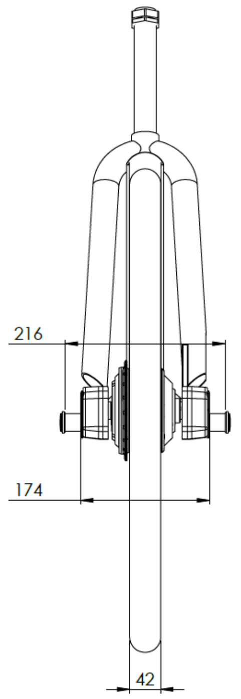


Docking point - Open interface

This document describes the Open interface offered by a corporation between Kimaldi and KT steel and Tech Mobility.

This document is targeted bike designers that would like to design compatible bikes to this open docking point interface.

.

Contents

1. Introduction

This document provides mechanical information about the docking points installed in Stavanger, Copenhagen, Utrecht and Rotterdam. The information is provided to a detailed level in order to allow third party to interface to the docking station and charging infrastructure.

Background

In order to accept the bike into the docking station it needs to have the right dimensions and need to be able to accept the provided charging current and be able to communicate on FSK.

This document outlines all necessary information in order to design a suitable bike solution.

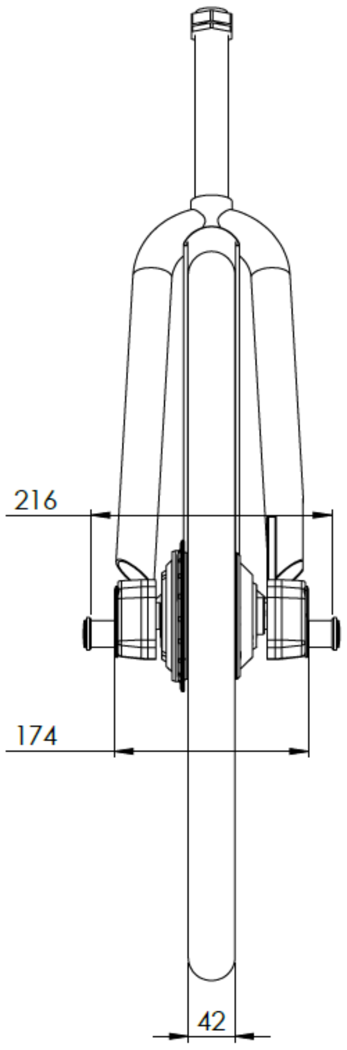
Readers guide

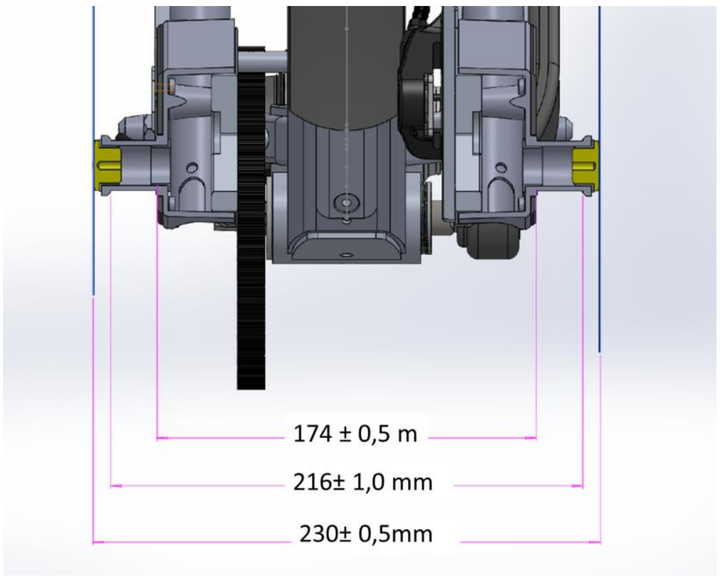
The document is divided into 3 parts:

- Section 2: Mechanical interface
- Section 3: Electrical Interface
- Section 4: Protocol interface

2. Hardware interface

2.1. Front fork – Physical dimensions





2.2. Maximum width of front fork

The docking point entrance is 177 mm wide. The measurements 174 mm is the maximum width measured, at the wides point on the plastic parts. The plastic part shall avoid metal against metal when the bike is entered into the docking point.

2.3. Maximum width at contact points

At the centre of the contact points the measurements shall be 230mm.

2.4. Tire width

The entrance to the docking point is 50 mm. Therefore, it is not recommended to use tires that are above 48mm width. In the drawing above 42 mm is shown. This is the normal for a standard tire.

2.5. Spare parts examples

The interface contains basically of 3 main parts (on each side of the bike):

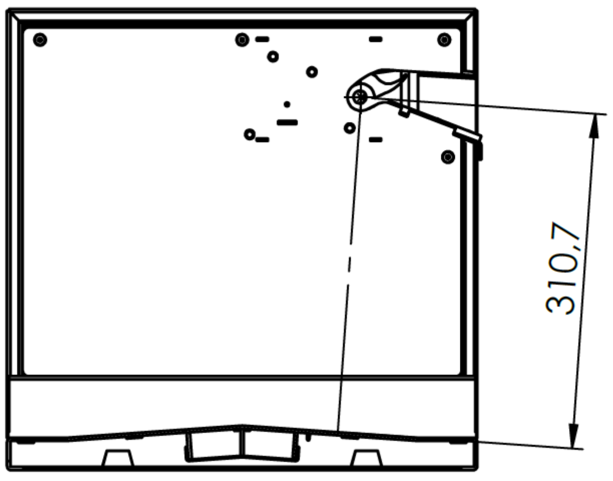
- The Physical frame part, installed on the front fork that provides the counterpart for the locking mechanism in the docking point.
- Two brass parts (connected by a brass threaded rod) and
- Two plastic Isolation

Further details may be found in appendix 3 to this document.

2.6. Distance from centre of contact point to ground

The distance from the centre of the ground is 310.7 mm. However, it is required that the bike is "hanging" in the docking point to ensure that the bike detect switch is working correctly. During autumn leaves might fill up the docking point. Empiric from operation suggests 10 mm free space required under the tire to ensure proper operation. Therefore, 300 mm is the recommended distance from centre of the brass contact to ground.

See drawing below.



SECTION E-E

2.7. Handlebar

The physical distance between the bikes when docked is 640mm. However, the docking points may be installed on a none-flat area and the bike might lean to one or the other side.

Maximum width is, therefore, recommended to be: max. 600 mm. Note that this is also the limit for the brake handles and especially the cables must be kept well width-in this limit. If the cables and other parts of the bike exceed the maximum width, then it might lead to un-intended vandalism when pulling out the bike.

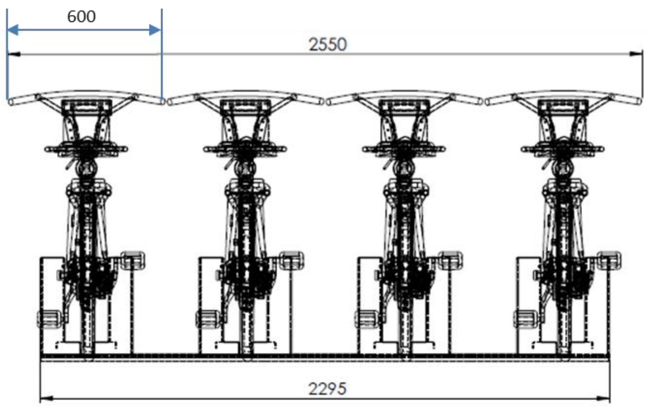


Figure: All handlebar elements must be well with-in the max. handlebar size to avoid unintended vandalism.

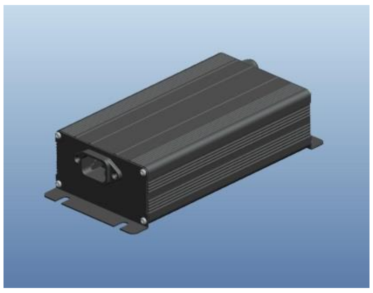
3. Electrical interface

The docking points supply 42V charging power max. 3.2 A. The battery charger, and charging curve, is described in Appendix 1: BPC-L10S03G2.PDF

The current version only supports charging of 42V lithium-ion batteries. The charging curve is shown in the appendix.

The left side charging pole always provides ground and the right side + tension.

The BikePCB, SmartBox and FSK add-on (piggyback) are all designed to accept alternated cables (Should the bike mechanics alternate the charging cables).



3.1. Isolation class 2

Docking stations are designed as isolation class 2. However, normally the metal frame is connected to ground in order to avoid the felling of leakage current on metal surfaces.

3.2. Floating frame

It is recommended to design the bike with a floating frame. That is a frame without connection to either ground or positive on the battery.

3.3. Plus and minus charging poles

The docking point provides negative charging tension from left side and Positive on the right side. However, it is recommended that the left and right charging cables can be interchanged without electrical circus being destroyed (Use of Diode Bridge on bike).

4. Protocol interface

When the bike enter into the docking point it will locked automatically, provided the FSK module answer correctly within the first 10 seconds it will remain locked and charging may be started.

4.1. Protocol specification

The High level protocol is found in appendix 4.

5. Appendix

Appendix 1: BPC-L10S03G2.PDF

Appendix 2: KT Drawing 14905-DCK4PT-01.pdf

Appendix 3: Examples of Mechanical parts.pdf

Appendix 4: Kimaldi Manual version 1.30.pdf

6. Document version control

Version	Date	Author	Description
1	14 Nov. 2017	PSA	Initial document
2	20 Mar. 2018	PSA	Handlebar max size underlined with new image.

Industrial Graded Lithium Ion Battery Pack Charger

(Model: BPC-L10S03G2)

April 7, 2014

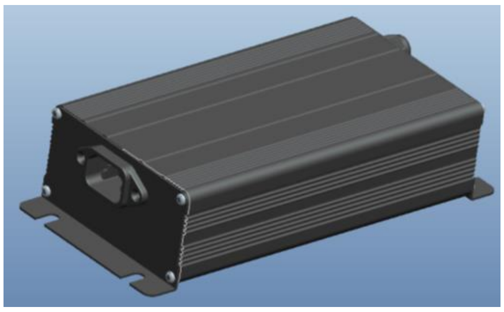
Draft 1C

REVISION HISTORY

Date	Versions #	Revision Items
Feb. 27, 2014	Draft 1A	Initial draft based on a 42.8VDC/3.2A specification
April 2 , 2014	Draft 1B	Updated the stand by power from 1.5A continue to 1A continue , 2A peak for 5 seconds.
April 7 , 2014	Draft 1C	Updated the 7.5 mechanical layout and 12 decal for Label.

Table of Contents

1. Introduction



This document is to specify an industrial graded lithium ion battery pack charger (model: BPC-L10S03G2) for GoBike application. The charger is cooled by natural air (without an internal fan) and it comes with aluminum housing with IP54 waterproof rating. It has several built-in protection functions, for examples, output current protection, short circuit protection, reverse protection, etc. In addition, it supports auto AC power detection without manual switch-over between 100VAC and 240VAC. The unique function of this charger is that it can provide dual DC powers. The primary power source is operated at maximum of 42.8VDC/3.2A and its main function is to charge a lithium ion battery pack. The charging status can be visualized via a single LED indicator with 3 different colors. The second power source provides an uninterrupted power which is operated at 12VDC/1A. No LED indicator is available for the 12VDC power. Both primary and secondary power sources can be operated simultaneously. The charger can be operated between -20°C to 40°C. When storing at -25°C, it will not malfunction after temperature return to -20°C.

2. Main product specification

Input voltage range	**Output voltage**	**Output current**	**Max. output power**	**Control Status**
90-264VAC	42.8 +/- 0.2VDC	3.2A+/-0.3A	150W	Controlled by I/O
	12 +/- 2 VDC	1A continue , 2A peak for 5 seconds	24W	Always On

3. Environmental conditions

No.	Items	Ratings	Units	Remarks
1	Operating temperature	-20 to +40°C，typical value 25°C (Note: Surface temperature of the charger can be operated at 60 degrees C)	°C	In fully loaded state for four hours, it will show error message
2	Storage temperature	-25 to +70°C，typical value 25°C	°C	Under -25°C the charger must not start or only start at the temperature where it would work properly
3	Humidity	5%－95%		
4	Altitude	≤3000	m	Work normally,
5	Cooling	Fan less		

4. Electrical characteristics

4.1 Input characteristics

No.	Items	Ratings	Units	Remarks
4.1.1	Rated input voltage	100-240	VAC	Full range
4.1.2	Input voltage range	90-264	VAC	
4.1.3	AC input voltage frequency	47~63	Hz	
4.1.4	Inrush current (less than 0.001 sec)	75A (nominal) 100A (maximum)	A	264VAC input/start-up in cold condition /environmental temperature is 25 °C
4.1.5	Max input current	3	A	Vin=90VAC, rated load
4.1.6	Power factor	＞0.9		

4.2 Output characteristics

No.	Items	Ratings	Units	Remarks
4.2.1	charging voltage	20 to 42.8	VDC	Base on 0.1A load
4.2.3	Constant current	3.2±0.3	A	
4.2.4	Cross regulation	±0.2	VDC	

4.2.5	Power efficiency	≥80%		Vin=220VAC,rated load
4.2.6	I/O pin (refer to Section 7.2: 42VDC + 12VDC Connector)	High : above 4V Low : below 2V	Vdc	When voltage above 4V, stop charging , when voltage below 2V,start charging

4.3 Protection characteristics

No.	Items	Ratings	Units	Remarks
4.3.1	Output Reverse Protection	The charger is electronically protected against permanent reverse battery connection		Fuse cut off
4.3.2	Over voltage protection	>46	V	Cut off
4.3.3	Over Temperature protection	Charger internal temperature protection		< -20°C cut off. > 75°C cut off
4.3.4	Output short circuit protection	When an external fault is applied to the charger power supply, such that short circuit is applied to the output, the charger power supply shall shut down.		Shut down and no damage.
4.3.5	Input fuse on the AC end	250VAC/5A		

4.4 LED indicator

No.	Charging Status	Status LED		Remarks
1	Stand by	Steady Green		I/O high **:** 42V cut off
2	In progress	Flash Yellow		I/O low **:** 42V output
3	Error Status	Over Current	Flash Red	I/O pull high for one second then restore
		Over Voltage		I/O pull high for one second then restore
		Over high temperature		I/O pull high for one second then restore
		Over low temperature		I/O pull high for one second then restore
		Time out/ voltage below 20V/short cut/stand by power short cut		I/O pull high for one second then restore

5. Safety & EMC

No.	Items		Standard or testing conditions	Remarks
1	Eelectrical strength test	primary—secondary	1500Vac/10mA/1min	No breakdown
2	Isolation resistance	Input—Output	≥10MΩ@500Vdc	
3	SAFETY			Comply with CE standard
4	Leakage current		<300Ua	Vin=264Vac,50~60Hz
5		RE	CLASS B	EN55014/EN55014
		CE	CLASS B	EN55014/EN55014
		Air discharge	LEVEL 3	EN61000-4-2(discriminationＢ)
		Contact discharge	LEVEL 3	EN61000-4-2(discriminationＢ)
		RS	LEVEL 3	EN61000-4-3(discriminationＡ)
		CS	LEVEL 3	EN61000-4-6(discriminationＡ)
		EFT	LEVEL 3	EN61000-4-4 (discrimination B)
		Surge	LEVEL 3	EN61000-4-5, differential module□KV, common module 2KV(discriminationＢ)
		Voltage Dips and Interruptions	LEVEL 3	EN61000-4-11
		Harmonic Current emission	CLASS A	EN61000-3-2
		Voltage Fluctuation and Flicker	CLAUSE 5	EN61000-3-3
		CE(LVD)	CLASS B	EN60335-1/ EN60335-2-29
		EMF	CLASS B	EN62233

Remark: discrimination A: function OK under technical requirement range; discrimination B: function temporarily debasement without reposition and halt is allowed; discrimination R: physical damage or failure of equipment are not allowed, but damage of protection device (fuse) caused by interference signal of outside is allowed, and the whole equipment can work normally after replacement of protection device and reset of running parameter.

5.1 Regulation Compliance

This product must compliance with following regulations:

CE (EMC+LVD) and EN15194 (where applicable)

6. Environmental testing requirements

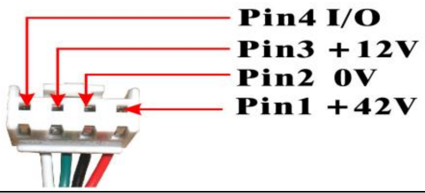
No.	Items	Technical specification	Remarks
1	High temperature operating	+40°C (note: surface temperature can be operated at 60 degrees C)	Features ok
2	Low temperature operating	-20°C	Features ok
3	High temperature storage	+70°C	Work normally after recovery under normal temperature for two hours
4	Low temperature storage	-25°C	Work normally after recovery under normal temperature for two hours
5	Vibration （sine）	5~9Hz，3.5 mm amplitude； 9~200Hz ，10 m/s^2 acceleration； sweep frequency vibration for 5 times at 3 perpendicular direction （about 3×50 min）	（1）element （2）appearance （3）each target
6	Shock	1/2 sine wave, acceleration is 20g, pulse width is 11ms，X,Y,Z three directions ，three times per direction	

7. Mechanical characteristics

7. 1 Charger Dimension

Items	Ratings	Remarks
Dimension (mm)	192.4L x100W x 52H	1) Cable length is defined separately. 2) Tolerance of outline dimension is ±0.5mm，others are ±0.2mm in the diagram;
Weight (kg)	1.8 (approx.)	
Placement	Horizontal or vertical	

7. 2 Input / Output Connectors

Items	Connector Types & Photo	Pins Definitions	Cable Length
100/ 240VAC Connector	TBD	Pin 1:L(Line) Pin 2:FG(Ground Fault)	
42VDC + 12VDC Connector	JST VHR-4N (4 pins female) or compatible model 	Pin 1: +42V (red) Pin 2: 0V (black) Pin 3: +12V (green) Pin 4 (I/O): 42V enable - active low (white)	60 cm (wire gauge: AWG20)

Note: The mechanical layout of the entire charger is defined in section 7.5.

