- LiFePO4 Akkus mit etwa gleicher Kapazität finden wie mein alter Akku
- Bauform finden die ich auf meinem Fahrrad unterbringe
- Die vorhandene Steuerung möchte mit dem Akku(genau genommen mit dem Batterie Management - BMS) kommunizieren. D.h. diese BMS Funktion muss nachgebildet werden.

1.2 Lösungsideen

1.2.1 Akku

Ich habe mich für eine Kombination von 3 LiFePO Modulen mit 12,8V/10Ah entschieden. Es gibt verschiedene Anbieter. Ein Modul hat die Abmessung

mit 152 x 66 x 94 mm.

2 davon bringe ich anstelle des alten Akkus unter, der dritte muss woanders hin; vielleicht vor den Gepäckkoffer.

1.2.2 BMS Nachbildung

Die LiFePO Module haben zwar ein BMS – aber da komme ich nicht ran. Außerdem weiß ich nicht ob dieses BMS eine passende Kommunikationsschnittstelle hätte.

Da ich schon einige Elektronik-Projekte mit Arduino realisiert habe, mein Entschluss: auch hier kommt ein Arduino rein!

1.2.3 Gesamtes Fahrzeug

Der Aufbau soll so erfolgen, dass das Fahrzeug einfach wieder auf den originalen Akku zurück gebaut werden kann.

2 Hardware

2.1 Anforderungen

2.1.1 Mikrocontroller

- Es gibt Arduino Boards im Arduino Nano Format mit unterschiedlichen Prozessoren
- ATmega168, ATmega328 oder LGT8F328
- LGT8F328 ist Favorit. Niedriger Stromverbrauch, günstiger Preis, hohe Leistung – leider kein echtes EEPROM; muss noch getestet werden ob die EEPROM Emulation die Zwecke erfüllt.

Was wird benötigt:

	Benötigt	ATmega168	ATmega328	LGT8F328
I²C Schnittstelle	1	1	1	1
ADC Kanäle	6	6	6	7
DIO mit PWM	1 (1 Timer)		6 (3 Timer)	8 (4 Timer)
USB	1	1	1	1
EEPROM	?		1kByte	Simulated E2PROM

2.1.2 Spannungsversorgung

- Die Elektronik soll aus der Batteriespannung versorgt werden
- Die maximale Batteriespannung beträgt 42 Volt
- Es soll verhindert werden dass die Batterie von der Elektronik ‚leer gesaugt‘ wird. Abschaltbar machen
- Absicherung gegen Kurzschluss und Verpolen?

2.1.3 Messung von Strömen

Die Ströme werden über jeweils einen Shunt-Widerstand geführt. Die dort abfallende Spannung wird mit einem Operationsverstärker so verstärkt, dass der zu erwartende Messbereich mit dem Messbereich des ADC (5V) übereinstimmt.

2.1.3.1 Ladestrom

Messbereich: 0 … 2A + Sicherheit.

An einem 33mOhm Shunt würden da 66mV abfallen → das wären 0,132W. Verstärkung <75
An einem 16,5mOhm Shunt würden da 33mV abfallen → das wären 0,066W. Verstärkung <150

2.1.3.2 Entladestrom

Messbereich: 250W / 36V = 6,95A + Sicherheit also 0 … 7A

An einem 11mOhm Shunt würden da 77mV abfallen → das wären 0,539W. Verstärkung <64

2.1.4 Messung von Spannungen

2.1.4.1 Batteriespannung 36V

Gehen wir von der maximal zulässigen Spannung über alle 3 Batteriemodulen aus errechnet sich eine Spannung von 3x14,6V = 43,8V. Mein Ladegerät begrenzt die Ladespannung laut spec. Auf 42V. Die gemessene Ladeschlussspannung liegt sogar noch niedriger; d.h. ich kann nicht die ganze Kapazität nutzen.

Die Batteriespannung wird über einen Widerstandsteiler herunter geteilt und an den ADC geführt. Da fließt dann natürlich immer ein kleiner Strom wodurch der Akku entladen wird. Um die Entladung gering zu halten müssen hohe Widerstandswerte verwendet werden.