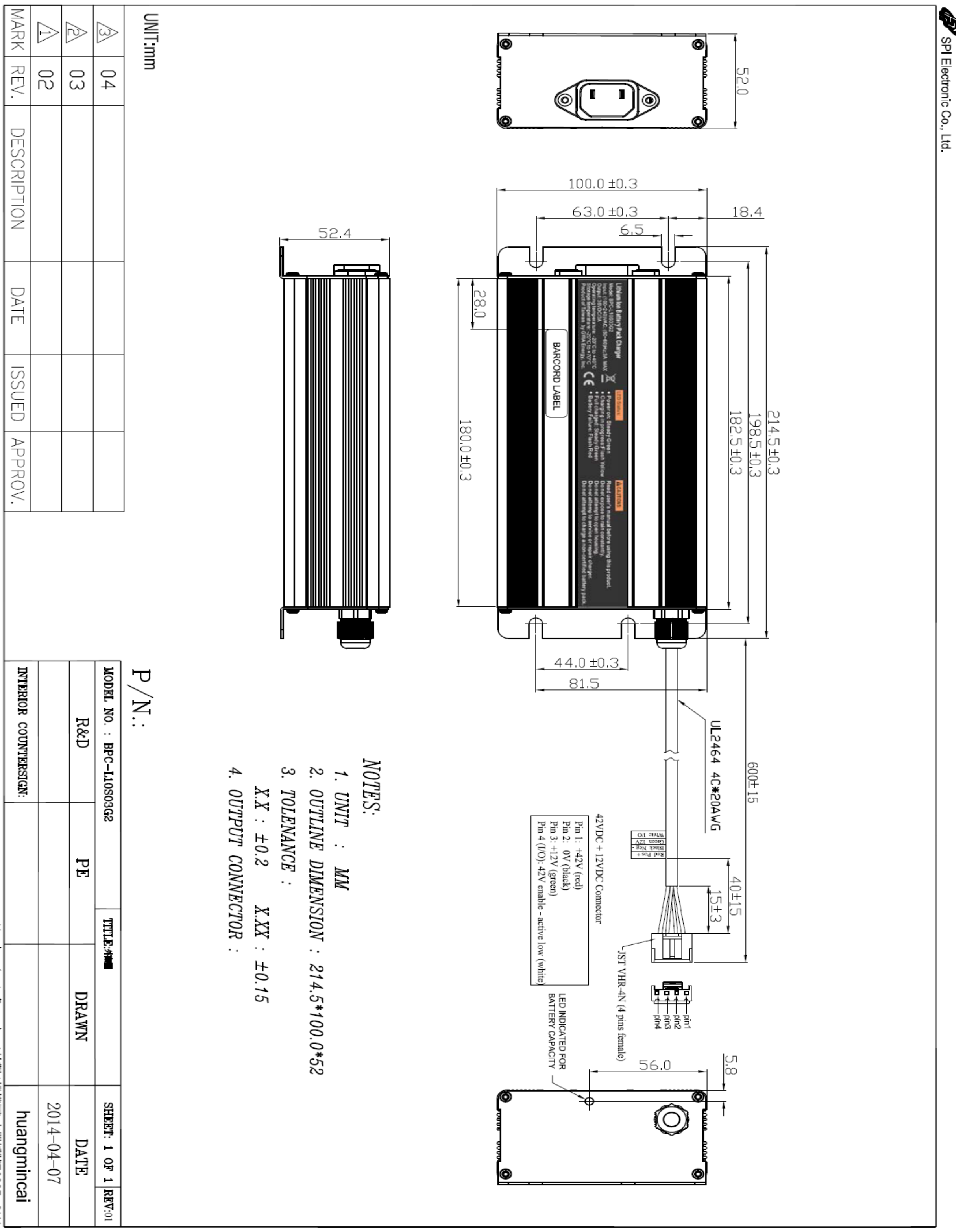
7. 3 Material Used in External Housing

Items	Material	Remarks
External housing	Aluminum	Rust-proof is required

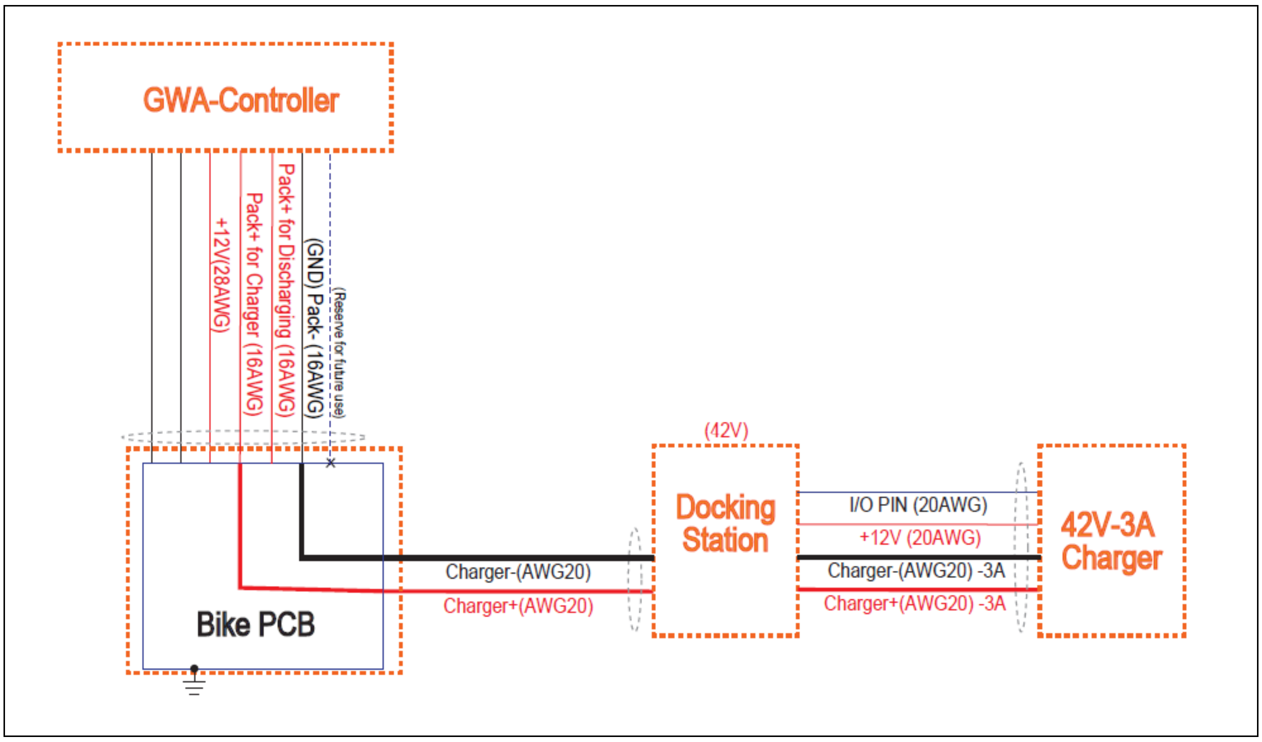
7.4 Other requirements

No	Items	Technical specification	Remarks
1	Input terminator	TBD	
2	Output terminator	JST VHR-4N (4 pins female)	
3	IP rating	IP54	Due to the use of a C14 inlet AC socket, the CE certification agency cannot grand the use of an IP54 mark on the external label. However, the agency does indicate in its report that the enclosure of the aluminum housing is compliance with IP54 rating.
4	Housing color	Black	

7.5 Mechanical Layout



8. Overall Functional Diagram



9. Package, transportation & storage

9.1 Package

There are product name, model, making of manufacturer, safety approval, and manufacturing date on the package box and manual of specifications and packing list in the package box. Final package will be defined in Appendix A.

9.2 Transportation

The products should be shielded by tent from sunshine, and loaded and unloaded carefully. It can be transported either by truck, ship, or plane via proper handling procedures.

9.3 Storage

Products should be stored in package box when it is not used. And warehouse temperature should be -25°C ~ +70°C, and relative humidity is 5%~ 95%. In the warehouse, there should not be harmful gas, inflammable, explosive products, and corrosive chemical products, and strong mechanical vibration, shock and strong magnetic field affection. The package box should be over ground at least 20cm height, and 50cm away from wall, thermal source, and vent. Under this requirement, product has 2 years of storage period, and should be rechecked when over 2 years.

10. Reliability requirements

MTBF (standard, environmental temperature, load requirement）≥30Khour ；testing condition：25°C, full load, testing proved value.

11. Product Warranty

This product is warrant for 24 months from date of factory shipment.

12. Attention

12.1 Distance of assembled bottom board and pin of element on power supply panel should be more than 8mm; distance of heat radiator and other conductor should be more 8mm, if cannot, isolation treatment such as placing PVC sheets or colloidal silica sheets is needed.
12.2 Pay attention to high voltage, avoiding touch areas marked with “high voltage” logo.

13. Decal

Decal color: black background with silver letters
Size: to be defined per final housing
Bar code: 1) must be able to track production week, 2) serial number, 3) can be scaned

Front Decal: Letters on the front end of decal are as follows:

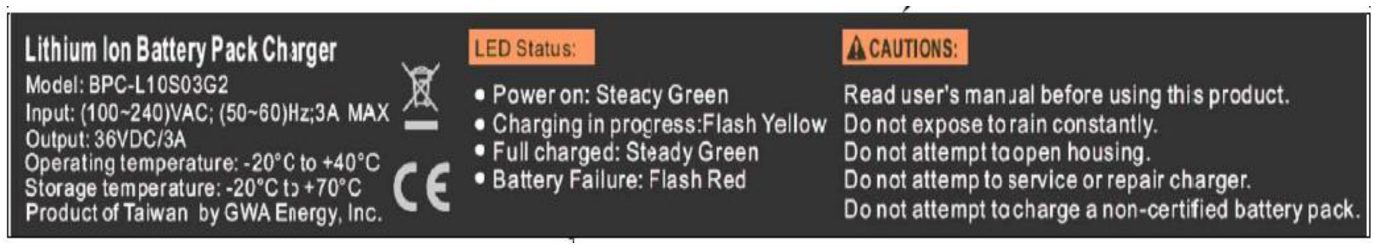
Lithium Ion Battery Pack Charger
Model: BPC-L10S03G2
Input: (100~240)VAC; (50~60)Hz; 3A MAX
Output: 36VDC/3A
Operating temperature: -20°C to +40°C
Storage temperature: -20°C to +70°C
Product of Taiwan by GWA Energy, Inc.

LED Status:
- Power on: Steady Green
- Charging in progress: Flash Yellow
- Full charged: Steady Green
- Battery Failure: Flash Red

CE mark

Rear Decal: Letters on the rear end of the decal are as follows:

CAUTIONS:
Read user's manual before using this product.
Do not expose to rain constantly.
Do not expose to open housing
Do not attemp to open, service or repair charger.
Do not attempt to charge a non-certified battery pack.



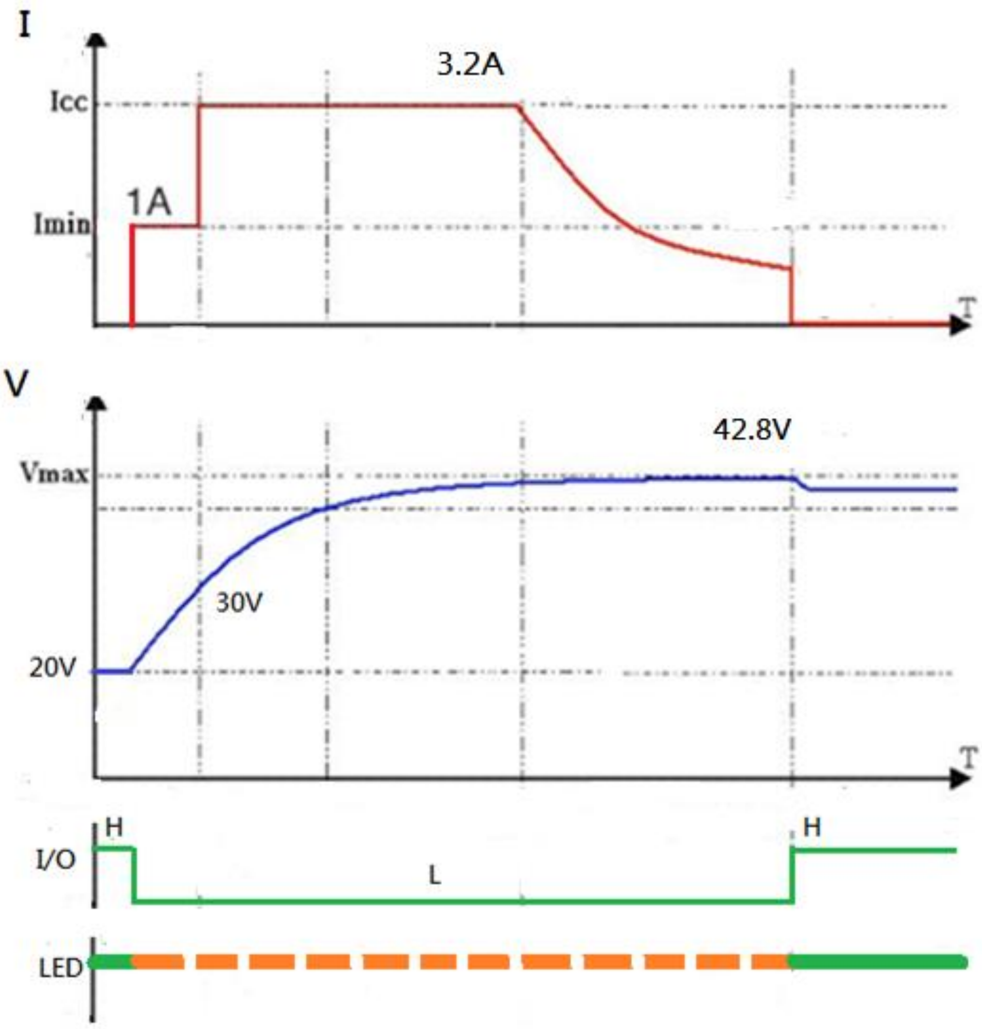
Lithium Ion Battery Pack Charger
Model: BPC-L10S03G2
Input: (100~240)VAC; (50~60)Hz;3A MAX
Output: 36VDC/3A
Operating temperature: -20°C to +40°C
Storage temperature: -20°C to +70°C
Product of Taiwan by GWA Energy, Inc.

LED Status:
- Power on: Steacy Green
- Charging in progress:Flash Yellow
- Full charged: Steady Green
- Battery Failure: Flash Red

CAUTIONS:
Read user's manual before using this product.
Do not expose to rain constantly.
Do not attempt to open housing.
Do not attemp to service or repair charger.
Do not attempt to charge a non-certified battery pack.

<u>Note: Due to the use of a C14 inlet AC socket, the CE certification agency cannot grand the use of an IP54 mark on the external label as shown above. However, the agency does indicate in its report that the enclosure of the aluminum housing is compliance with IP54 rating.</u>

14. Charging curve



Above charging profile is operated at following conditions:

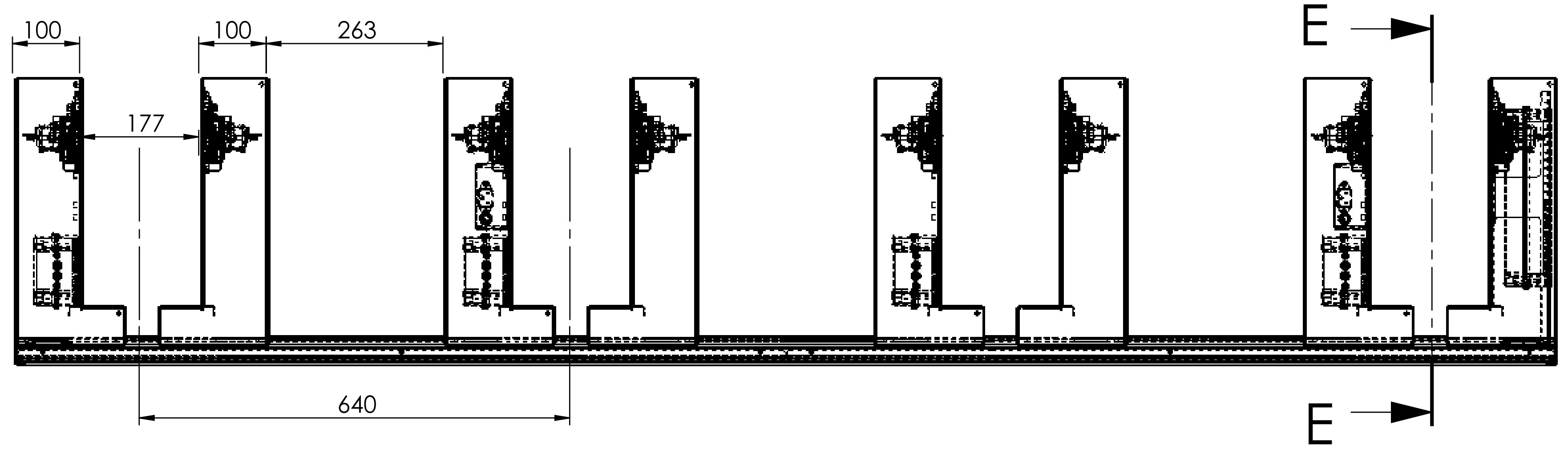
- Pre-charge current :1A±0.3A ,between 20V to 30V
- Normal charge current: 3.2A±0.3A ,between 30V to 42.8V
- Constant voltage: 42.8V±0.2V

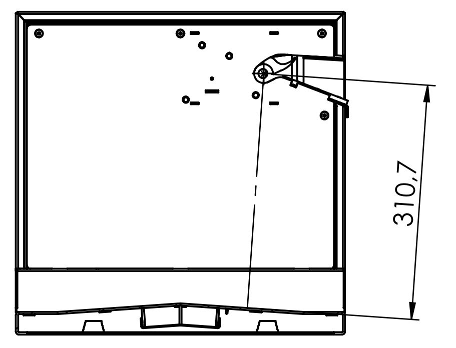
Note: If the normal charge time exceeds 4 hours at 3.2 A continuously , then time out error will occur.

Appendix A – Final package

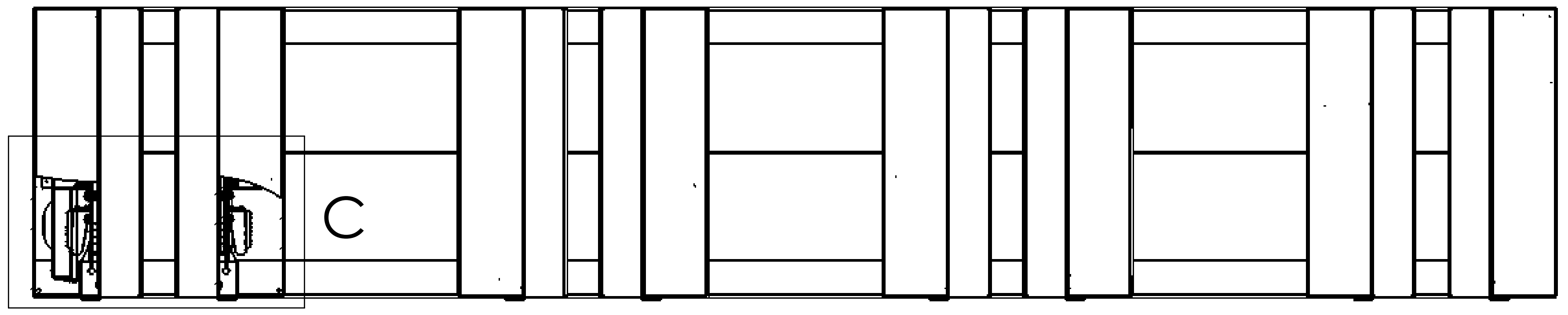
TBD

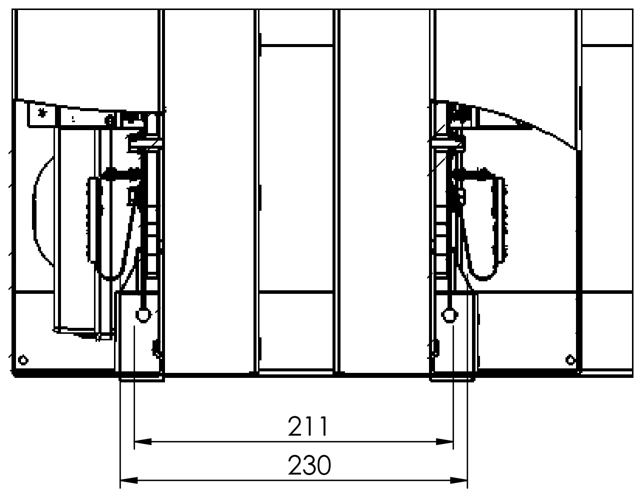
The END





SECTION E-E





DETAIL C
SCALE 1 : 5

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING — REVISION 01
	NAME	SIGNATURE	DATE			KT STÅLINDUSTRI A/S
DRAWN	Lars Hansen	LH	17.11.12			TITLE:
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. 14905-DCK4PT-01 — A3
						SCALE:1:10 — SHEET 3 OF 3

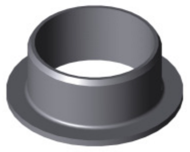
Appendix 3: Examples of Mechanical parts

Appendix 3: Examples of Mechanical parts

The drawings below have been used in SWAN1. The drawings are open domain.

Two set of tooling has been developed: First set belong to MIFA. A newer set of tooling belongs to Gobike AS.

GFM_1719_09_1_ISOMETRIC (IGUS GFM-1719-09).jpg (No drawing with measurements, this is a standard part delivered by IGUS, Germany



IGUS GFM-1719-09

In the following pages you may find the drawings for:

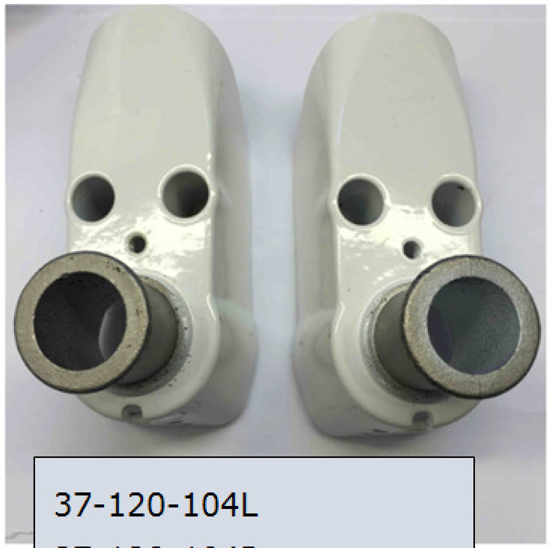
18-040-620.jpg

18-041-148.pdf

18-042-150.jpg

37-120-104L.jpg

37-120-104R.jpg



37-120-104L

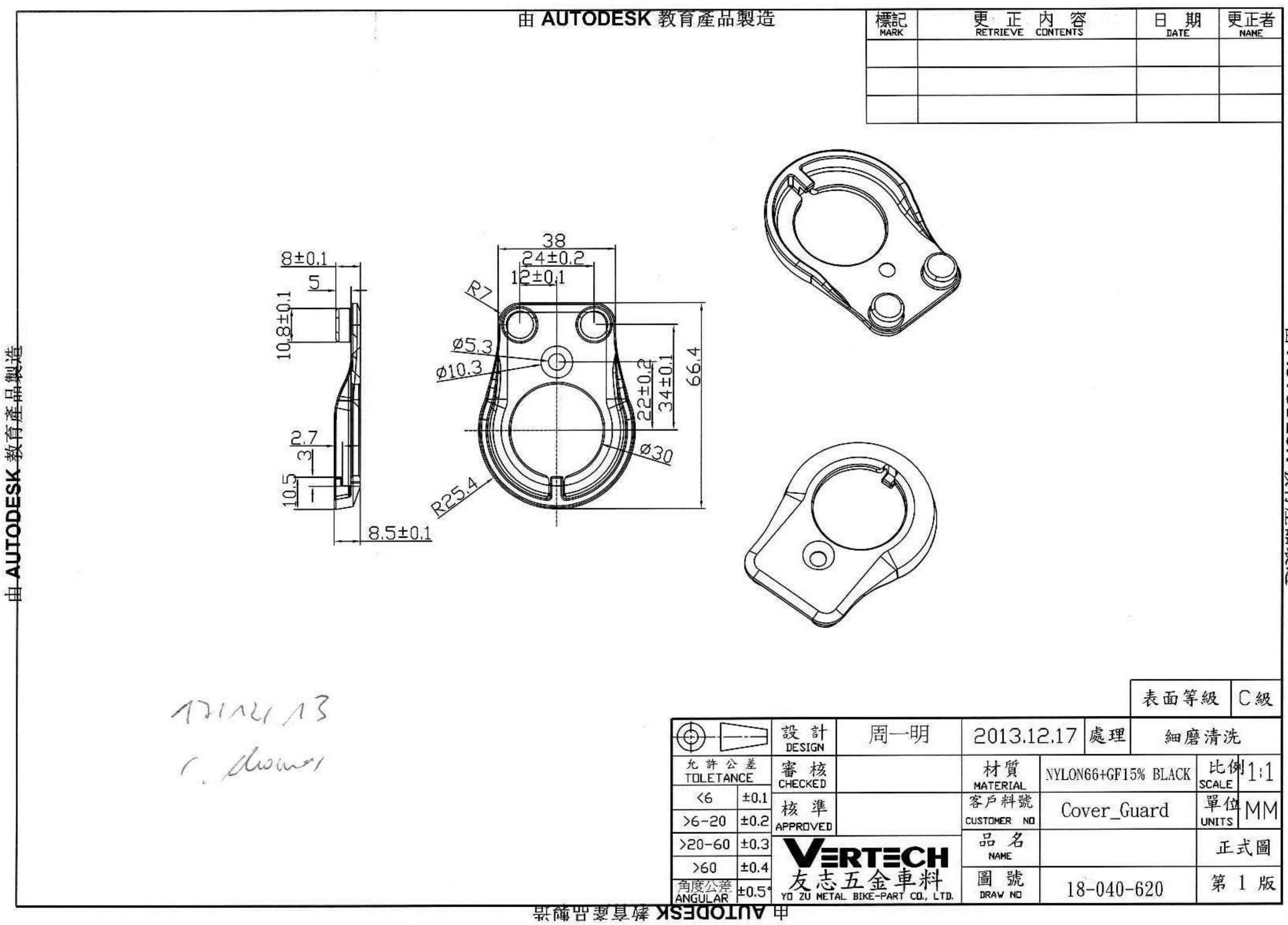
37-120-104R

18-041-148

18-042-150

18-040-620

Appendix 3: Examples of Mechanical parts

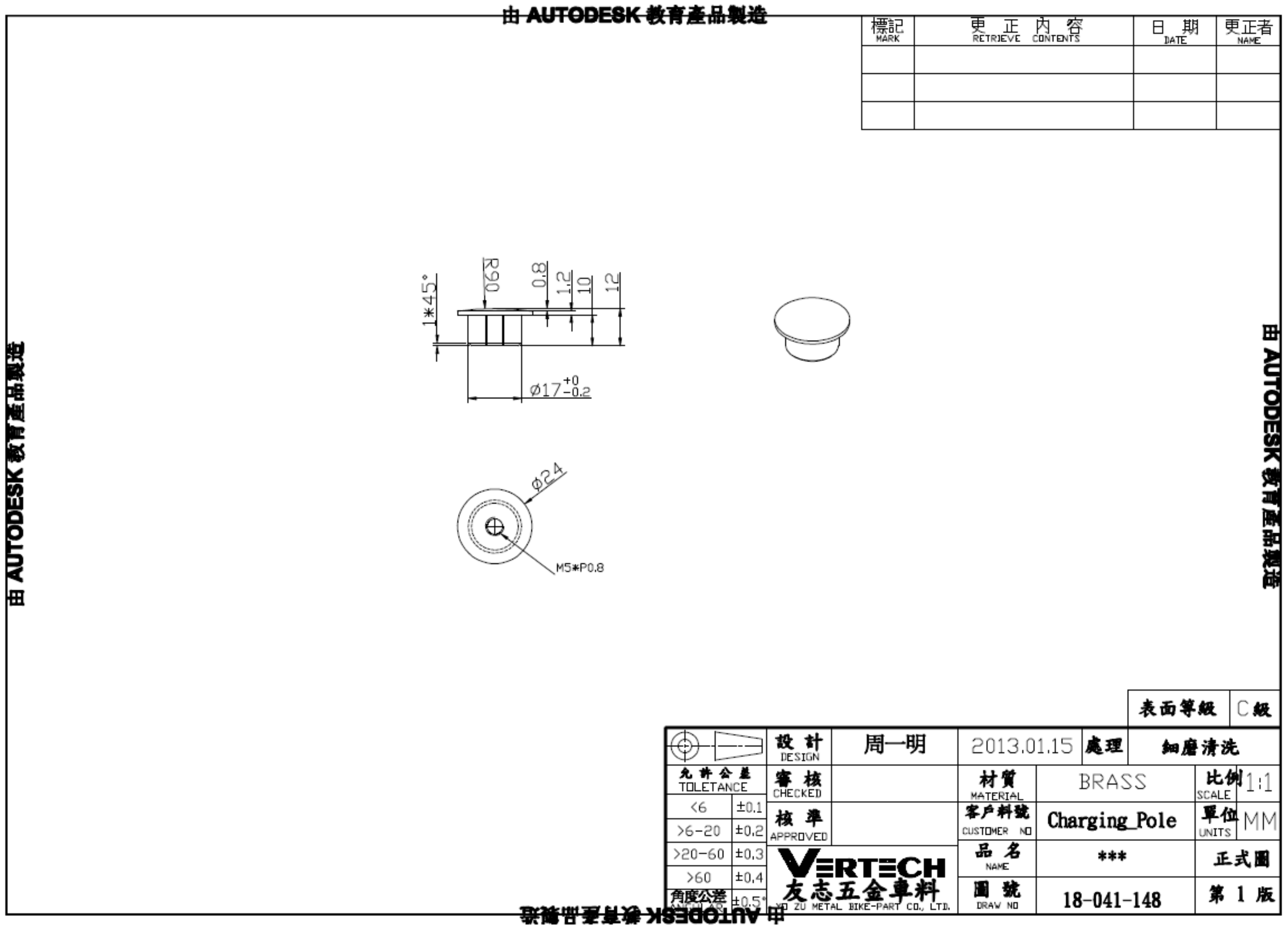


標記 MARK	更 正 內 容 RETRIEVE CONTENTS	日 期 DATE	更正者 NAME

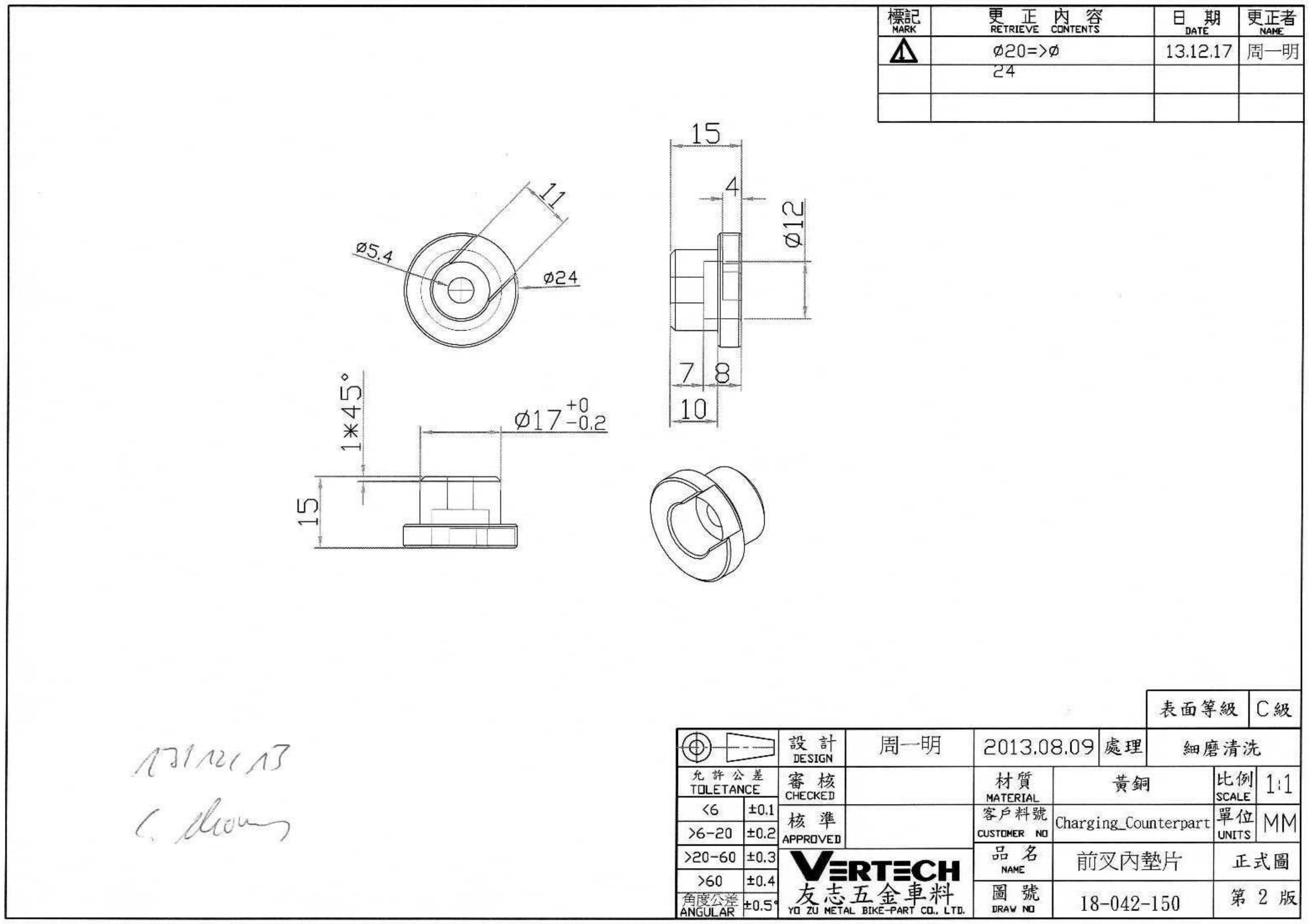
					表面等級	C級
	設 計 DESIGN	周一明	2013.12.17	處理	細磨清洗	
允 許 公 差 TOLETANCE	審 核 CHECKED		材質 MATERIAL	NYLON66+GF15% BLACK	比例 SCALE	1:1
<6 ±0.1	核 準 APPROVED		客戶料號 CUSTOMER NO	Cover_Guard	單位 UNITS	MM
>6-20 ±0.2						
>20-60 ±0.3	VERTECH 友志五金車料 YO ZU METAL BIKE-PART CO., LTD.		品 名 NAME		正式圖	
>60 ±0.4			圖 號 DRAW NO	18-040-620	第 1 版	
角度公差 ANGULAR ±0.5°						

17/12/13
C. [illegible]

Appendix 3: Examples of Mechanical parts



Appendix 3: Examples of Mechanical parts



標記 MARK	更 正 內 容 RETRIEVE CONTENTS	日 期 DATE	更正者 NAME
⚠	Ø20=>Ø 24	13.12.17	周一明

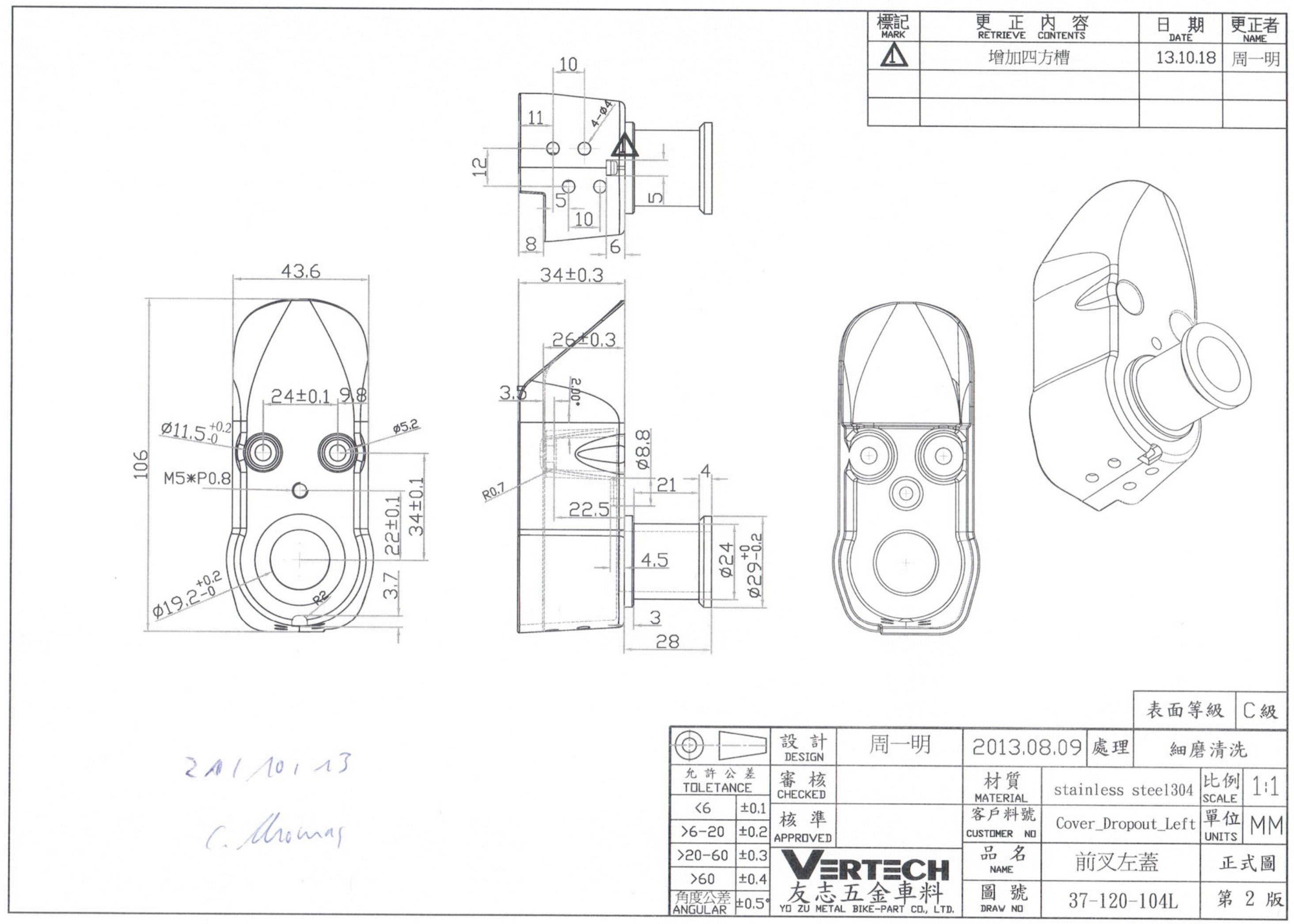
表面等級	C級

設 計 DESIGN	周一明	2013.08.09	處理	細磨清洗
審 核 CHECKED		材質 MATERIAL	黃銅	比例 SCALE 1:1
核 準 APPROVED		客戶料號 CUSTOMER NO	Charging_Counterpart	單位 UNITS MM
VERTECH 友志五金車料 YO ZU METAL BIKE-PART CO., LTD.		品 名 NAME	前叉內墊片	正式圖
		圖 號 DRAW NO	18-042-150	第 2 版

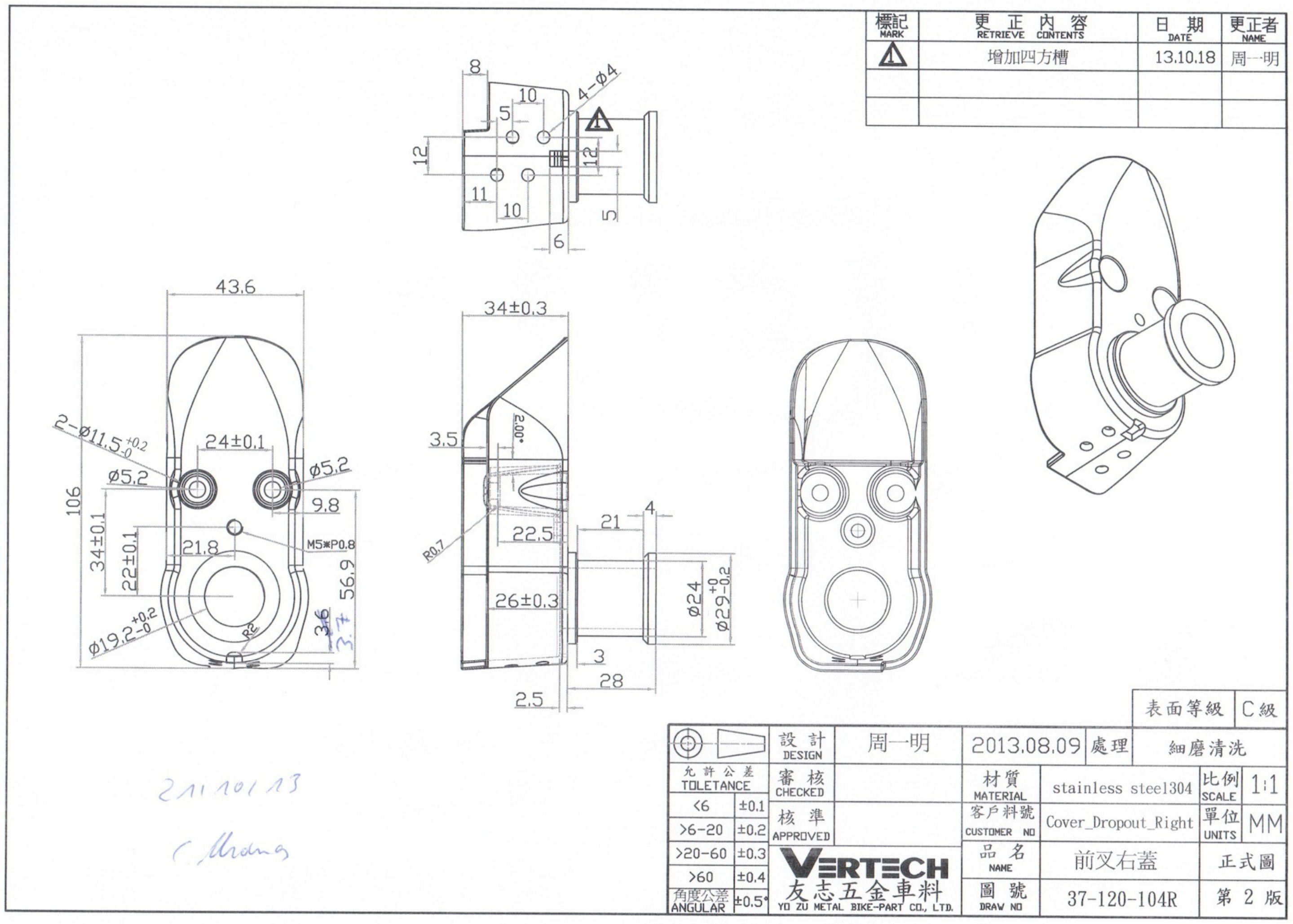
允許公差 TOLETANCE	
<6	±0.1
>6-20	±0.2
>20-60	±0.3
>60	±0.4
角度公差 ANGULAR	±0.5°

13/12/13
[illegible]

Appendix 3: Examples of Mechanical parts



Appendix 3: Examples of Mechanical parts



[illegible]

Installation and Programming Manual

TechMobility Project

V.1.30

INDEX

1 INTRODUCTION

The TechMobility Project is designed to provide a smart bike-sharing system in which some electronic devices designed by Kimaldi are involved. The present document refers to hardware-related considerations as well as to the instruction set that is to be used.

In the following sections, two electronic boards will be presented (BikePCB and Slave Controller), and the communications protocol that link them to the Host controller will be explained (KSP). For maintenance purposes a third board called Master Controller will be mentioned but its description is out of the scope of the present manual.

2 ARCHITECTURAL OVERVIEW

From the electronic point of view, the TechMobility Project double-docking system includes two cells, each of them consists of a BikePCB in the bicycle, and a Slave Controller in the docking module. The two Slave Controllers in the double-docking station share a single power source (both 12V and 42V) and are connected via CAN-bus.

The BikePCB links directly to the Tablet PC and relays instructions to and from the GWA Controller (via RS-232), as well as to the Slave Controller (via Power Line Communication, FSK).
Also, the Slave Controller is responsible for docking the bicycle and starting the recharge process.

3 THE BIKEPCB

The BikePCB is the electronics device located in the bicycle, with the following functions:

- Connect the Tablet PC to the bicycle GWA Controller, sensors (Kickstand) and actuators (Lamps, Lock and Seat), and to the docking module electronics (Slave Controller) .
- Store a TechMobility -defined Bike_Serial_Number code, so that it may be read from the Tablet PC or from the Slave Controller.
- Perform some peripheral control, either online or offline. That includes switching LED lights on and off, managing the kickstand and also the bicycle lock, as well as the electric seat.

Only two powerline wires are available for connection between the bicycle and the docking module, therefore communication must be modulated on top of the powerline, and that is achieved via FSK modulation. All the required electronics are also included on the BikePCB. The present chapter describes mechanical aspects of the BikePCB, whereas instructions are presented further ahead.