Messbereich: 0 … 42V

Spannungsteiler 42V → <5V damit ein Verhältnis von (42-5)/5 = 7,4; z.B. 680k/<91k (also z.B. 47k+39k=86k
wenn das dauerhaft anliegt saugt das natürlich den Akku leer, aber sehr langsam I=36V/680k=53µA. Bei 7Ah dauert das dann 132000 Stunden.

2.1.4.2 Batteriezwischenspannung 24V

Spannung über Module 1 und 2. Wird benötigt um zu erkennen ob die Batteriemodule unterschiedliche geladen sind. Eine Spannung von 2x14,6V = 29,2V.

2.1.4.3 Batteriezwischenspannung 12V

Spannung über Modul 1. Wird benötigt um zu erkennen ob die Batteriemodule unterschiedliche geladen sind. Eine Spannung von 1x14,6V = 14,6V.

2.1.4.4 Schaltregler Ausgangsspannung

Wenn der Schaltregler nicht läuft liegt hier die Batteriespannung. Also ebenfalls etwa 42Vmax.

2.1.5 5V Spannungsversorgung für Arduino

Der Arduino hat zwar einen Spannungsregler onBoard. Die Eingangsspannung darf aber maximal 12V sein. Die Batteriespannung beträgt aber bis zu 42V. Ein 5V Längsregler mit Eingangsspannung bis 42V muss verwendet werden. Meine Wahl TLE4271.

2.1.6 Einfacher Schaltregler (optional)

Zur Reduzierung des Strombedarfs dieser Elektronik kann dem 5V Längsregler ein Schaltregler vorgeschaltet werden.

2.1.7 Selbstabschaltung

Wenn erkannt wird dass das Fahrrad ausgeschaltet wurde soll sich diese Elektronik selbst abschalten um die Batterieentladung zu reduzieren. Dafür muss ein Transistor eingebaut werden welcher über einen Mikrocontroller Pin ein- bzw. ausgeschaltet werden kann.

Zusätzlich wird ein Taster benötigt, welcher den Transistor überbrückt. Zum Einschalten muss der Taster so lange gedrückt werden bis die Software gestartet ist und den Transistor eingeschaltet hat.

2.1.8 Kommunikation mit der Motorsteuerung

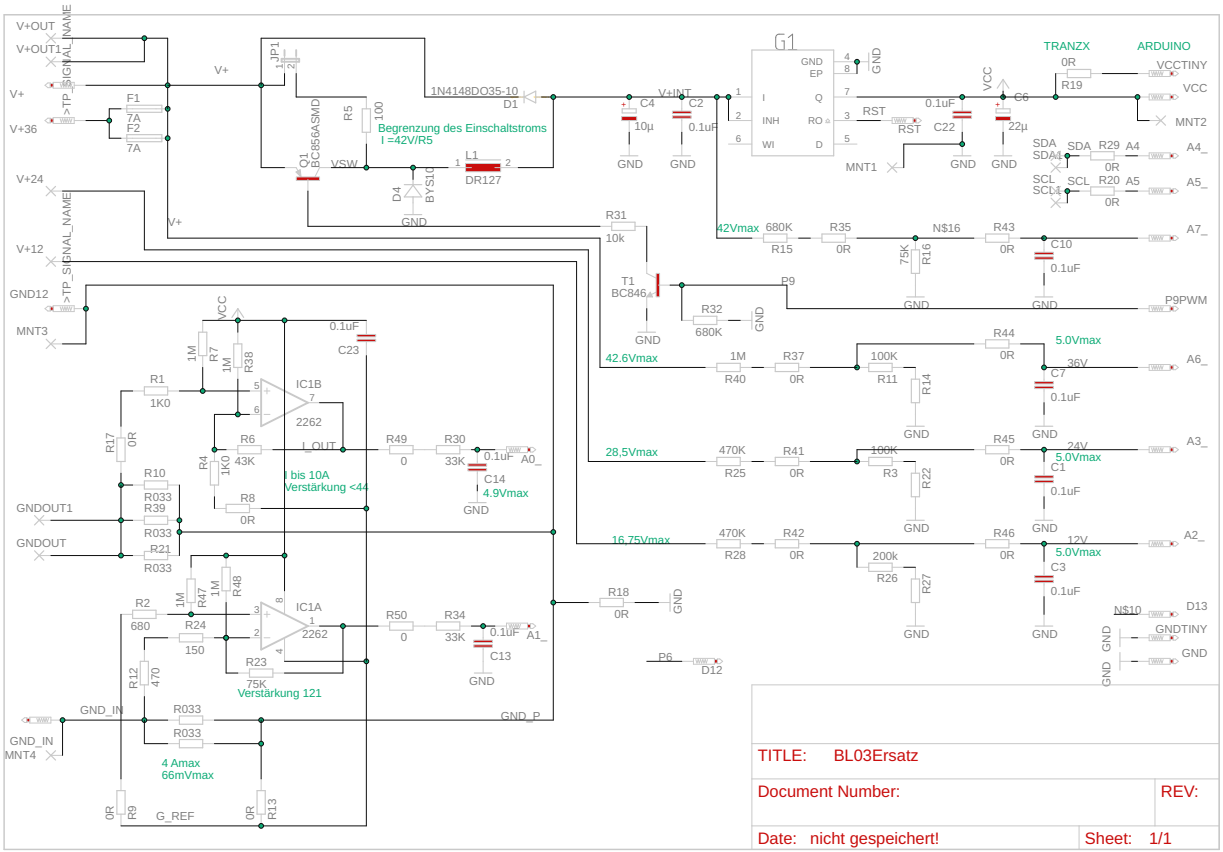
Erfolgt über die I²C Schnittstelle. Die dafür benötigten Pins SDA und SCL sind die gleichen wie für die ADC Kanäle A4 und A5; diese ADC Kanäle können somit nicht genutzt werden.