3.1 BOARD DESCRIPTION

The BikePCB electronics board has the following hardware resources:

- UART0: Virtual COM Port (USB) to Tablet PC
- UART1: RS-232 port to GWA Controller
- UART2: FSK Power Line communication to Slave Controller.
- LED driver for Bicycle Lamps (front and rear managed together)
- One Digital Output module to control Bicycle Lock
- One H-Bridge driver to Bicycle Seat
- Two Digital Inputs for Bicycle Lock and Kickstand
- Four internal Analogue-to-digital (ADC) channels to monitor voltages and currents

3.2 TECHNICAL SPECIFICATIONS

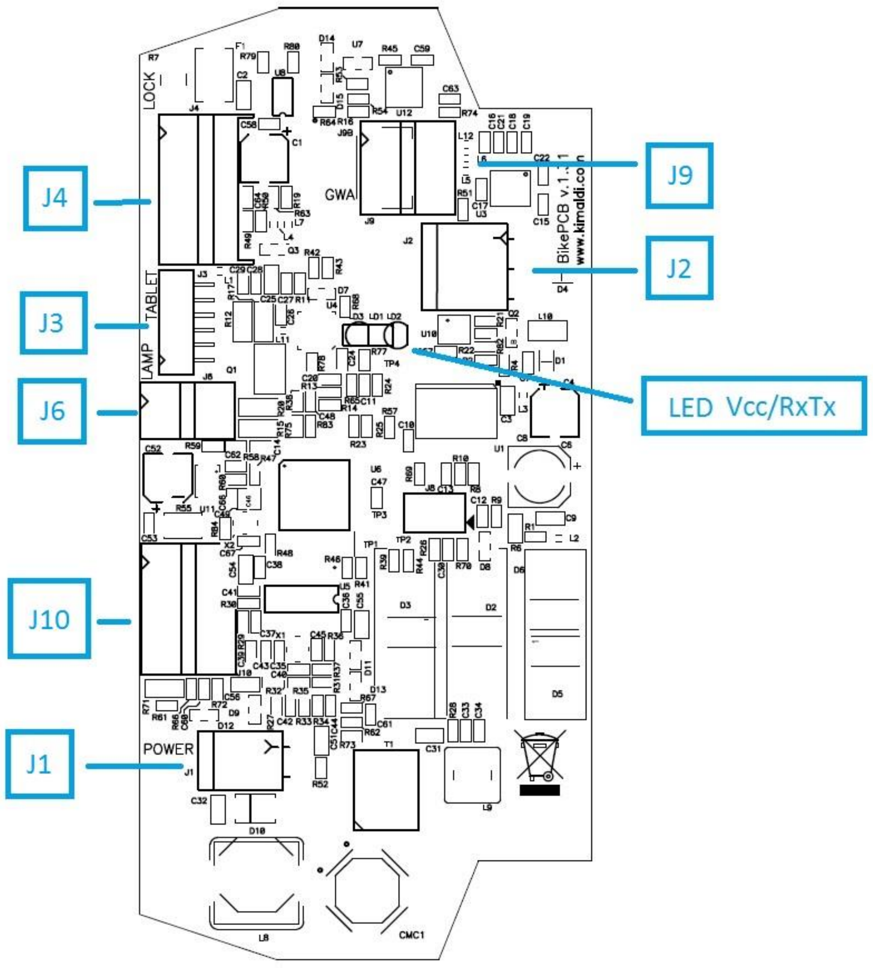
Size:	OEM board: 100 mm x 60 mm x 25 mm
Supply Voltage:	12 VDC. ± 10%
Maximum consumption:	<20 mA
UART0, USB Interface:	19200 baud; n,8,1
UART1, GWA Controller:	1200 baud; n,8,1
UART2, FSK Interface:	2400 baud; n,8,1
LED Driver:	V_out = 12V, Source impedance = 22 ohm, short-circuit current = 540 mA.
Lock Driver:	V_out = 8V, short-circuit protected. Peak current 1Amp.
Seat H-Bridge output:	V_out = 8V, Peak current = 1 Amp.
Seat Hall Power Output:	V_out = 5V, Source impedance = 22 ohm, short-circuit current = 540 mA.
Digital inputs:	2 relay-type digital inputs. In open circuit (contact open) their logic value will be 0. In contact with GND (contact closed), logic value 1.
ADC channels:	4 Analogue-to-digital channels provide monitoring to supplied voltages and current delivered to lock and lights.

3.3 ADDRESSING

When connecting from the Tablet PC, the BikePCB always shows App_ID = 0x00, Node_ID = 0x00.

For maintenance purposes, when addressed from the Master Controller, the BikePCB shows App_ID = 0x00, along with Node_ID that is equal to the one of the Slave Controller that relays to it.

3.4 CONNECTOR LAY-OUT



!!! WARNING !!!

BikePCB may handle up to +42Vdc supply

Take the necessary antistatic precautions when handling this product to avoid damaging the sensitive electronic devices.

3.5 CONNECTION DETAILS

J1 - Docking Connector: power and FSK communication coming from the Docking Module. Although marked as Left and Right, reversed polarity is admitted.

Pin 1 - DOCK_L

Pin 2 - DOCK_R

J2 - GWA Connector: power to/from GWA Controller

Pin 1 - 12V - Power delivered by the GWA Controller to the BikePCB (including peripherals) and to the Tablet PC.

Pin 2 - PWR-/GND - Negative supply to battery. It corresponds to the GND reference for the 12V supply

Pin 3 - PWR+ - Positive supply to battery (up to +42 Vcc)

J3 - Tablet PC Connector

Pin 1 - USB D+

Pin 2 - USB D-

Pin 3 - USB PWR+ (+5V)

Pin 4 - USB PWR- (GND)

Pins 5, 6 - Tablet VDD (+12V)

Pin 7 - Tablet nWAKE (Open Collector Output)

Pins 8, 9, 10 - Tablet VSS (GND)

Pins 11, 12 – Reserved

J4 - Bike Lock & Kickstand Detect Connector / Cable pin out

Pin 1 - Lock Power (Vcc)

Pin 2 - Lock - Control Signal A

Pin 3 - Lock - Control Signal B

Pin 4 - Lock Detect (Digital input)

Pin 5 - Lock Detect (GND)

Pin 6 - Kickstand Detect (Digital input)

Pin 7 - Kickstand Detect (GND)

J6 - Front and Rear LED Lamp Connectors

Pin 1 - LED+ (+12V, 22 ohm)

Pin 2 - LED- (Open Collector, 22 ohm)

J9 - GWA Connector: power from and RS-232 communication to GWA Controller

Pin 1 - RFU - Reserved for Future Use

Pin 2 - RS232-Tx - Transmit signal to GWA Controller

Pin 3 - RS232-Rx - Receive signal from GWA Controller

Pin 4 - 42V_in - Reserved for Future Use (GWA Cable not connected)

J10 – Seat Connector

Pin 1 – Seat - Control Signal A (H- Bridge A)

Pin 2 – Seat - Control Signal B (H- Bridge B)

Pin 3 – Seat Signal Power (GND)

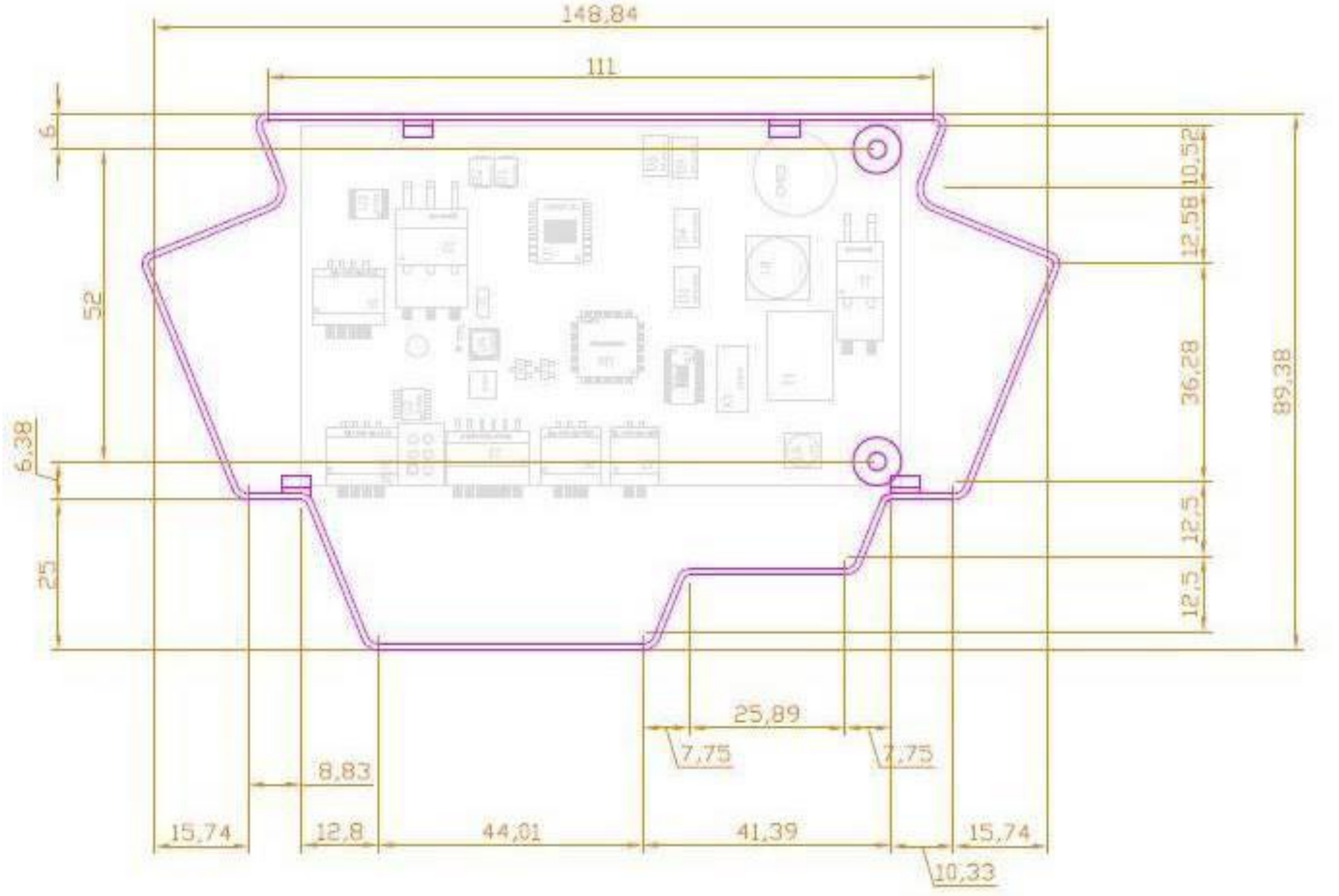
Pin 4 – Seat Signal Power (+5V)

Pin 5 – Seat Input Signal Hall B

Pin 6 – Seat Input Signal Hall A

3.6 BIKE PCB MECHANICAL OUTLINE

Size in millimeters:



4 THE SLAVE CONTROLLER

The Slave Controller is the electronics device located in the docking module, with the following functions:

- Work as a node to the Docking Station CAN-Bus.
- Manage docking and undocking operations.
- Communicate with the BikePCB, in order to retrieve Bike_Serial_Number and monitor bicycle status.
- Manage battery charging process from the GWA Charger in the Docking Module towards GWA Controller on the bicycle.
- Provide a simple, LED-based, User Interface.
- Manage temperature-related actions (i.e, heating strategy).
- Manage Charger Resource arbitration to allow sharing the resource between both Slave Controllers.
- Manage the 12V Power Supply arbitration to reduce peak current consumption.

Only two power line wires are available for connection between the bicycle and the docking module, therefore communication must be modulated on top of the power line, and that is achieved via FSK modulation. All the required electronics are also included on the Slave Controller. The present chapter describes mechanical aspects of the Slave Controller, whereas instructions are presented further ahead.

4.1 BOARD DESCRIPTION

The *Slave Controller* electronics board has the following hardware resources:

- CAN Bus Interface to interconnect the two Slave Controllers in a Double-Docking Station.
- UART1: TTL-level UART for maintenance operations (diagnostics, configuration, reprogramming).
- UART2: FSK Power Line communication to BikePCB.
- Two Solenoid drivers (left and right) for Bicycle Dock and heating.
- Four Digital inputs for *Docking Detection* (left and right).
- Five internal Analogue-to-digital (ADC) channels to monitor voltages and currents.
- Three LED outputs for User Interface (suggested: Green, Red, Blue).
- Two relays to manage power towards bicycle.
- Hall-effect sensor to monitor battery charging current.
- On-board temperature sensor.

4.2 TECHNICAL SPECIFICATIONS

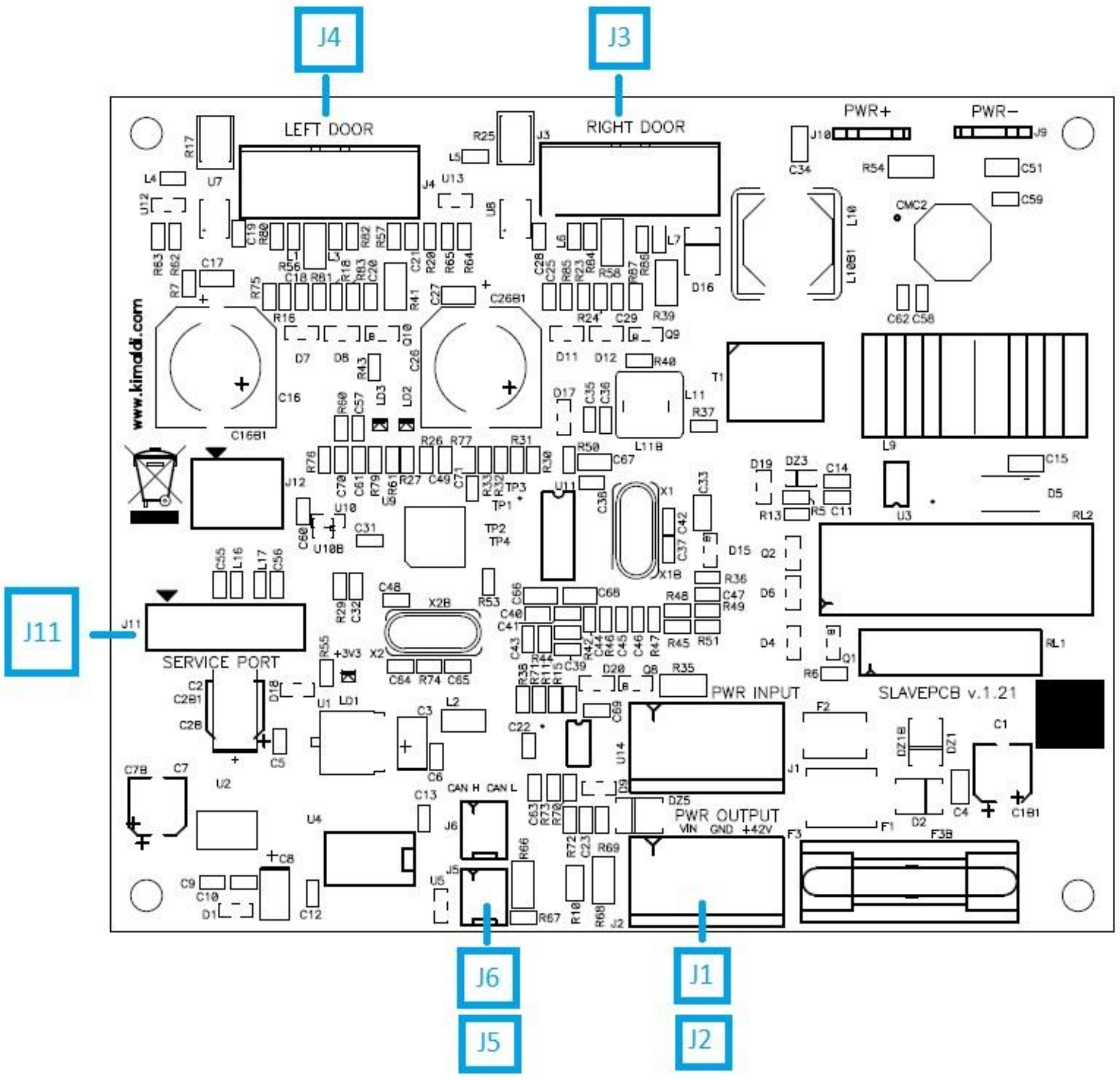
Size:	OEM board: 112.2 mm x 93 mm x 31 mm
Supply Voltage:	12 VDC. ± 10%
Maximum consumption:	50 mA
CAN Interface	
UART1, Service:	19200 baud; n,8,1
UART2, FSK Interface:	2400 baud; n,8,1
LED Drivers:	V_out = 12V, Source impedance = 100 ohm, short-circuit current = 120 mA.
H-Bridge outputs:	V_out = 12V, Peak current = 1 Amp.
Digital inputs:	4 relay-type digital inputs. In open circuit (contact open) their logic value will be 0. In contact with GND (contact closed), logic value 1.
ADC channels:	5 Analogue-to-digital channels provide monitoring to ambient temperature and current delivered to battery, and solenoids.
Main switching relay:	Switches between 12V, 0.5 Amp and 42V, 4 Amp (2 CO).
Secondary relay:	Opens/closes 12V, 0.5 Amp (1 CO).

4.3 ADDRESSING

When connecting from the Tablet PC, the Slave Controller always shows App_ID = 0x02, Node_ID = 0x00.

For maintenance purposes, when addressed from the Master Controller, the Slave Controller has App_ID = 0x02, and Node_ID as configured or negotiated from the Master Controller. Default value for Node_ID is the least-significant byte of Unique Identifier EUI-64, coded by Kimaldi. In all cases: 0x00 < Node_ID < 0xFF.

4.4 CONNECTOR LAY-OUT



!!! WARNING !!!

Slave Controller may handle up to +42Vdc supply

Take the necessary antistatic precautions when handling this product to avoid damaging the sensitive electronic devices.

4.5 CONNECTION DETAILS

J1 (Input, White) **& J2** (Output, Black) - GWA Charger Connector.

Mating connector reference is JST B4P-VH-B.

Pin 1 - +42V

Pin 2 - 0V

Pin 3 - +12V

Pin 4 - nCHARGER_ON

J3 & J4 - Docking Connectors (Right and Left, respectively).

Mating connector reference is Molex 0430251000.

Pin 1 - Dock-B (H-Bridge Output B)

Pin 2 - Dock-A (H-Bridge Output A)

Pin 3 - GND - For Dock, Solenoid and Door Detect switches

Pin 4 - Bicycle Power Connector (J3: PWR+, J4: PWR-)

Pin 5 - Bicycle Power Connector (J3: PWR+, J4: PWR-)

Pin 6 - Dock Detect (Digital Input)

Pin 7 - LED+ (Anode, 12V, 270 ohm)

Pin 8 - Solenoid Detect (Digital Input)

Pin 9 - LED- (Open Collector, 270 ohm)

Pin 10 - Bicycle Power Connector (J3: PWR+, J4: PWR-)

J6 (Input, White) **& J5** (Output, Black) – Data CAN-Bus Connectors.

Mating connector reference is Molex 22-01-2025 or 22-01-2027.

Pin 1 – CAN H

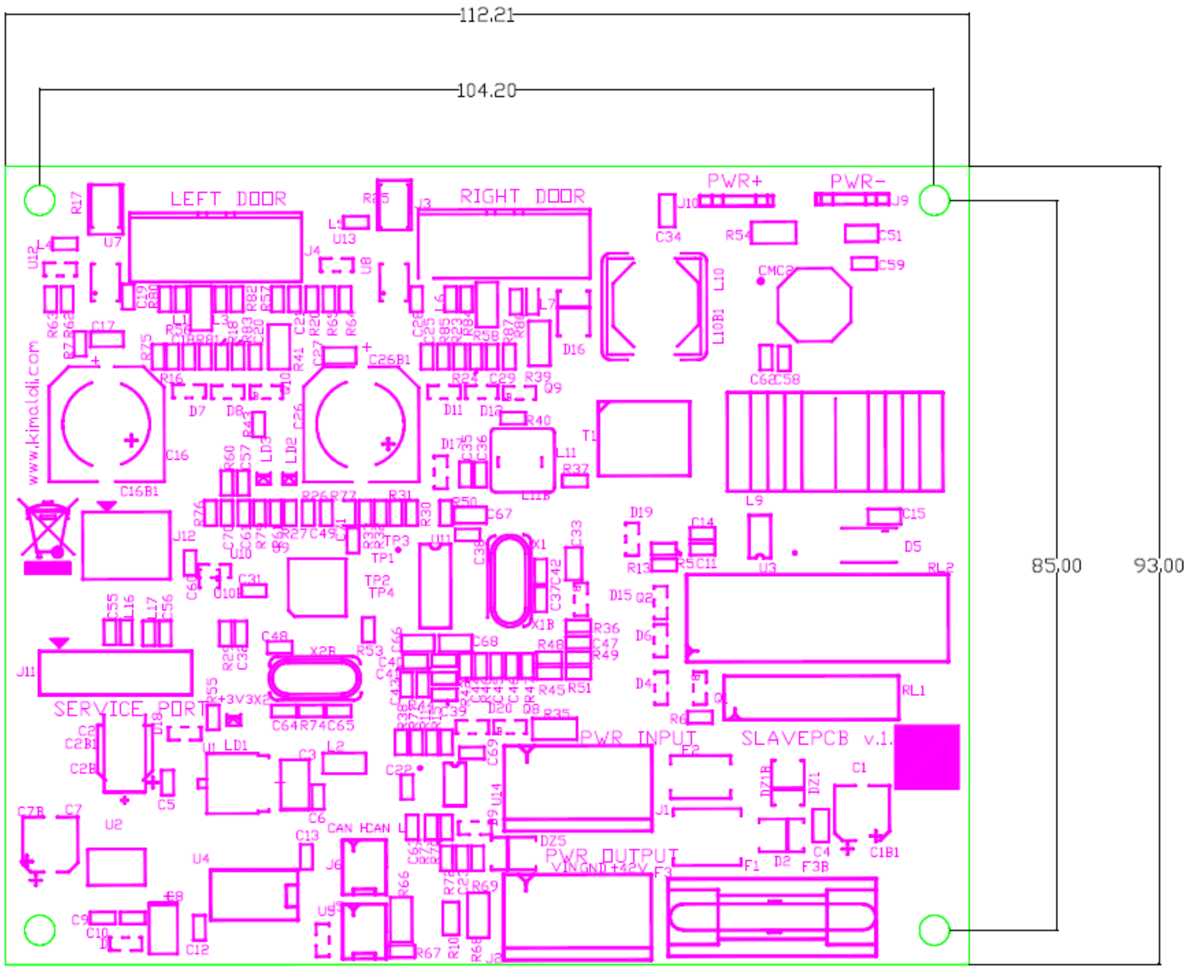
Pin 2 – CAN L

J11 – Servicing Connector.

It mates to FTDI-Chip cable reference TTL-232R-3V3. It works with the same USB drivers as the BikePCB interface to the Tablet PC.

4.6 SLAVE CONTROLLER MECHANICAL OUTLINE

Size in millimeters:



5 NETWORK TOPOLOGY

This section introduces the KSP network topology, as explained in dedicated documents. So far, let's just review the three different roles in any KSP network and how they apply in the TechMobility Project:

1. Host Computer: that is where the main application is running. In the TechMobility Project, every bicycle has a Tablet PC running an application that connects to the BikePCB.

2. Mid-Points or Gateways: any KSP device that relays instructions between the Host Computer and another device is acting as a Mid-Point. In the TechMobility Project, the BikePCB acts as a Mid-Point, thus allowing communication between the Tablet PC and the GWA Controller at all times, and between the Tablet PC and the Slave Controller whenever the bicycle is docked. From the other end, the Master Controller will act as a Mid-Point between the peer-slave and the CAN bus.

3. End-Points or End Nodes: each of the electronic devices connected to the KSP network is called an End Node. The End Node is the recipient of the Host's instructions, and therefore the one issuing answer frames towards the Host. Any BikePCB, Slave Controller or even GWA Controller can be considered End Nodes in our KSP network.

5.1 ADDRESSING SCHEME – FROM TABLET PC

From the standpoint of the application running on the bicycle's Tablet PC, the following addresses are available. Note that addresses will be expressed as [App_ID, Node_ID]:

- [0x00, 0x00]: That is the BikePCB connected to the Tablet PC.
- [0x80, 0x00]: GWA Controller on the bicycle
- [0x02, 0x00]: when the bicycle is docked, that is the Slave Controller directly attached to the BikePCB. Essentially, it is the Gateway to the CAN bus, even if we are not accessing it.

5.2 ADDRESSING SCHEME – FROM MASTER CONTROLLER

For maintenance purposes, when there is an application running on a remote server and accessing the CAN bus through the Master Controller, the following addresses are available. Note that addresses will be expressed as [App_ID, Node_ID]:

- [0x02, 0x00]: Master Controller, Gateway to the CAN bus.
- [0x02, 0x01] to [0x02, 0xFE]: Slave Controllers on the CAN bus.
- [0x02, 0xFF]: Broadcast address to Slave Controllers.
- [0x00, N]: BikePCB attached to Slave Controller [0x02, N].

6 COMUNICATIONS PROTOCOL

There are two aspects that must be taken into account, when we consider the KSP protocol:

- Frame format
- Instructions (OpCode + Arguments) and events.

In the following section, a summary of the KSP frame format is presented, before going into detail with the instruction set.

6.1 KSP-TCP OR KSP-COM FRAME FORMAT

KSP communication always occurs between the Host and a Gateway. When using a Virtual COM Port, ASCII-Hex codification is used. That means, every byte is encoded as two characters, except for <STX>, <ETX>. For example, value 0xA0 is encoded as ASCII ‘A0’, equal to Hexadecimal 0x41 0x30:

<STX><AppID><NodeID><KSP_Opc><Len><Label><OPC><ARG><CRC><ETX>

Where:

<STX>	[1 char] ASCII Value 0x02
<AppID>	[2 chars] KSP Application identifier.
<NodeID>	[2 chars] KSP Node identifier.
<KSP_Opc>	[2 chars] KSP OpCode. Its value is 0xF8 for the data frames and 0xF0 for the HandShake frames. Other values reserved.
<Len>	[2 chars] Length that follows. Its value is 2 + NA.
<Label>	[2 chars] Value that is always different, indicating the frame number. The responding ACK frame must use the same value
<OPC>	[2 chars] Device OpCode. It is the instruction to be executed
<ARG>	[2*NA chars] This field will be made up of <NA> arguments that complement the OpCode, NA from 0 to 48
<CRC>	[2 chars] Data checksum.
<ETX>	[1 byte] ASCII Value 0x03

App_ID and Node_ID refer to the destination node for transmission frames, and to the sending node for reception frames. That is, if we send an instruction, from the Host, towards node 0x03.20 (App_ID, Node_ID respectively), that is the value both for the outgoing and the incoming frames.

6.2 INSTRUCTIONS TYPES

KSP devices support, indeed, up to three different kinds of instructions:

- KSP instructions: default instruction set for BikePCB, Slave Controller and Master Controller.
- Data Relay instructions: instructions that allow communication with third-party electronic devices, namely the GWA Controller.
- Service / discovery instructions: whenever in Bootload mode, the KSP stack is not available. Therefore, a reduced instruction set is used, in order to test communications and reprogram firmware.

The following chapters explain each of the types listed above.

7 KSP INSTRUCTIONS

The default protocol for BikePCB and Slave Controller will be KSP. That provides all the handshake and retry functionality that guarantees a reliable communication.

KSP instructions always show 0xF8 as KSP OpCode, whereas the specific instruction code appears in the OPC position. Some instruction codes are common for both BikePCB and Slave Controller, whereas some other are device-specific. All that, is shown in the following sections.

7.1 GENERAL INSTRUCTIONS

7.1.1 COMMUNICATIONS TEST

Instruction	
OPC	0x00
NA	0x00
ARG	None
Response	
OPC	0x80
NA	0x00
ARG	None

Function: It allows verifying communications with the KSP device.

7.1.2 FIRMWAREVERSION / UNIQUE ID EUI-64

Instruction	
OPC	0x02
NA	0x00 or 0x01
ARG	None or 0x02, 0x04, 0x05 or 0x0C
Response	
OPC	0x82
NA	0x02, 0x04, 0x05 or 0x0C

ARG	Firmware versions and EUI-64 (Unique ID). Byte 1: Boot FW Major. Byte 2: Boot FW Minor. Byte 3: FW Major. Byte 4: FW Minor. Byte 5: FW release Bytes 5 to 12: EUI-64 (Unique ID) of the device

Function: Returns the device's firmware versions if ARG[0] has a value from 0x02 to 0x05 (Boot and regular FWs). Optionally (ARG[0] = 0x0C) the Unique ID (EUI-64) of the device is also returned

7.1.3 READCFG_BYTE

Instruction	
OPC	0x3A
NA	0x01
ARG	Byte 1: Number of the configuration parameter to read.
Response	
OPC	0xBA
NA	0x02
ARG	Byte 1: Number of the configuration parameter read. Byte 2: Value of the configuration parameter.

Function: Allows reading the value of a configuration parameter stored in the device's non-volatile memory. See Section 8.4 for BikePCB configuration parameters and Section 9.5 for Slave Controller.

7.1.4 WRITECFG_BYTE

Instruction	
OPC	0x3B
NA	0x02
ARG	Byte 1: Number of configuration parameter to write. Byte 2: Value of the configuration parameter.
Response	
OPC	0xBB

NA	0x01
ARG	Byte 1: Number of the configuration parameter written.

Function: Allows writing the value of a configuration parameter stored stored in the device's non-volatile memory. See Section 8.4 for BikePCB configuration parameters and Section 9.5 for Slave Controller.

7.1.5 FACTORYCFG_BYTE

Instruction	
OPC	0x08
NA	0x00
ARG	None.
Response	
OPC	0x88
NA	0x00
ARG	None.

Function: It allows restoring the factory configuration of the device.

7.1.6 APPLYCFG

Instruction	
OPC	0x39
NA	0x00
ARG	None
Response	*(No time to send a response)*
OPC	*0xB9*
NA	*0x00*
ARG	*None*

Function: Restarts the device so that the new configuration values can be applied.

7.1.7 READ NODE TABLE

Instruction	
OPC	0x4A
NA	0x00
ARG	None
Response	
OPC	0xCA
NA	0x06 (Node is BikePCB) or 0x08 (Node is Slave Controller)
ARG	Byte 1: NodeID Byte 2: Sync Timer value Bytes 3, 4: EUI-64 least-significant word Byte 5: Flash2 Major Byte 6: Node Status (BikePCB or Slave Controller State) Byte 7: Heater Status and flags (for Slave Controller only) Byte 8: Digital Input Status (for Slave Controller only)

Function: It reports the status of the board associated to the one we are communicating with. For instance, if BikePCB is queried, Read_NodeTable will report the status of Slave Controller (its NodeID is 0x00). If Slave Controller is queried, the status reported will correspond to BikePCB.

The Sync Timer value can help us determine whether synchronization is ongoing correctly. Slave Controller must report a Sync Timer value of 0x07 to allow communications from BikePCB to Slave, whereas a value of 0x06 is sufficient for communications from Slave Controller to BikePCB. A lower value means that some sync frames are lost. If the value is 0x0, then we have an association error.

The status of the slave controller that is reported corresponds to the value stored in the Node Table, as a result of the automatic synchronization protocol between Slave Controller and BikePCB. That status is updated every 2 seconds in the absence of any other traffic between Slave Controller and BikePCB.

Please note that Slave Controller Status can be queried from BikePCB, either with the Read_NodeTable discussed here or with Get Slave Controller Status detailed in Subsection 7.2.8.

7.1.8 ERROR CODES

Both BikePCB and Slave Controller may answer to any instruction with any one of the following OpCodes, instead of the corresponding Response OpCode:

- OpCode 0xFD: Frame Delay (wait, then re-send).
- OpCode 0xFE: Frame Error (wrong frame format, or illegal value on arguments).
- OpCode 0xFF: Instruction Error (OpCode not valid).

7.2 BIKEPCB INSTRUCTIONS

7.2.1 GET BIKE_SERIAL_NUMBER

Instruction	
OPC	0x10
NA	0x00
ARG	None
Response	
OPC	0x90
NA	0x01 to 0x30
ARG	Bike Serial Number

Function: Retrieves the Bike_Serial_Number string (ASCII values). End of string is determined by the presence of Char (0x00) or Char (0xFF)in the last written EEPROM position, which is not returned.

That should be a write-once attribute for BikePCB.

7.2.2 SET BIKE_SERIAL_NUMBER

Instruction	
OPC	0x11
NA	0x01 to 0x30
ARG	Bike Serial Number + 0x00
Response	
OPC	0x90
NA	0x01
ARG	Byte 1: Error condition - 0x00 if operation success

Function: Stores the Bike_Serial_Number string (ASCII values) into BikePCB EEPROM. If fewer than 48 characters are used, it is recommended to add a Char (0x00) at the end of Bike_Serial_Number (therefore incrementing NA by 1).

7.2.3 ACTIVATE LED

Instruction	
OPC	0x3C
NA	0x02
ARG	Byte 1: Selection of output to activate 0x00: green LED 0x01: red LED 0x02:Tablet nWAKE signal Byte 2: Activation time 0x00 to 0xFE in units of 100ms.
Response	
OPC	0xBC
NA	0x00
ARG	None

Function: Activates one of the LEDs or nWAKE signal during the specified time interval. *Only for testing purposes.*

7.2.4 DIGITAL INPUT STATUS

Instruction	
OPC	0x35
NA	0x00
ARG	None
Response	
OPC	0xB5
NA	0x01
ARG	Byte 1: Digital Input bitmap Bit 0: Kickstand is deployed when bit is set (i.e, bike standing). Bit 1: Lock is secured when bit is set (i.e, bike is locked).

Function: Queries about the status of the digital inputs managed by the device.

Only for testing purposes. Normally, a change in a relevant Digital Input involves a status change. Therefore, use Get Bike Status in user application.

7.2.5 ADC READ

Instruction	
OPC	0x38
NA	0x01
ARG	Byte 1: Channel Number
Response	
OPC	0xB8
NA	0x03
ARG	Byte 1: Channel Number Byte 2: ADC Reading (see below) Byte 3: Error Condition

Function: we can read values from the different Analog-to-Digital Converters available on BikePCB. Readings are not direct, but they are continuously polled by the BikePCB and recorded only when their value is not zero (specifically, for the lock and the lamps). Channels are arranged as follows:

- Channel 0: Light, current sense (250 counts equal to 500 mA).
- Channel 1: Same as channel 0.
- Channel 2: Tablet supply, voltage sense (250 counts equal to 50 V).
- Channel 3: Charger supply, voltage sense (250 counts equal to 50 V).
- Channel 4: Bicycle lock, current sense (250 counts equal to 1000 mA).
- Channel 5: Same as channel 4.

For convenience, additional channels are available with statistical values:

- Channel 6: Charger supply, maximum voltage sensed (250 counts equal to 50 V).
- Channel 7: Bicycle lock, average current sensed (250 counts equal to 250 mA).
- Channel 8: Tablet supply, minimum voltage sensed (250 counts equal to 50 V).

Maximum and minimum values are automatically reset after being read by the Host.

7.2.6 GET BIKEPCB STATUS

Instruction	
OPC	0x56
NA	0x00
ARG	None
Response	

OPC	0xD6
NA	0x01
ARG	Byte 1: Current BikePCB Status

Function: the current status of the bicycle is reported. Bike Status is encoded as detailed on the state diagram on Figure 2, Section 8.1 and on Table 1.

The status of the bicycle may change without the Host's intervention, either because the user has pushed the bicycle into the docking module, or because the kickstand or the lock have been activated. In those cases, an event is issued by the BikePCB, so that the Host can respond to the new situation.

7.2.7 SET BIKEPCB STATUS

Instruction	
OPC	0x57
NA	0x01
ARG	Byte 1: New BikePCB Status
Response	
OPC	0xD7
NA	0x01 or 0x02
ARG	Byte 1: Current BikePCB Status Byte 2: Additional information, depending on New Bike Status.

Function: the Host requests a status change to the bicycle. Status change may or may not be successful. The response will show us the status after the instruction. In particular:

- When the bicycle is docked (BIKE_DOCKING or BIKE_CHARGING), the "undock" action must be commanded to the Slave Controller, then to the BikePCB.
- In some occasions, it is the situation of the kickstand or the lock which will determine the possibility to transition to the new status.
- The returned Current Bike Status may or may not match New Bike Status. When it does not match, a second argument may be issued, in order to provide additional information on the reasons why the requested status was not achieved.

Bike Status is encoded as shown in Subsection 7.2.6., Get Bike Status.

7.2.8 GET SLAVE CONTROLLER STATUS

Instruction	
OPC	0x59
NA	0x00
ARG	None
Response	
OPC	0xD9
NA	0x04
ARG	Byte 1: Current Slave Controller Status Byte 2: Heater Status (and other flags) Byte 3: Digital Input Status Byte 4: Sync Timer value

Function: the current status of Slave Controller is reported. Slave Controller Status is encoded as explained in Section 9.1.: “Offline behavior”.

Bitmap for Digital Input Status is the same as in Digital Input Status instruction.

Same as with Read_NodeTable, the values reported in the current instruction correspond to the ones stored in the Node Table, which are updated every time that BikePCB and Slave Controller synchronize automatically. Therefore, the quality metric for these values is provided by the value of Sync Timer: normally, Sync Timer is 7, which means that the data is refreshed on time. If Sync Timer decays between 6 to 1, 1 to 6 refresh frames have been missed, so the data are not real-time any more. If Sync Timer is 0, we’re probably in Association Error, so the data are not realistic. If Sync Timer is 0xFF, then Slave is not associated to the bike at all and the data are not valid at all.

7.2.9 OUTPUT TO GWA CONTROLLER

Instruction	
OPC	0x5B
NA	0x03
ARG	[GWA CommandCode][LowByte][HighByte]
Response	
OPC	0xDB

NA	0x03
ARG	[GWA ResponseCode][LowByte][HighByte]

Function: communication to GWA Controller may be established, via KSP. The rest of the GWA Controller's protocol is directly implemented by BikePCB (i.e, no need to add CRC). This method is alternative to the one in Section 7.4.: "Data Relay instructions".

7.2.10 SEAT OPERATE

Instruction	
OPC	0x3E
NA	0x02 or 0x03
ARG	Specifies the seat module instruction code and its argument: Byte 1: Seat module instruction code 0x00-SeatStop 0x01-SeatMoveRelativeUp 0x02-SeatMoveRelativeDown 0x03-SeatGoAbsolute 0x04-SeatFastCalibrationEnable/Disable 0x05-SeatCancelAbsoluteMode Byte 2: Seat module instruction Parameter_1. Its meaning depends on each instruction as explained in Seat behavior section. If NA=0x03: Byte 3: Timeout of the Seat Operating State that will apply after instruction execution, in steps of 80ms. Maximum value 0x7D (equals 10 sec.) . Any higher value will be interpreted as the maximum. If Byte 3 is not specified, the default timeout will be 10 sec.
Response/Event	
OPC	0xBE
NA	0x03
ARG	After issuing a SeatOperate the BikePCB will issue an AnsSeatOperate containing three bytes of information. Also every second during execution and just after execution has been completed AnsSeatOperate frames will be send. The following is the format:

	Byte 1: Bit 7,6: 00: the instruction has been correctly received. 01: the instruction is beeing executed 10: the instruction has been completed Bit 5: 0: the seat is operating in relative mode. 1: the seat is operating in absolute mode Bit 4: 0: fast calibration is disabled 1: fast calibration is enabled Bit3..0: indicates the instruction code Byte 2 and Byte 3: Contains the Result_1 and Result_2 value respectively. The meaning depends on the instruction as explained in Seat behavior section.. When representing position values, it is important to take into account that the precision of all reported measures is +/- 1 count.

7.2.11 SEAT POLL

Instruction	
OPC	0x3F
NA	0x00
ARG	None.
Response/Event	
OPC	0xBF
NA	0x02
ARG	Byte 1: Operation Flags Bit 7: Indicates if seat shaft has reached the bottom position. Bit 6: Always read zero. Bit 5: The seat is calibrated. (i.e. absolute mode) Bit 4: Fast calibration is enabled Bit3..0: Validity nibble. If all zeroes: the seat is not operating and the previous bits are reliable. If all ones: the seat is operating, all previous bits will read zero.

	Byte 2: If the seat is calibrated this value will contain the absolute position of the seat. If not will read zero.

7.2.12 LOCK SUPPLY

Instruction	
OPC	0x3D
NA	0x01
ARG	0x00: Switches off the lock power supply. 0x01: Switches on the lock power supply 0x02: Requests the lock power supply status
Response/Event	
OPC	0xBD
NA	0x02
ARG	Byte 1: echoes the instruction parameter if within the admitted range, or zero otherwise. Byte 2: reports the lock power supply status after the instruction execution. 0x00 stands for "power off" 0x01: stands for "power off"

Function: Allows the programmer to switch on and off the lock power supply, and to know the power current status. Switching on and requesting the status can be done at any time. But switching off will have no effect if the bike is actually operating the lock or the seat.

7.3 SLAVE CONTROLLER INSTRUCTIONS

7.3.1 ACTIVE LED

Instruction	
OPC	0x3C
NA	0x02
ARG	Byte 1: Selection of output to activate 0x00: PWR 12V output relay 0x01: PWR 42V output relay 0x02: Left dock LED 0x03: Right dock LED 0x04:Charger I/O (nEnable) Byte 2: Activation time 0x00 to 0xFE in units of 100ms.
Response	
OPC	0xBC
NA	0x00
ARG	None

Function: Activates one of the LEDs during the specified time interval. *Only, for testing purpose.*

7.3.2 DIGITAL INPUT STATUS

Instruction	
OPC	0x35
NA	0x00
ARG	None
Response	
OPC	0xB5
NA	0x01
ARG	Byte 1: Digital Input bitmap Bit 0: Dock left sensor. If set, bike is being detected. Bit 1: Dock right sensor. If set, bike is being detected. Bit 2: Solenoid left sensor. If set, the left solenoid is engaged. Bit 3: Solenoid right sensor. If set, the right solenoid is engaged.

	Bits 4, 5: Door sensors. Always read closed (correct). Bit 6: if set, 42V are on (i.e, charging detected) Bit 7: if set, heating is on.

Function: Queries about the status of the digital inputs managed by the device.

Normally, a change in a relevant Digital Input involves a status change. However, digital inputs may need to be monitored prior to the Undock process, so that it the typical use case for Digital Input Status. In other cases, using Get Slave Controller Status is the preferred method.

7.3.3 ADC READ

Instruction	
OPC	0x38
NA	0x01
ARG	Byte 1: Channel Number
Response	
OPC	0xB8
NA	0x03
ARG	Byte 1: Channel Number Byte 2: ADC Reading (see below) Byte 3: Error Condition

Function: we can read values from the different Analog-to-Digital Converters available on the Slave Controller. Channels are arranged as follows:

- Channel 0: Temperature sensor (0 counts equal to -50ºC, 50 counts equal to 0ºC, 100 counts equal to 50ºC).
- Channel 1: GWA Charger current (250 counts equal to 5 Amp).
- Channel 2: Left Docking Solenoid current (250 counts equal to 1250 mA).
- Channel 3: Right Docking Solenoid current (250 counts equal to 1250 mA).
- Channel 4: GWA Charger Voltage (250 counts equal to 50 V).

For convenience, additional channels are available with limit values:

- Channel 5: Minimum temperature sensed (50 counts equal to 0ºC).
- Channel 6: Maximum temperature sensed (50 counts equal to 0ºC).
- Channel 7: GWA Charger, maximum current sensed (250 counts equal to 5 Amp).
- Channel 8: Left Docking Solenoid, maximum current sensed (1250 counts equal to 250 mA).
- Channel 9: Right Docking Solenoid, maximum current sensed (250 counts equal to 1250 mA).

Maximum values are automatically reset after being read by the Host.

7.3.4 GET SLAVE CONTROLLER STATUS

Instruction	
OPC	0x59
NA	0x00
ARG	None
Response	
OPC	0xD9
NA	0x01 o 0x04
ARG	Byte 1: Current Slave Controller Status *If bit 7 on Parameter 0x09 is set (see Subsection 9.5.3.), additional arguments are displayed:* Byte 2: Previous Slave Controller Status Byte 3: Heater Status (and other flags) Byte 4: Digital Input Status

Function: the current status of the Slave Controller is reported. Slave Status is encoded as explained in Section 9.1.: “Offline behaviour”.

The status of the Slave Controller may change without the Remote Host’s intervention, either because the user has pushed the bicycle into the docking module, or because one door has been open. When connected to the Master Controller, an event is issued by the Slave Controller, so that the Remote Host can respond to the new situation. Bitmap for Digital Input Status is the same as in Digital Input Status instruction.

7.3.5 SET SLAVE CONTROLLER STATUS

Instruction	
OPC	0x5A
NA	0x01
ARG	Byte 1: New Slave Controller Status

Response	
OPC	0xDA
NA	0x01 or 0x02
ARG	Byte 1: Current Slave Controller Status Byte 2: Additional information, depending on New Slave Controller Status.

Function: the Host requests a status change to the Slave Controller. Status change may or may not be successful. The response will show us the status after the instruction. If status has not changed, it may report Digital Input Status as an explanation for that (in particular, it means that the sensors do not allow undocking, or that heating has been turned on or off).