2.1.9 EEPROM

Einige Werte sollen nicht verloren gehen, wenn die Elektronik abgeschaltet wird. Diese können in einem EEPROM gespeichert werden. → Ausgewählter Mikroprozessor soll ein EEPROM enthalten.

2.2 Umsetzung

Schaltplan und Layout wurden mit der Software ‚Eagle' erstellt



3 Software

3.1 Anforderungen

3.1.1 Betriebssystem

3.1.1.1 Zyklische Programmausführung

Ein einfaches Zeitscheibensystem soll Aufgaben erledigen die in einem festen Zeitraster durchgeführt werden müssen:

In der 250ms Zeitscheibe werden die ADC Werte erfasst und für Mittelwertbildung aufaddiert.

In der 1000ms Zeitscheibe werden aus den Mittelwerten physikalische Werte berechnet.

In der verbleibenden Zeit (background task) werden nicht zyklische Aufgaben erledigt

3.1.1.2 Interrupt getriebene Programmausführung

Zusätzlich werden Interrupt Routinen verwendet, wenn eine Anfrage von der Motorsteuerung auf dem I²C Bus kommt. Dann müssen wir die Anfrage beantworten. Das läuft asynchron zu den Zeitscheiben. In den Interrupt Routinen keine Ausgaben zur seriellen Schnittstelle – führt zu Datenmüll. Nur ein Flag setzen und die Ausgabe erfolgt dann in der background task.

3.1.1.3 Sonstiges

Hintergrund: Der TI Controller bq2931, welcher vermutlich im Original Akku verwendet wird, erfasst 4 Werte pro Sekunde und bildet daraus einmal pro Sekunde einen Mittelwert. Auch die Kommunikation über I²C Bus ist in den Datenblättern dieses Bausteins beschrieben.

Optional: Um Strom zu sparen könnte der Prozessor in den Sleep mode gehen wenn es nichts zu tun gibt. Das klappt jedoch noch nicht – also erst mal ohne sleep. Auch ein Schaltregler kann Strom sparen; die dafür benötigte PWM Ansteuerung kann im Prozessor erfolgen.

3.1.1.4 Ausgabe von Messwerten auf dem USB Bus

Zur Funktionskontrolle und Debugging sollen verschiedene Werte auf dem USB Bus gesendet werden.

3.1.1.5 Nachlauf

Die Steuerung soll sich abschalten wenn erkannt wird dass sich die E-bike Steuerung abgeschaltet hat und eine Nachlaufzeit abgelaufen ist. Vor der Abschaltung werden EEPROM Werte gespeichert und eine Meldung auf Serial ausgegeben.