Some status changes require a command from the Host, either through the Master Controller or from BikePCB (i.e, from the Tablet PC).

Slave Status is encoded as explained in Section 9.5.: “Slave Controller States”.

Bitmap for Digital Input Status is the same as in Digital Input Status instruction.

7.3.6 SET DATE AND TIME

Instruction	
OPC	0x50
NA	0x04
ARG	See Section 9.4.1.: “Date and Time format”
Response	
OPC	0xD0
NA	0x01
ARG	Byte 1: Error condition - 0x00 if operation success.

Function: Updates date and time values on Slave Controller.

Hint: This instruction can be sent to one single Slave Controller (from the Tablet PC), or broadcast to all the Slave Controllers attached to the bus (from the Master Controller), so that logging information stored in Slave Controller makes sense.

7.3.7 GET DATE AND TIME

Instruction	
OPC	0x51
NA	0x00
ARG	None
Response	
OPC	0xD1
NA	0x04
ARG	See Section 9.4.1.: “Date and Time format”

Function: Date and Time values may be retrieved from Slave Controller.

7.3.8 GET STATISTICS

Instruction	
OPC	0x52
NA	0x00
ARG	None
Response	
OPC	0xD2
NA	0x0A
ARG	Bytes 1,2: number of successful dockings Bytes 3,4: number of unsuccessful dockings Bytes 5,6: number of successful undockings Bytes 7,8: number of unsuccessful undockings Bytes 9,10: number of received broadcast errors See Section 9.4.3.: “Statistical information”

Function: Success and Fail statistics may be retrieved from Slave Controller. It is a quick way to spot failure conditions.

7.3.9 START LOG QUERY

Instruction	
OPC	0x53
NA	0x00
ARG	None
Response	
OPC	0xD3
NA	0x01
ARG	Bytes 1: number of logs recorded.

Function: Sets the log read pointer to the beginning of the log stack (most recent log), and announces how many log records are available for retrieval. A maximum of 127 log records are available at any time (see Section 9.4).

7.3.10 RETRIEVE NEXT LOG RECORD

Instruction	
OPC	0x54
NA	0x00 or 0x01
ARG	Byte 1 (optional): Event Code to be retrieved
Response	
OPC	0xD4
NA	0x00 or 0x09
ARG	Byte 1: read pointer position Bytes 2 to 9: log record. See its format in Section 9.4

Function: Retrieves a log record from Slave Controller.

If no arguments are provided, the next log record to be read is returned, and the read pointer is automatically moved to the next position.

If an event code is provided as an argument, the next log record that matches the event code is returned, and the read pointer is automatically moved to the next position.

If no more records are available, 0 arguments are returned

7.3.11 DELETE RECORDS

Instruction	
OPC	0x55
NA	0x01
ARG	Byte 1: Bit 0: if set, statistics are cleared Bit 1: if set, log records are all cleared
Response	
OPC	0xD5
NA	0x01
ARG	Byte 1: Error condition - 0x00 if operation success.

Function: Allows resetting of the statistical counters and/or the event log database.

N.B: log deletion takes a significant amount of time. Please take into account that answer will be delayed in the order of 1 second.

7.3.12 SERVICE DOCK-UNDOCK

Instruction	
OPC	0x51
NA	0x02
ARG	Byte 1: 0x00 DOCK 0x01 UNDOCK Byte 2: 0x00 Execute only if NOT exist Firmware load in Flash 2. 0x01 Execute ALWAYS.

Response	
OPC	0xD1
NA	0x01
ARG	0xF0: Ok 0xF1: Fail

Function: It forces the DOCK or UNDOCK of bicycle. It may result convenient to send this instruction using the DEMO software. To do so click the "Additional functions" button, select "Customization" fill the instruction in the "Cmd." Box and press "Send" button. Take into account that you have to type OPC,NA,ARG1,ARG2 using two digits for each hexadecimal value. For example, to undock the bike in any case type 51020101 and press send.

7.4 DATA RELAY INSTRUCTIONS

Whenever data is sent to the GWA Controller, the following frame format will be used:

Instruction	
AppID, NodeID	[0x80, 0x00]
KSP OpCode	**0x71**: that is the "Data Relay" OpCode
Length	0x03: Number of bytes that follow, NOT including CRC
CommandCode	GWA Command Code
LowByte	Least-significant byte of the Command Code Word Argument
HighByte	Most-significant byte of the Command Code Word Argument
CRC	Modulo-256 addition of all the above (in ASCII)

Response or Event	
AppID, NodeID	[0x80, 0x00]
KSP OpCode	**0x71**: that is the "Data Relay" OpCode
Length	0x03: Number of bytes that follow, NOT including CRC
CommandCode	GWA Command Code
LowByte	Least-significant byte of the Command Code Word Argument
HighByte	Most-significant byte of the Command Code Word Argument
CRC	Modulo-256 addition of all the above (in ASCII)

The format described above is intended to encapsulate the GWA Command Set into the KSP data frame. The BikePCB will automatically add the 0xAA header and the final XOR checksum, before sending data to the GWA Controller.

This method is alternative to the one described in Subsection 7.2.9.: Output to GWA Controller. Please note that the method explained here requires a specific address [0x80,0x00] and the 0x71 OpCode, which does not follow the KSP Acknowledge and Resend scheme, whereas the one in Subsection 7.2.9. is fully KSP compliant.

7.5 ENUMERATION AND SERVICE FRAME

In order to communicate with network devices even if they are not running the KSP stack (for instance, if one device is in Bootload Mode), the same KSP frame format allows for special functions, some of which will be listed below:

Instruction	
AppID, NodeID	[0x80, 0x00]
KSP OpCode	**0x71**: that is the “Data Relay” OpCode
Length	0x03: Number of bytes that follow, NOT including CRC
CommandCode	GWA Command Code
LowByte	Least-significant byte of the Command Code Word Argument
HighByte	Most-significant byte of the Command Code Word Argument
CRC	Modulo-256 addition of all the above (in ASCII)

Response or Event	
AppID, NodeID	[0x80, 0x00]
KSP OpCode	**0x71**: that is the “Data Relay” OpCode
Length	0x03: Number of bytes that follow, NOT including CRC
CommandCode	GWA Command Code
LowByte	Least-significant byte of the Command Code Word Argument
HighByte	Most-significant byte of the Command Code Word Argument
CRC	Modulo-256 addition of all the above (in ASCII)

Enumeration and Service Frames do not use Handshake. The only way to assert that the Service Frame has been correctly processed is the Response Frame that the Destination Node must send back. Any resend policy must be implemented by the Host application and it is out of the scope of KSP.

Below is the list of Service and Reprogramming Frames.

7.5.1 ENUMERATION

Instruction	
AppID, NodeID	Use any AppID NodeID = **0xFF**: sending to all devices of a given AppID
OpCode	**0x02** - GetFW Version
Length	**0x01**
Args	**0x06** - Requested number of response arguments
CRC	Modulo-256 addition of all the above (in ASCII)

Response / Event	
AppID, NodeID	AppID, NodeID from each responding node
OpCode	**0x82** - Answer to 'GetFW Version'
Length	**0x06**
Args	BootLoad FW Version, Main FW Version, EUI-64 LSW
CRC	Modulo-256 addition of all the above (in ASCII)

The enumeration service will trigger responses from all the nodes in the network with the requested Application ID. Each node will report its FW versions (Boot-Flash and Main FW). They will also report the Least-Significant Word (i.e, last 16 bits) of their EUI-64 ID. That allows a quick sampling of which devices are connected (FW major is different for every device type), and how their EUI-64 ID's match their KSP ID's.

The Host may repeat the process if more than one Application ID is present on the network.

Enumeration event may appear regularly, if the device is configured to do so (i.e, if bit 2 on Parameter 0x09 is set). Please refer to Subsection 9.5.2., for more details.

7.5.2 DEVICE BOOT-UP

It is possible to configure any KSP device so that it issues an Enumeration Frame as soon as it boots up. Actually, that is the default configuration. However, the Enumeration Frame comes in a shorter format this time:

Event	
AppID, NodeID	AppID, NodeID
OpCode	**0x82** - Answer to 'GetFW Version'
Length	**0x02**
Args	BootLoad FW Version
CRC	Modulo-256 addition of all the above (in ASCII)

7.5.3 ENTER BOOT MODE

Prior to reprogramming the FW, the device must be reset into Boot Mode:

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x01** - Enter Boot Mode
Length	**0x01**
Args	0xB0 or **0xB1** - Enter Normal mode or **Boot Mode (0xB1)**
CRC	Modulo-256 addition of all the above (in ASCII)

The device will immediately reset, with no time to issue a response.

7.5.4 ERASE FLASH

The erase flash instruction can admit two different frame formats, depending on the number of arguments selected. If 2 arguments are selected, the flash erase will be interpreted as "full erase flash". If 3 arguments are encoded, then the instruction will be considered a "partial flash erase" or ·flash block erase". Both, Slave Controller and BikePCB support the "full erase" instruction, but only the first accepts "partial erase"
Full erase:.

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x23** - Erase Flash
Length	**0x02**
Args	**0x4000** - 16-bit Start Address
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	**0xA3** - Answer to ‘Erase Flash’
Length	**0x01**
Args	0xF0 if all correct; other values (0xF1, 0xF2...) if error
CRC	Modulo-256 addition of all the above (in ASCII)

Partial erase:

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x23** - Erase Flash
Length	**0x03**
Args	Byte1,2: 16-bit Start Address of the block to be erased Byte 3: 0x00 – full erase Other than 0x00 - partial erase. Only the flash-2 block containing the specified Erase Address)
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	0xA3 - Answer to ‘Erase Flash’
Length	0x01**:** Complete Flash2 Erase 0x05**:** Partial Flash Erase

Args	If Length = 0x01 Byte 1: 0xF0: complete erase success.. 0xF1: operation failed. If Lenght = 0x05 Byte 1: 0xF0: partial erase success. Byte 2,3: 16-bit Erase actual Start Address. Byte 4,5: 16-bit Earase actual Final Address.
CRC	Modulo-256 addition of all the above (in ASCII)

Function: in one single instruction, the whole Flash2 section where the regular FW is stored gets erased.

N.B: Flash2 erasure takes a significant amount of time. Please take into account that answer will be delayed in the order of 1 second.

7.5.5 PROGRAM / VERIFY FLASH

There are two different opcodes for Programming and verifying that can be interpreted by both The BikePCB and the Slave controller. The format is the following one:

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x24** - Program Flash **0x25** - Verify Flash
Length	NA > 0x02
Args	Byte 1, Byte 2: 16-bit Start Address Bytes 3 to NA: FW data bytes
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	**0xA4** - Answer to ‘Program Flash’ **0xA5** - Answer to ‘Verify Flash’
Length	**0x01**
Args	0xF0 if all correct; other values (0xF1, 0xF2...) if error
CRC	Modulo-256 addition of all the above (in ASCII)

The SlaveController is also prepared to interpret an enhanced version of the programming instruction that performs both programming and verifying in a single step. The 0x24 overcharged format instruction is the following one:

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x24** - Program and Verify Flash
Length	NA > 0x02 NA <= 0x22 for FSK or UART reprogramming NA <= 0x06 for CAN reprogramming
Args	Byte 1, Byte 2: 16-bit Start Address Bytes 3 to NA: FW data bytes
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	**0xA4** - Answer to ‘Program and Verify Flash’
Length	**0x01 or 0x03**
Args	Byte 1: 0xF0 if all correct 0xF1: writing to Flash2 module failed 0xF2: Start address is not correct 0xF3: Data exceeds Flash2 addressing. If NA= 0x03 Byte 2: CRC8(modulo-256 addition) of all the bytes programmed as read from flash. Byte 3: Verification result. 0x00 indicates the verification has been successful, i.e. the data read from the specified addressing range is the same data contained in the received instruction frame. 0x01 error. At least one bit that was written as one actually reads as zero in the flash. This is a severe failure that will require erasing the affected flash positions, since programming just consists of writing zeroes over an erased flash that only contains ones. 0x02: erros. At least one bit that was written as zero actually reads as one. This error situation may be addressed just by retrying the write sequence (no need to partially erase Flash2).
CRC	Modulo-256 addition of all the above (in ASCII)

7.5.6 READ FLASH

Generally speaking, the FW written on Flash cannot be read. That is why verification is implemented by re-sending the data that we want to check as correctly encoded.

Exceptionally though, only the FW header can be read, in order to learn about the FW version that is currently installed, if any (i.e, one might have erased the FW and not programmed a new one).

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x26** - Read Flash
Length	**0x02**
Args	Byte 1, Byte 2: 16-bit Start Address. Address range: - 0x3200: First section of FW Header - 0x321C: Last section of FW Header
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	**0xA6** - Answer to 'Read Flash''
Length	**0x04**
Args	Bytes 1..4: Section of FW Header
CRC	Modulo-256 addition of all the above (in ASCII)

7.5.7 GET FLASH CHECKSUM

The Slave Controller only is prepared to accept the following instruction that allows to calculate the whole flash-2 16-CRC that is also stored in the Flash2 header. At start-up, the Slave Controller will use this instruction to check against the Flash2 header and make sure the Flash2 that is about to be launched is not corrupted. This instruction can also be launched by the host to perform the same check as the last step of the reprogramming process.

Instruction	
AppID, NodeID	AppID, NodeID
OpCode	**0x28** – Retrieve Flash CRC16
Length	**0x04**
Args	Byte 1, Byte 2: 16-bit Start Address. Lower nibble must be 0x0 Byte 3, Byte 4: 16-bit Final Address. Lower nibble must be 0xF
CRC	Modulo-256 addition of all the above (in ASCII)

Response	
AppID, NodeID	AppID, NodeID
OpCode	**0xA8** - Answer to 'Retrieve Flash CRC16''
Length	**0x01 or 0x06**
Args	If NA = 0x01, Byte 1 is the Error Code - 0xF1: the specified start address is not correct - 0xF2: the specified end address is not correct - 0xF3: start address is greater than end address If NA=0x06 Bytes 1,2: 16-bit Start Address. Byte 3, 4: 16-bit Final Address. Bytes 5,6: CRC16 of the Flash2 range of addresses.
CRC	Modulo-256 addition of all the above (in ASCII)

7.5.8 FSK BEACONS

Event	
AppID, NodeID	AppID, NodeID
OpCode	**0x7E** - MOSI - Beacon from Master (Out) to Slave (In) **0x7F** - MISO - Beacon to Master (In) from Slave (Out)
Length	0x04, 0x06 (typically)
Args	Byte 1: Beacon type Bytes 2 to NA: payload, depending on beacon type
CRC	Modulo-256 addition of all the above (in ASCII)

8 BIKEPCB OPERATION

BikePCB is intended to provide the following functionality:

- Relay data between the Tablet PC (Host) and the peripheral controllers (GWA and Slave Controllers).
- Control bicycle lights, lock and seat as a function of bicycle status (Docking, running, etc).
- Wake up Tablet PC upon bicycle status change (namely when kickstand is removed).

8.1 OFFLINE BEHAVIOR

After the start-up condition reflected by status START has ended, the BikePCB will monitor its digital inputs (both lock and kickstand), the voltage received through its front wheel axis connection, and the communication link status (associated or not). According to the precise combination of all this observed values, the BikePCB machine will jump to the appropriate state among a list of nine possible stable states. These are ASSOC_0V, ASSOC_12V, ASSOC_42V, NOT_ASSOC_12V, PROBE_42V, RUNNING, STANDING, LOCKED, and DOCKED_AND_LOCKED.

If the communication link happens to be established, the reflected status will be either ASSOC_0V, ASSOC_12V, ASSOC_42V. In any of these states, if association is lost because of a proper undock, then the machine will jump to EVENT_DISASSOC and then transit to the corresponding state according to the observed values. But if the communication link is lost unexpectedly, then the BikePCB will flow to ERROR_ASSOC_0V, ERROR_ASSOC_12V or ERROR_ASSOC_42V accordingly, and immediately after to NOT_SLAVE in order to signal the unexpected lost of communication with the Slave Controller.

Through the observation of the communication link parameters (SyncTimer) the Bike_PCB will keep track of its condition as to be considered docked and identified by the system, or not. According to this information, in case 42V are detected and the communication link is not working, it is possible to distinguish between the two following cases. The first is the communication link is not working because the bike has not been yet associated. In this case 42V_PROBE state will be reported. The second is communication link is not working due to a communication error, but the bike had been previously properly associated. In this new case UNCOND_CHARGING state will be reported.

Once in NOT_SLAVE the state machine will remain there until either a reset occurs, the communication link is recovered, voltage from the docking point is sensed, or an instruction from host instruct to leave the mode. In NOT_SLAVE state, the Sync Timer in the node table will read 0xFF but the Slave Controller EUI-64 and status will still be available. The node table information about the Slave Controller will be cleared if the bike is instructed to exit the state by any host instruction. Also, at the reception of this instruction, if there is no tension detected and the communication link with the Slave Controller is not working, the Bike_PCB will consider to be free and forget its previous docked and identified by the system condition.

In off-line mode, the BikePCB status machine will survey the digital input of the lock and the lamps current consumption to signal an alarm event if necessary.

If the lock is expected to be closed and the lock digital input reads opened, then the Bike will transit briefly to ERROR_SENSORS to inform the host. If the opposite is the case, the lock is expected to be opened and the digital input reads closed, then the machine will transit briefly to ERROR_LOCK_TAMPER.
If the lamps current consumption whenever lamps are supposed to be on is below the configured value in paramenter 0x16- MIN_LAMPS_CURRENT, the BikePCB will briefly transit to ERROR_LAMPS_UNDERCURRENT. Similarly, if the current exceeds the value stated in 0x17- ERROR_LAMPS_OVERCURRENT, then the transited status is ERROR_LAMPS_UNDERCURRENT.
All these alarm event status are brief transitions followed by the corresponding event back to the stable state. The alarm events are rised about every 2 minutes while alarm condition persists.

8.1.1 BIKEPCB STATES

BikePCB state machine is implemented through the following states and values. In the previous section all off-line used status have been introduced. In the table below there are several states that belongs to the on-line operation that can only be triggered by the Host (in normal operation or in service or test modes). These states appear high-lighted. The rest are involved in the off-line operation.

Table 1: BikePCB States

Value	**Name**	**Description**
0x00	ASSOC_0V	Bike has detected KSP beacon from Slave Controller. V(PWR+) = 0V
0x01	ASSOC_42V	V(PWR+) = 42V. Association completed
0x02	ASSOC_12V	V(PWR+) = 12V. Association completed
0x09	EVENT_DISASSOC	Bike has been disassociated. It will soon go into running mode (or else, Host may force it)
0x10	RUNNING	Bike is running, lamps are on [1]
0x20	**CMD_MAKE_LOCK**	Start locking sequence (TabletPC-triggered)
0x23	**USER_LOCKS**	Locking sequence awaiting user intervention (move lock lever)

[1]BIKE_RUNNING state may only be forced by the Host if BikePCB enters an exception state (such as an error condition) or has entered a transitional state, while disassociated (e.g, BIKE_DISASSOC)

0x28	LOCKED	Bike is locked
0x29	LOCKED_WAKEUP	Kickstand has been retracted while bike is locked.
0x2A	**UNLOCK**	Start "Open Lock" sequence (TabletPC-trig'd)
0x30	STANDING	Kickstand is deployed (mechanically-triggered)
0x40	NOT_SLAVE	No communication with Slave Controller (normally due to power failure on the station)
0x41	UNCOND_CHARGING	No communication with Slave Controller, but 42V are on, due to unconditional charging
0x50	NOTASSOC_12V	V(PWR+) = 12V. Association pending
0x51	42V_PROBE	No communication with Slave Controller, but 42V are on, due to probing.
0x60	**SEAT_OPERATE_DOCKING**	Activates seat operation while in Docking Point
0x61	**SEAT_OPERATE_STANDING**	Activates seat operation out of Docking Point
0xC0	START	Boot-up state, while determining digital inputs. Will transition to BIKE_RUNNING, BIKE_ STANDING or BIKE_LOCKED
0xCC	**CMD_ONLINE_TEST**	Allows online control on Digital Outputs (mainly, the lock)
0xCE	**CMD_HW_TEST**	HW test: Continuous, Repetitive Operation (CRO), intended for EMC or wear-out tests.
0xD0	**SERVICE_MODE_OK**	Bike will shine its lamps continuously, to signal to the service staff that bike is OK
0xD4	**SERVICE_MODE_FAIL**	Bike will blink its lamps on and off, to signal to the service staff that bike has some failure
0xE0	ERROR_SENSORS	Lock status unexpectedly unlocked.
0xE1	ERROR_ASSOC_0V	KSP link failed during docking
0xE3	**ERROR_LOCKING**	User did not move the lever, therefore locking sequence did not end properly
0xE4	ERROR_ LOCK_TAMPER	Lock status unexpectedly locked.
0xE5	ERROR_UNLOCKING	Digital Input does not report "Unlocked" status after Unlock operation.
0xE6	ERROR_ASSOC_42V	KSP link failed while 42V are on

0xE7	ERROR_ASSOC_12V	KSP link failed while 12V are on
0xE8	DOCKED_AND_LOCKED	Bike was inserted into the docking point while locked
0xEA	ERROR_LAMPS_OVERCURRENT	While lamps are expected to be on, the current consumption is over the configured threshold in Paramenter 0x17 (MAX_LAMPS_CURRENT).
0xEB	ERROR_LAMPS_UNDERCURRENT	While lamps are expected to be on, the current consumption is below the configured threshold in Paramenter 0x16 (MIN_LAMPS_CURRENT).
0xEE	**ERROR_LOCK_CURRENT**	Current delivered to the Lock is higher than *Parameter 0x15* (MAX_LOCK_CURRENT)
0xF0	ERROR_MACHINE	The Bike main status machine has detected an unrecoverable error and will reset the BikePCB

8.1.2 ACTUAL OPERATION

Now that the states have been introduced, let's review the typical scenarios:

Running:

While the bike is in operation, state will be RUNNING. Front and rear lights will be on continuously.