3.1.2 Ermittlung von Messgrößen

3.1.2.1 Erfassen von Analogwerten

Die folgenden Werte sollen über die Analogeingänge erfasst werden:

- Ladestrom

- Entladestrom
- Batteriespannung 36V
- Batteriespannung 24V
- Batteriespannung 12V
- Interne Spannung

Pro Sekunde werden jeweils 4 Analogwerte erfasst. Aus dem Mittelwert dieser 4 Analogwerte wird dann 1x pro Sekunde der physikalische Wert bestimmt.

3.1.2.2 Ermittlung weiterer Größen

- Batterieladung (Capacity)
  Dieser Wert wird aus dem Ladestrom, Entladestrom und der Zeit ermittelt. Der Wert wird vom Motorsteuergerät abgefragt. Der Wert muss nicht flüchtig (im EEPROM) gespeichert werden
- Batteriestatus
  hier sind mehrere Statusbits zusammengefasst. Der Wert wird vom Motorsteuergerät abgefragt.

3.1.3 Kommunikation mit der Motorsteuerung

Die Kommunikation erfolgt über die I²C-Bus Schnittstelle. Der verwendete Mikrocontroller hat entsprechende Hardware integriert.

Die folgenden I²C Botschaften werden vom Motorsteuergerät abgefragt und müssen beantwortet werden: Details stammen aus den Datenblättern des IC bq20z80A von TexasInstruments

3.1.3.1 RelativeStateOfCharge(0x0d)

This read-word function returns an unsigned integer value of the predicted remaining battery capacity expressed as a percentage of FullChargeCapacity, in %, with a range of 0% to 100%, with any fractions of % rounded up.

CMD	Mode	Name	Format	Size in Bytes	Min Value	Max Value	Default	Unit
0x0d	Read	RelativeStateOfCharge	uint	1	0	100	-	%

Related Variable: FullChargeCapacity(0x10)

3.1.3.2 BatteryStatus(0x16)

This read-word function returns the status of the battery

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
High byte	OCA	TCA	reserved	OTA	TDA	reserved	RCA	RTA
Low byte	INIT	DSG	FC	FD	EC3	EC2	EC1	EC0

OCA — Over Charged Alarm

TCA — Terminate Charge Alarm

OTA — Overtemperature Alarm

TDA — Terminate Discharge Alarm

RCA — Remaining Capacity Alarm; see SBS:RemainingCapacityAlarm(0x01) Section A.2.

RTA — Remaining Time Alarm; see SBS:RemainingTimeAlarm(0x02), Section A.3

INIT— Initialized. This flag is set after device reset, after all parameters have been measured and updated.

DSG — Discharging

0 = bq20z80A is in charging mode

1 = bq20z80A is in discharging mode, relaxation mode or valid charge termination has occurred

FC— 1 = Fully Charged

FD— 1 = Fully Discharged

EC3, EC2, EC1, EC0 — Error code, returns status of processed SBS function

0,0,0,0 = OK bq20z80A processed the function code with no errors detected.

0,0,0,1 = BUSY bq20z80A is unable to process the function code at this time.

0,0,1,0 = Reserved bq20z80A detected an attempt to read or write to a function code reserved by this version of the specification or bq20z80A detected an attempt to access an unsupported optional manufacturer function code.

0,0,1,1 = Unsupported bq20z80A does not support this function code as defined in this version of the specification.

0,1,0,0 = AccessDenied bq20z80A detected an attempt to write to a read-only function code.

0,1,0,1 = Over/Underflow bq20z80A detected a data overflow or underflow.

0,1,1,0 = BadSize bq20z80A detected an attempt to write to a function code with an incorrect data block.

0,1,1,1 = UnknownError bq20z80A detected an unidentifiable error.

3.1.3.3 Weitere Botschaften

Sind beschrieben, aber optional, da von der Motorsteuerung nicht abgefragt.

3.1.4 Kommunikation mit USB

Zur Funktionskontrolle und Debugging werden verschiedene Werte auf dem USB Bus gesendet.

3.2 Umsetzung

Der Source Code wird auf GitHub gespeichert.

4 Konstruktion