Standing / Locking

If the user stops the bike, he or she has the option to lock it. First, we assume that the kickstand is deployed, so we move from RUNNING to STANDING. Then, the bike may be locked through the Host Application running on the Tablet PC. The Host Application must set the state MAKE_LOCK. After a couple of seconds, an event will appear showing USER_LOCKS. The Host Application may prompt the user to act on the locker's lever, to secure the bike. If that is done, another event will show LOCKED, thus completing the Lock operation. Otherwise, if the user does not act on the lever, the lock motor will automatically revert to lock condition after a while. The BikePCB will automatically detect the motor activity, signal ERROR_LOCKING and will revert back to STANDING. Anyway, if after USER_UNLOCK_TMO seconds (CFG_Param 0x13), motor activity has not been detected, then ERROR_LOCKING event will appear, and then back to STANDING.

Once the bike is locked, it may only be unlocked through the Host Application, by setting UNLOCK state. Normally, that will bring us back to STANDING. Then, the user may fold the kickstand, to resume RUNNING.

The Lock Digital Input is polled all the time. If its value goes unexpectedly to lock (high), that probably means that the lock has been cut off. That is shown by rising the ERROR_LOCK_TAMPER event every 2 minutes, without changing the state. If Lock Digital Input goes unexpectedly to unlock (low), the system will signal ERROR_SENSORS every 2 minutes while error condition is present, without changing the state. The error will disappear as soon as the sensor proper functioning is re-established.

The lock operation is also controlled by the current sensor:

- If no current is sensed after MAKE_LOCK, it does not make sense to wait for the user to act the lever. Instead, we go back to STANDING.
- If no current is sensed after UNLOCK, we must assume that we are still in LOCKED, so that the operation can be repeated.

Docking (default)

Normally, the battery is full enough to power the Tablet PC and BikePCB. In that case, as soon as the user pushes the bike into the docking point, the Slave Controller will act on the solenoids to dock the bike. After that, FSK beacons are sent to BikePCB and association is completed in a few seconds.

Therefore, the default transition goes from RUNNING to ASSOC_0V. A few seconds later, 42V are sensed on PWR+, and an event tells us about ASSOC_42V.

If the user docks the bike with its lock engaged, BikePCB will report 0xE8, LOCKED_AND_DOCKED. The Host Application is supposed to unlock the bike automatically (it is safe to do it, since the bike is docked).

Docking (sleep mode)

If the battery is depleted, the Host Application may have decided to turn off the battery controller while the bike is not docked. In that case, the user will be cycling with no motor assist and eventually will push the bike into the docking point to return it.

As usual, the Slave Controller will sense that the bike has been introduced, solenoids will be triggered to dock the bike and FSK beacons will be sent. However, BikePCB is not powered; therefore it will not respond to Slave Controller.

Slave Controller will perform the impedance test at 12V and then try sending FSK beacons while 42V are on in PWR+. Both changes will be announced as events (NOTASSOC_ 12V, NOTASSOC_42V), but the Host Application is not expected to run yet (Tablet PC still powering up). When the Host Application is up and running, the state shown on BikePCB is most likely ASSOC_42V (i.e, FSK association is done and 42V are on PWR+), since the whole probing process takes about 10 seconds.

States NOTASSOC_12V and NOTASSOC_42V_PROBE rapidly switch bike lamps on and off, as the only way to provide feedback to the user that the electronics are powering up again (there is also a tiny green LED on the Tablet PC).

Charging

When we are in the state ASSOC_42V, charging may only stop by acting on the Slave Controller (or the Slave Controller will act by itself). Please refer to the Bike Charging Module Subsection 9.3. for further detail on interactions with Slave Controller.

Eventually, the only way out of ASSOC_42V is ASSOC_0V, which will appear as an event.

Unconditional Charging

During NOTASSOC_42V state, switching noise on the powerline might get high enough as to prevent communications between BikePCB and Slave Controller. If that happens, both BikePCB and Slave Controller will report ERROR_ASSOC_42V. Slave Controller will disable charging, so that the Host Application on the TabletPC can manage the situation. If the Host Application decides to enable Unconditional Charging, synchronization may be lost again. Then, disassociation will take place immediately and BikePCB will enter state 0x41, NOT_ASSOC_42V until communication can be restored.

Undocking

Same as above, undocking operation is managed by the Slave Controller (see Subsection 9.4.). In normal conditions, Slave Controller will issue a disassociation beacon to BikePCB, which the latter will acknowledge.

As soon as BikePCB is disassociated, it will move from ASSOC_0V to RUNNING, and lights will switch on again.

If the docking solenoids are released but disassociation beacon did not achieve its goal, BikePCB will still show ASSOC_0V. Only after a few seconds will it report ERROR_ASSOC_0V and immediately after NOT_SLAVE. Then, the Host Application is expected to resolve the situation: if it has triggered an undock operation, it must set state RUNNING.

Another reason for ERROR_ASSOC_xxV and NOT_SLAVE to appear, would be that the bike has been forced out of the docking point by burglars. In that case, probably the feedback from the GPS or the pedal sensors can confirm that the bike is moving or being moved, and trigger the corresponding alarms to the maintenance service.

Finally, if Slave Controller runs out of power or it is in Boot Mode for FW reprogramming, ERROR_ASSOC_0V will appear immediately transiting to NOT_SLAVE.

Exit from Reset

When BikePCB boots up, it decides which state it is in, based on three considerations:

- Voltage at PWR+: check whether any voltage is applied from the Docking Point.
- Association frames: check whether any communication from Slave Controller is received.
- Digital Input Status: check whether the kickstand or the lock are active.

If BikePCB is associated, then the state will be ASSOC_0V or ASSOC_42V, depending on the voltage value at PWR+.

If BikePCB is not associated but PWR+ shows 12V or 42V, then we are in the process to associate. No events will be issued to the BikePCB host until associated (ASSOC_42V), or rejected (NOTASSOC_42V).
Finally, if BikePCB is not associated and PWR+ does not show any voltage, we must assume RUNNING. However, if the Kickstand is deployed or the Lock is engaged, status goes to STANDING or LOCKED, directly after Reset.

Digital Inputs are still evaluated if the bike is docked. In particular:

- Kickstand may be lowered, thus a Wake-up signal to the TabletPC is sent.
- If lock is found to be engaged, an error condition (LOCKED_AND_ DOCKED) prompts the Host Application to disengage the lock.

Service Mode

The Host may set states SERVICE_MODE_OK or SERVICE_MODE_FAIL to show to the maintenance staff whether the bike is in good shape. If SERVICE_MODE_OK, the front and back lights will switch on continuously (bike is supposed to be docked or standing). If in SERVICE_MODE_FAIL, lights will blink on and off. The Host may then set BIKE_START to exit Service Mode, otherwise that will happen automatically after a configurable time-out has expired (see Subsection 8.4.1.).

8.2 SEAT BEHAVIOR

Actuator considerations

The Seat control module implemented in the BikePCB is intended to control the actuator CONCENS CON35 with a Gear ratio of 1:71, provided with a hall sensor that generates a pulse every 0,0282 mm.
The nominal operational voltage/current is 12V / 3.6A , but due to current limitations of the bike, it will be operated at 8V (CONCENS recommendation) and a current limitation of 800mA. At this rating, the load is expected to be less than 300Newton, as this is the value specified for 800mA @12V. The measured speed is 2,8 mm/s.

Due to current limitations it is important to pay special attention to both extreme positions (completely retracted / completely extended). In both ends, the actuator gets stuck, and the following considerations has to be taken into account to operate it safely.

Bottom end:

A brand new actuator is supplied completely retracted, and tightly stuck at this position. To deploy it, a higher current than the bike limitation is needed. So before mounting it in the bike it has to be connected to a power supply to deploy it a few centimeters.

Once installed in the bike, and driven by the BikePCB firmware, the actuator will be able to reach the lower end without getting stuck.

Upper end:

Upper end has to be avoided. This can be easily achieved if the calibration process is always initiated with a down movement to the lower end. This operation will enable the Seat Operation Module to prevent the actuator to reach the upper end. See the calibration procedure for details.

Operation Description

The Seat Control Module can only be operated form SEAT_OPERATE_DOCKING (0x60) and SEAT_OPERATE_STANDING (0x61) bike states. To transit to these states a SetBikeStatus instruction has to be used.

When issued from DOCKED_AND_LOCKED, NOT_SLAVE, or any state that signals association or that implies the bike is being powered by the DP, it will be possible to transit to SEAT_OPERATE_DOCKING.

When issued from STANDING, LOCKED, LOCKED_WAKE_UP or NOT_SLAVE then the transit to SEAT_OPERATE_STANDING will be accepted.

Once in any seat operate states, it will be possible to remain there while operating the seat, but after 10 seconds without activity, the BikePCB will leave the seat operating state and will return to the state from it was called. Optionally, every time an instruction is send it is possible to specify a lower timeout in case it is needed to exit the seat operating mode faster.

SEAT_OPERATE_DOCKING and SEAT_OPERATE_STANDING states the Seat Control Module can be accessed through the KSP instruction SeatOperate (Opcode 0x3E) and its corresponding answer AnsSeatOperating (OpCode 0xBE). Through this KSP instruction 6 different seat module specific instructions can be accessed. These are SeatStop, SeatMoveRelativeUp, SeatMoveRelativeDown, SeatGoAbsolute, SeatFastCalibrationEnable/Disable, and SeatCancelAbsoluteMode.

In SEAT_OPERATE_STANDING status a kickstand survey will be performed. In case of retraction any seat operation will be stopped (as if a STOP instruction had been issued) and the BikePCB will transit to RUNNING.

After reset, the SeatOperation Module has no means to determine the absolute position of the bike seat. It is said it is in relative positionning mode. In this mode only relative movements up/down are accepted. After callibration, the process by which the absolute position can be known, the Seat Control Module will operate in absolute mode. This means that apart from the relative movements, it will be prepared to accept instructions indicating an absolute final position. In absolute mode the bottom end correspponds to position 0x00, and the uppermost safe operating end corresponds to 0xFF. In relative mode, the length of the step is maintained.

Here follows detailed operation of each instruction, including codification of Parameter_1, and how to interpret Result_1 and Result_2 bytes that comes together with the answer.

SeatStop:
Issuing this instruction when the Seat Control Module is moving up/down will immediately cancel the movement. Parameter_1 does not matter. The AnsSeatOperate frame will report the cancelled instruction code, not the SeatStop instruction code. Issuing the instruction when stopped will have no efect on the seat movement. In this case the AnsSeatOperate frame will indicate the SeatStop instruction code, Result_1 will be always 0x00, and Result_2 will be also 0x00 if in relative mode, or will be a number between 0x00 and 0xFF indicating the absolute position of the bike seat if in absolute positionning mode.

SeatMoveRelativeUp/SeatMoveRelativeDown:

Issuing this instruction will start the movement in the specified direction up/down for the amount of steps indicated in Parameter_1. Just as a rule of thumb, each step is a little bit less than a millimeter. The AnsSeatOperate frame will indicate in Result_1 the number of steps already done, and in Result_2 will indicate the maximum number of steps it is expected to perform. This last number will be no higher than the value specified in Parameter_1, but will not always equate Parameter_1. The reason is that the Seat Control Module keeps track of all previous movements and is aware of the lowermost and uppermost reached positions. According to the history, if the lowermost reached position is say 50 steps downwards, it is clear that it is not acceptable to rise 255 additional steps. The number in Result_2 is an expected value and may not be the final number of steps done. The movement may stop after completition of all expected steps, or because a mechanical constrain has been found. The detection of mechanical limitations is performed through the permanent supervision of the motion speed. Falling below 2.5 mm/s causes the operation to stop.

SeatGoAbsolute

Issuing this instruction will only be accepted if in absolute positionning mode. This instruction will start the movement up/down to reach the absolute position specified in Parameter_1. The AnsSeatOperate frame will inform of the execution progress. Result_1 will provide the present absolute position whereas Result_2 will contain the final absolute destination position. The movement will end by the same reasons explained in the previous instruction.

SeatFastCalibrationEnable/Disable

As explained, the Seat Control Module keeps track of all previously performed movements, recording the lowermost and uppermost reached positions. When the distance between both extreme position is 0xFF, that is the full length of the seat actuator range, the Seat Control Module automatically transits to absolute positioning mode. This fact can be used to calibrate the seat in order to admit SeatGoAbsolute instructions. To do this callibration the first step is to issue a relative downwards movement of 0xFF steps, and immediately after, a relative upwards movement also of 0xFF steps. After both movements are completed, if no mechanical constrains have been found, the seat will operate in absolute mode.
Since this calibration is time consuming, there is also the possibility to perform a faster callibration. The first required step is to habilitate the fast callibration using the present instruction. Then a relative downward movement of 0xFF steps must be issued. When the movement stops because of a mechanical constrain, the Seat Control Module will consider it has reached the bottom end. This will be set as absolute zero position and from that moment on, SeatGoAbsolute instructions will be admited.
To issue the SeatFastCallibrationEnable/Disable instruction the Parameter_1 must be selected in the following way. Specify a 0x00 to disable the mode. Any other value will enable it, but only if in relative mode. Once in absolute mode, as callibration is already done, it makes no sense to enable the mode. The first byte of the AnsSeatOpertion frame will contain all the information needed to analise the operation result. As no extra data is needed, Result_1 and Result_2 will read 0x00. The effect of this instruction is to set the Seat Control Module in

absolute positionning mode as soon as the bottom is reached, that is, a constrain has been found in any downward movement after the fast callibration has been enabled.

SeatCancelAbsoluteMode
Since the fast callibration mode considers that any constrain in the downward movement equals reaching the bottom end, although it is the most likely cause, it may also not be the case. So it will wrongly enter absolute mode. This instruction is intended to correct this misunderstanding, in order to allow a new callibration to be done. This instruction does not care of Parameter_1 value, and when AnsSeatOperate frame is send, Results_1 and Result_2 will both read 0x00.

Whenever the seat is not being operated, whatever the BikePCB status may be, the status of the seat can be polled using the KSP Instruction SeatPoll (Opcode 0x3F) and its corresponding answer AnsSeatPoll (OpCode 0xBF). See Bike KSP Instruction description for details.

8.3 ENERGY MANAGEMENT

In terms of energy management, the Tablet PC may be sleeping when in “Locked” or “Standing” modes.

- When transitioning from “Standing” to “Running”, BikePCB will wake up the Tablet PC.
- When LOCKED, the Tablet PC may be in Sleep Mode too. In order to restart, the kickstand must be lifted by the user. That will make BikePCB wake up the Tablet PC, which will then ask the user for the PIN to unlock the bike.

Opportunistically, sleep mode may be entered when in “Docked” or “Charging” modes, although that is not critical, due to supply availability from the docking module.

BikePCB has HW and FW mechanisms available, which allow it to monitor digital inputs, voltages and incoming instructions even during sleep mode.

8.4 BIKEPCB CONFIGURATION

The configuration of BikePCB is stored in EEPROM memory, which gives it around 10,000 read/write cycles. This has to be taken into account when managing the configuration parameters. Default configuration values should be the right ones for BikePCB operation, anyway.

8.4.1 PARAMETER ARRAY

BikePCB is configured from the following configuration parameters that are sent using command WriteCFG_Byte and can also be read using command ReadCFG_Byte.

All the device's parameters are listed below:

Table 2: List of parameters of the BikePCB

Num	Description	Default Value
0x01	CFG_UART_HOST – Determines the baud rate of communication with the Host. Protocol: KSP-UART 0x20: 9600 baud 0x21: 19200 baud	0x21
0x02	ICHAR_TMO_HOST – It determines the maximum standby time between consecutive characters of communication with the Host to be interpreted for the same command frame Allows any value between 0x04 and 0xFF, and the standby time of the value of this byte is multiplied by 5 ms	0x0A
0x03	CFG_UART_GWA Determines the baud rate and protocol of communication with the GWA controller. The only admitted value is the default one.	0x45
0x04	ICHAR_TMO_GWA – It determines the maximum standby time between consecutive characters of communication with the GWA to be interpreted for the same command frame Allows any value between 0x04 and 0xFF, and the standby time of the value of this byte is multiplied by 5 ms	0x50
0x05 to 0x06	RFU – Reserved for future use.	
0x07	CFG_DIN_ACTIVE_STATE – Specifies whether Digital Inputs are Active-High or Active-Low	0x03

	Each digital input may be defined active-high or active-low, and that depends on the hardware. - Bit 0 (least-significant): Kickstand - Bit 1: Lock If corresponding bit is cleared, digital input is active-low. If set, digital input is active-high. Our current default (0x03) means that both Kickstand and Lock are active-high.	
0x08	RFU – Reserved for future use.	
0x09	CFG_ECHO – It defines the events that are reported to the Host. •Bit 0:Reserved •Bit 1: If set, events are reported to TabletPC by default (e.g, at startup), else to Slave Contr. •Bit 2: If set, event is sent to default Host at startup. •Bit 3: If set, KSP Associate Request is sent to default Host at startup. •Bit 4: If set, FSK beacons are echoed to Host Bits 5,6,7: Reserved.	0x02
0x0A	DIN_VALIDATION_TMR – It tells how many milliseconds must elapse with the Digital Input active, before it is considered valid (units: 5 ms). A Digital Input is only considered valid when some time has elapsed, while keeping its active value. Default value is 500ms	0x64
0x0B	RFU – Reserved for future use.	0x60
0x0C	RFU – Reserved for future use.	0x0A
0x0D	WAKEUP_PULSEWIDTH – Activation time of nWAKE signal to Tablet PC (units: 100 milliseconds). Legal values: 0x01 to 0xFE	0x0A
0x0E	RFU – Reserved for future use.	0x20
0x0F	LOCK_TIME_A - Lock operation is internally organized as multiple slots of this value. Do not modify unless agreed with Kimaldi.	0x14
0x10	BIKE_UNDOCK_TMO - Time interval between disassociation and running (units: 100 millisec.).	0x14
0x11	RFU – Reserved for future use.	

0x12	SERVICE_MODE_TMO - Time interval allowed for the Service Mode (units: seconds). After this time, BikePCB goes back to BIKE_START.	0x20
0x13	USER_LOCK_TMO - Time interval allowed for the user to move the lock lever (units: seconds) in case the lock motor movement is not detected. The AXA Lock allows 14 seconds to the user, to push the lever into the lock. If not done within this period, lock goes back to its unlocked position and BikePCB will detect this motion. In case the reversal movement could not be detected, this parameter indicates a timeout.	0x1E
0x14	MIN_LOCK_CURRENT - Lock movement is not considered to occur if sampled current is below this threshold. A current-sense loop will tell us whether the lock motor is moving. If measured current is below this configuration value, the lock is not considered to be moving (or not connected at all)	0x08
0x15	MAX_LOCK_CURRENT - Lock is considered to be short-circuited if sampled current is above this threshold. If the measured current goes above this threshold, a short circuit is considered to have occurred.	0x80
0x16	MIN_LAMPS_CURRENT - When lamps are switched on and the current flowing is below this configuration value, lamps are considered to be unplugged.	0x19
0x17	MAX_LAMPS_CURRENT - When lamps current consumption exceeds this configuration value, a short circuit is considered to have occurred.	0xB0 for (.33.) 0xF5 for (.32.)
0x18	SEAT_LEN_CONV – Value to define the operative SEAT shaft length. CFG_PARAM = SEAT shaft length [mm] / 7.219 Range value: 0x0E ~ 0x21 If the value configured in this parameter is out of range, the default value is assumed.	0x14
0x19 to 0x24	RFU - Reserved for future use.	

8.4.2 CONFIGURATION OF COMMUNICATION WITH THE HOST

The behaviour of the Virtual COM Port is defined from parameters 0x01- CFG_UART_HOST, 0x02-ICHAR_TMO_HOST and 0x09- CFG_ECHO of the configuration.

CFG_ECHO parameter allows us to define a start-up event, either to the Tablet PC through Virtual COM (hence the default value, 0x02), or to the Master Controller, through the Slave Controller (FSK, then CAN bus), which requires a value of 0x08.

There is another parameter that affects communications although it is not affecting directly the UART's. It is 0x10-BIKE_UNDOCK_TMO. When the communication link has been broken, communications will not try to establish them until the timeout configured in this parameter expires. This is to prevent re-association immediately after the undock sequence is finished.

8.4.3 TABLETPC WAKE-UP CONFIGURATION

At times, Tablet PC may fall into Sleep Mode. In that event, communications with BikePCB are halted. The BikePCB will sense its digital inputs an emit a pulse to wake-up the tablet at any change. The pulse width is configured in parameter 0x0D- WAKEUP_PULSEWIDTH.

8.4.4 SERVICE MODE CONFIGURATION

For convenience, behavior during Service Mode may be configured. Parameter 0x12- SERVICE_MODE_TMO determines the time span of the service mode condition.

8.4.5 HARDWARE CONFIGURATION

Parameter 0x07-DIN_ACTIVE_STATE instructs the BikePCB whether to use positive or negative logic to interpret the active or inactive conditions of Lock and Kickstand Digital Inputs.

8.4.6 LOCK CONFIGURATION

Lock behavior and self-testing can be configured through the following parameters, 0x0F-LOCK_TIME, 0x13-USER_LOCK_TMO, 0x14-MIN_LOCK_CURRENT, and 0x15-MAX_LOCK_CURRENT.

8.4.7 LAMPS CONFIGURATION

Lamp behavior and self-testing can be configured through the following parameters, 0x16-MIN_LAMP_CURRENT, 0x17-MAX_LAMP_CURRENT. These parameters affect the overcurrent and undercurrent alarm events that operate in off-line mode.

9 SLAVE CONTROLLER OPERATION

The Slave Controller operation is performed by four different software modules that activate immediately after START state:

Bike acceptance module: Its purpose is to determine through docking sensors and solenoids sensors the presence of a bike in the docking point and decide whether it has to be docked.

Bike identification module: Once the bike is docked, this module purpose is to establish the FSK communication link that will allow bike identification.

Bike charging module: After the bike has been correctly identified, the Bike charging module will proceed to charge the bike battery.

Bike release module: the purpose of this module is to release the bike to allow its physical withdrawal from the docking point.

Each module consists of a number of states of the SlavePCB state machine. Here is the detailed functioning of every module and its states.

9.1 BIKE ACCEPTANCE MODULE

This module has two different algorithms to determine whether a bike has to be accepted. We can name them as “algorithm to accept already present bikes” and “algorithm to accept newly arrived bikes”. Both algorithms are initiated in the IDLE state.

The “algorithm to accept already present bikes” will accept the bike whenever at least one of the docking sensors is detecting the bike presence, or at least one of the solenoid sensors reports a locked condition, just after SLAVE_START (state machine initialization). If accepted the SlavePCB will jump to the MAKE_DOCK state and will signal an internal flag called “Error_not_bike_allowed” as false. Otherwise the algorithm will quit and call the “algorithm to accept newly arrived bikes”, because as there is not bike present this algorithm makes no sense.

The “algorithm to accept newly arrived bikes” will accept a bike only if both docking sensors raise the activation event within a configurable time window specified in CFG_PAR. PA0.DIN_VAL, typically 0.5 sec to 1 sec. This condition is supposed to be easily verified by a real bike being pulled into the docking point, but is less likely to pass in case the sensors are manipulated by hand or other objects. If this is the case, the Slave Controller will jump to the SLAVE_ERROR_SOLENOID_SENSORS state and will remain there until the object is withdrawn and both docking sensors report no presence. While waiting, the LEDs will flash. If CAN backchannel is activated, the event will be sent to the peer slave. While sensors report no presence the Slave Controller will remain in IDLE state, and if bike is accepted, it will jump to the MAKE_DOCK state and will signal the internal flag “Error_not_bike_allowed” as true.

The “algorithm to accept already present bikes” is selected at start-up, and may also be called from the Bike release module as it will be explained later. The “algorithm to accept newly arrived bikes” is always triggered by the previous algorithm whenever it fails to accept (can’t find a present bike), and may also be triggered by the Bike release module.

The MAKE_DOCK state purpose is to activate the solenoids to lock the accepted bike into the docking point. First will apply an electrical pulse of a configurable duration to the right solenoid and check if it has closed properly. If the solenoid is not detected, it will be retracted and released again. This ensures the solenoid ends in the desired position despite a solenoid sensors failure. Immediately, on the left solenoid will repeat the described operation. The docking operation should be as fast as possible to prevent the wheel axis to be withdrawn from one side or both immediately after the bike has been accepted. Because of power consumption limitations it is not possible to operate both solenoids simultaneously, and one has to be locked after the other. To speed up the overall sequence, the duration of the electrical pulse applied to the solenoids will be optimized if the expected transition from open to close has been correctly reported by sensors before the nominal pulse duration exhausts.

The acceptance process is successful when both solenoid sensors end up detecting the locked condition. In that case the Slave Controller state machine will launch the Bike identification module. If it fails, the flag “Error_not_bike_allowed” will be evaluated. This flag indicates whether any automatic off-line release is allowed or not, and it takes its name after the ERROR_NOT_BIKE state, in which the most important automatic release decision making takes place, as it will be explained later. So if this flag is set to true, the Slave Controller will jump to the ERROR_DOCKING state and from there to the Bike release module since there is no certainty that the bike has been properly fixed. If the flag indicates false, the Slave Controller state machine will proceed as if a successful acceptance were the case.

9.2 BIKE IDENTIFICATION MODULE

The first state of this module is called FSK_0V_PROBE. As its name indicates it will try to establish a FSK link without applying any tension to the bike axis. If successful, it will jump to BIKE_DOCKED state. If not, the most likely failure cause is that the present bike is a depleted one. As it has no means to power its electronics on its own, no link can be established. To overcome this situation the Slave Controller will try to power the bike using the charger tension. Since the voltage is high and so is the current that may flow, before connecting the charger, the Slave Controller will check for short-circuit presence. To perform this test it will jump to FSK_12V_PROBE state.

In FSK_12V_PROBE state 12 volt are applied while current is continuously monitored. If a configurable current level is not exceeded, it will jump to the FSK_42V_PROBE state. Otherwise, the over current condition will be raised and this will cause a jump to FSK_PROBE_ERROR state. At this point the flag “Error_not_bike_allowed” is evaluated. If true, as its name indicates the program flow will be transferred to the ERROR_NOT_BIKE state, to signal the error condition, and from there will jump to the Bike release module. If false, the bike cannot be released. To proceed the docking sensors will be evaluated. If the bike is detected it will jump again to FSK_0V_PROBE after a few seconds for the process to be reinitiated, except that the impedance test will not be repeated. After the identification cycle has been retried for three times, the Slave Controller will wait a number of minutes specified in CFG_Parameter 0x1D (default: 30 minutes) in FSK_PROBE_ERROR state before jumping to FSK_0V_PROBE. This delay is intended to allow charging time to peer Slave Controller and will be inserted after every three retrials. While waiting the FSK link will be monitored. If the bike suddenly associates, the Slave Controller will leave the FSK_PROBE_ERROR and jump again to the beginning of the identification cycle. While waiting, the docking sensors will also be monitored. If the bike is detected but there is a change in the docking sensors signals (i.e at the beginning both sensors where activated and then only one is, or the opposite is the case, or at he beginning only right sensor detected bike presence and then only left one does, or the opposite is the case), the identification cycle will be initiated.

If in FSK_PROBE_ERROR state, both sensors signal bike absence, then the FSK will enter an inhibited state where no Keep Alive frames will be send. This is to avoid that a running bike that enters the D.P. is associated without being docked.

In FSK_42V_PROBE the Slave Controller will try again to communicate with the bike via FSK. If successful the state machine will jump to BIKE_DOCKED to signal the success. If failed, either because the 42V power is not available, or because of a communication error, it will jump to FSK_PROBE_ERROR, where the flag “Error_not_bike_allowed” is evaluated and the Slave Controller program flow will be diverted to ERROR_NOT_BIKE or to FSK_0V_PROBE accordingly.

The BIKE_DOCKED status is the successful end of the Bike identification module. Its purpose is to signal the success of the identification process, and to transfer the program flow to the charging module.

In case the FSK link worked at 0V, it will go to the Bike charging module and put the bike in charge if the configured delay specified in parameter PB0.CHARGE_DEL for dock-to-charge is neither 0 nor 0xFF. If any of these values are specified, the Slave Controller will remain in BIKE_DOCKED state. In that case a continuous surveillance of the association condition will be performed, and in case of losing the FSK link, it will jump to the ERROR_ASSOC state. If the transfer is to be made to Bike charging module, the effective delay will be the specified in the configuration plus an additional one that is function of the Node-Id.

If 42V were required to keep the link, the transfer to Bike charging module will result in an unconditional charging operation of 5 minutes to allow tablet boot up. In any case the flag “error_not_bike_allowed” will be set to false. As the bike has been successfully identified, any automatic release is forbidden, including the case of Slave Controller is reset.

The ERROR_ASSOC state signals the lost association condition and perform a jump to FSK_0V_PROBE state to recover the FSK link. From there, if association still fails, it will continue as explained above through FSK_42V_PROBE and cycle again if necessary.

9.3 BIKE CHARGING MODULE

This module has two different algorithms to perform the bike charging operation. One can be called “normal charging” and the other “unconditional charging”. When entering this module, one option has to be selected. Both algorithms will provide charge to the docked bike, and both will preserve the FSK link established in the previous module, in order to keep the associated condition. One of the differences between the two options stands in the association lost condition managing.

Consider the immediate step after docking. If a bike was docked without being powered by the docking point, it means the bike can keep the link on its own, and so, a restrictive error treatment can be performed. In case the bike needed external power to be docked, the Slave Controller has to be tolerant, since the bike ability to keep associated may not be solid enough. In the first case “normal charging” will be applied, whereas in the second the “unconditional charging” will be selected.

The “normal charging” algorithm is initiated in the START_CHARGING state, where after the above explained configurable time span, the Slave Controller will try to gain the charger. As the charger is a shared resource, it may be gained and then the Slave Controller will transit to the BIKE_CHARGING state, or it may be not accessed, and then it will enter the CHARGING_WAIT state from which it will try to gain the charger every 10 seconds. The waiting condition will be signaled periodically through events every 30 seconds, and also by means of the LEDS.

Once in BIKE_CHARGING state LEDS will flash differently to signal this state, and several conditions will be continuously evaluated in order to determine when to exit.

The first of these conditions is the charge over current. If this is the case the Slave Controller will jump to ERROR_CHARGING_OVERCURRENT state, disconnect the charger voltage and survey/retry association. If association is not possible to obtain, it will jump to the ERROR_ASSOC state. The second condition is association. If lost the Slave Controller will jump to ERROR_ASSOC_42V, disconnect charger voltage and survey/retry association. If association is not possible to obtain, it will jump to the ERROR_ASSOC state. (The old parameter P1C.ERR_ASSOC is not used anymore). The third condition has to do with 42V. If 42V are lost, i.e. undetected for a period of one second, the Slave Controller will jump to SLAVE_ERROR_CHARGING_VOLTAGE state, survey/retry association. If association is not possible to obtain, it will jump to the ERROR_ASSOC state. The forth condition is battery full. In case the charge current falls below a configurable level while charger tension is applied, the Slave Controller will consider the bike battery is completely charged and will transit to CHARGING_STOPPED. The last condition is the loss of the charger resource due to arbitration. If the peer Slave Controller needs the resource with higher priority, then the cedent Slave Controller will jump to CHARGING_WAIT until the charging operation can be resumed.

In case none of the previous exit condition is verified during a 5 hours period, it will stop charging due to timeout and jump to CHARGING_STOPPED.

The "unconditional charging" algorithm is initiated in the UNCONDITIONAL_CHARGING_START state. This option will provide bike charging during a predetermined time period that must be specified at launch time between the following values: 4H30, 2H30, 1H30, 0H30, 0H05. As in normal charging, the charger may not be available when desired. If this is the case the Slave Controller will enter the UNCONDITIONAL_CHARGING_WAIT state from where it will try to gain the resource as explained above. This state is also signaled using the docking point LEDS.

If the charger is available, it will redirect the execution to one of the following states:

UNCONDITIONAL_CHARGING_4H5,
UNCONDITIONAL_CHARGING_2H5,
UNCONDITIONAL_CHARGING_1H5,
UNCONDITIONAL_CHARGING_0H5,
UNCONDITIONAL_CHARGING_5MIN

As time passes, the state machine will transit these states in cascade in accordance with the remaining charge time, and LEDS will flash to signal unconditional charging.

During unconditional charging only two conditions may stop the charging automatically. One is the charge over current condition, and the second is 42V tension lose. In these events the Slave Controller will behave as in "normal charging". In all other situations the unconditional charging will stop at the end of the selected time span. Once the timeout is exhausted, the FSK link is evaluated, and if the bike is associated the Slave Controller will call the "normal charging" algorithm. If the association is lost, then the control will be transferred back to the Bike acceptance module, more specifically to the ERROR_ASSOC_42V state, where te 42V are removed and the association may be resumed at 0V on the power line. As in the case of normal charging, the unconditional charging operation may be interrupted by the peer Slave Controller if it requires the charger resource with higher priority. In that case the Slave controller will jump to UNCONDITIONAL_CHARGING_WAIT until the interrupted operation can be resumed.

9.4 BIKE RELEASE MODULE

This module has two different algorithms to perform the bike release. The first can be called "unconditional undock" and the second "undock with error control".

The “unconditional undock” is the only algorithm Slave Controller may invoke as a result of an automatic off-line decision, and will also be triggered upon reception of the Service Undock instruction from Host. It starts at MAKE_UNDOCK_DISASSOC state, since to release the bike the first required step is to disassociate the bike in compliance with the KSP protocol, that is to lose the association condition as a consequence of having closed the FSK link, not because any communication error condition. This is important for the bike to know a ride has started.

The second step is to disconnect any tension applied to the bike through its axis poles, and only after this, to proceed to activate the solenoids to physically release the bike. In some cases like the Bike Tester tool, no more commands can be sent to the Slave Controller after power has been removed.

Therefore the procedure must continue jumping to MAKE_UNDOCK_USERS state to allow the user time enough to withdraw the bike from the docking point (this time is configurable through P13.UNDOCK_TMO). If the user effectively withdraw the bike within the specified time slot, the program flow will jump to the MAKE_UNDOCK_DONE to signal the release success, and then will wait for 5 seconds before transferring the control back to the acceptance module launching the “algorithm to accept newly arrived bikes”. This guard time is to prevent an undesired acceptance due to the erratic movements the bike may perform during withdrawal. If the bike stays in the docking point after the withdrawal timeout is exhausted, then the control will be transferred to the acceptance module by a direct jump to the MAKE_DOCK state, and will set the “error_not_bike” flag.

The “undock with error control” algorithm can only be triggered as a result of the Host issued Make Undock instruction. In fact the instruction will only be accepted if both docking sensors report bike axis presence. This is to avoid actuating the solenoids in a position that may prevent the solenoid mechanics to move freely. The instruction is rejected by its response accompanied with the DIN Status information so that the user can be instructed to put the bike in place.

If bike is properly positioned, the algorithm will also start at MAKE_UNDOCK_DISASSOC state, and will perform disassociation, voltage disconnection and solenoid activation in the same way as the previous algorithm. After the solenoids are instructed to open, a retry is implemented in case of error, in a similar way as in the close operation. If at the end of retrials any solenoid sensor do not reply the expected opened condition it will jump to ERROR_UNDOCK_2 to signal the sensors error and then will transfer the control to the Bike acceptance module selecting the “algorithm to accept already present bikes” aborting the undock operation. As a result of the control transfer, the bike will be immediately docked and re-associated. This way the trip will be ended and the user will be notified.

If no sensors errors are detected, the Slave Controller will jump to MAKE_UNDOCK_USERS state to enable a time window to withdraw the bike as in the above algorithm. Equally, if withdrawn on time the state machine will move to the MAKE_UNDOCK_DONE state and from there will transfer the control to the Bike acceptance module launching the “algorithm to accept newly arrived bikes”.

If withdraw timeout expires the Slave Controller will move to the ERROR_UNDOCK_EXPIRED state, and immediately after the program flow will be redirected to the Bike acceptance module selecting the "algorithm to accept already present bikes".

9.5 SLAVE CONTROLLER STATES

All the above mentioned states are the only ones that the Slave Controller will report via events. To perform all operations some intermediate states are required, and may be reported to Host if the GetStatus instruction is issued. The Host may identify them but can discard them as all remarkable status information is reported by the set of states raised as events. Furthermore, none of them is stable in the sense all intermediate states are short in duration.

Here is the list of all states and its numerical code in hexadecimal. In red are the states reported as events.

Table 11: Slave Controller states

Value	**Name**	**Description**
0xC0	SLAVE_START	
0x10	SLAVE_IDLE	The Docking Point is idle, with no bike docked
0x20	SLAVE_MAKE_DOCK	A bike has been detected into the docking point, and lock mechanism is getting started
0x21	SLAVE_MAKE_DOCK_1	
0x5A	SLAVE_FSK_0V_PROBE	
0x50	SLAVE_FSK_12V_PROBE	
0x51	SLAVE_FSK_42V_PROBE	
0x28	SLAVE_BIKE_DOCKED	Both mechanical and communication docking have been successful. A bike is docked.
0x30	SLAVE_START_ CHARGING	Start the charging the bicycle's battery. That is supposed to happen automatically, but the Host may also trigger it.
0x32	SLAVE_START_ CHARGING_WAIT	
0x38	SLAVE_BIKE_CHARGING	Bicycle's battery is charging

0x3A	SLAVE_STOP_CHARGING	Stop battery charging. GWA charger automatically stops when the battery is full, but the Host may stop charging at any moment, usually before undocking
0x3F	SLAVE_CHARGING_ STOPPED	42V have been switched off, either by the Host or automatically.
0x70	SLAVE_UNCONDITIONAL_ CHARGING_START	
0x76	SLAVE_UNCONDITIONAL_ CHARGING_WAIT	
0x71	SLAVE_UNCONDITIONAL_ CHARGING_4H5	The Host Application sets this state when 4.5h of Unconditional Charging are started. Slave Controller will respond with state 0x70, then move to state 0x72 after 2 hours.
0x72	SLAVE_UNCONDITIONAL_ CHARGING_2H5	The Host Application sets this state when 2.5h of Unconditional Charging are started. Slave Controller will respond with state 0x70, then move to state 0x73 after 1 hour.
0x73	SLAVE_UNCONDITIONAL_ CHARGING_1H5	The Host Application sets this state when 1.5h of Unconditional Charging are started. Slave Controller will respond with state 0x70, then move to state 0x74 after 1 hour.
0x74	SLAVE_UNCONDITIONAL_ CHARGING_0H5	The Host Application sets this state when 0.5h of Unconditional Charging are started. Slave Controller will respond with state 0x70, then move to state 0x3A after half an hour.
0x39	SLAVE_UNCONDITIONAL_ CHARGING_5MIN	42V are applied, because bicycle was docked unpowered. BootUp stays for 5 minutes, then transitions to state 0x38.
0x29	SLAVE_MAKE_UNDOCK_D ISASSOC	
0x2A	SLAVE_MAKE_UNDOCK_1	Stars the bike undocking process. As a precondition both position sensors has to report “detected”.
0x2B	SLAVE_MAKE_UNDOCK_2	
0x2C	SLAVE_MAKE_UNDOCK_ USERS	After the KSP link has been cleared, the bike is mechanically released. If Host sets this state, SLAVE_BIKE_DISASSOCIATE will come first.
0x2D	SLAVE_MAKE_UNDOCK_ DONE	Solenoids have been disengaged. Bike can be retrieved by the user.
0x2E	SLAVE_MAKE_UNDOCK_O	Stars the bike undocking process. As a precondition at

	NE_SENSOR	least one position sensor has to report "detected".
0x44	SLAVE_HEATING_START	Transitional state used to trigger heaters from the Host, when bike is docked
0x45	SLAVE_HEATING_STOP	Transitional state used to stop heaters from the Host, when bike is docked. If heating started automatically, it is held off for 30 seconds.
0xCB	SLAVE_TEST_INI	
0xCC	SLAVE_ONLINE_TEST	Allows online control on Digital Outputs (mainly, the solenoids)
0xCE	SLAVE_HW_TEST	HW test: Continuous, Repetitive Operation (CRO), intended for EMC or wear-out tests.
0xCF	SLAVE_TEST_FIN	
0xD0	SLAVE_BOOTUP_INVALID _RTC	Slave Controller starts from a hardware reset. Real-Time Clock is not valid
0xD1	SLAVE_VALID_RTC	Real-Time Clock has been updated from the Host and is now valid
0xD2	SLAVE_SOL_VANDAL	Anti-vandal solenoid feature triggered.
0xDA	SLAVE_SERVICE_MODE_ UNDOCK	Docking solenoids are unconditionally released
0xE1	SLAVE_ERROR_ASSOC	Communication between Slave and BikePCB has been lost. Slave will try to recover
0xE2	SLAVE_ERROR_ CHARGING	The charger voltage has been unexpectedly lost while in NOTASSOC_42V or ASSOC_42V states.
0xE3	SLAVE_ ERROR_DOCKING	Docking could not be completed properly (i.e, sensors are not reporting proper solenoid activation). Docking will be resumed as soon as sensor conditions are correct.
0xE4	SLAVE_ERROR_ UNDOCK_2	Undocking could not be completed properly (i.e, sensors are not reporting proper solenoid retraction, or bike is being pulled while undocking). Undocking will be resumed as soon as sensor conditions are correct.
0xE5	SLAVE_ ERROR_UNDOCK_EXPIRE	Undocking was successful, but the user did not retrieve bike. After some time (see CFG_Param 0x13, UNDOCK_TIMEOUT), docking will start again.

0xE6	SLAVE_ ERROR_ASSOC_42V	Communication between Slave and BikePCB was lost during charging. That could be due to low impedance of the battery during recharge process, so charging will stop, to try to recover communication.
0xE7	SLAVE_ FSK_PROBE_ERROR	
0xE9	SLAVE_ ERROR_NOT_BIKE	Something was docked that has not been recognized as a bike (short-circuit on powerline, or no communication at any power applied). Undocking will be performed next.
0xEA	SLAVE_WHOAMI	This state value is only used in the System Log to store records concerning the 42V arbitration algorithm.
0xEB	SLAVE_CHARGING_OVER CURRENT	Charge current is too high, after charging has been successfully started (so it's been cut-off).
0xEC	SLAVE_ERROR_LEFT_ SOLENOID	Solenoid current is too high, either at self-test or during dock/undock operations ("left" and "right" may be reversed, though).
0xED	SLAVE_ERROR_RIGHT_ SOLENOID	Solenoid current is too high, either at self-test or during dock/undock operations ("left" and "right" may be reversed, though).
0xEF	SLAVE_ERROR_ SOLENOID_SENSORS	
0xF0	SLAVE_ERROR_MACHINE	

9.6 SLAVE CONTROLLER RESOURCE ARBITRATION

A given Slave Controller may need the charger resource while the peer Slave Controller is already using it. In that case a priority management is needed to determine which one is going to gain the resource. But it may also happen that both Slave Controllers decide to gain the charger at the same time. In that case a collision resolution algorithm is needed.

9.6.1 SLAVE CONTROLLER ARBITRATION 42V

To manage both the priority policy and the collision resolution, the Slave Controller's 42V charger arbitration managing consist of two different subsystems:

Collision arbitration: At a hardware level there are two signals at the basis of this arbitration module. The first is the charger Enable Signal, that is the charger input that allows switching on and off the charger. The second is the charger output, whose tension will rise near 42V when the enable input is active. In a peer-to-peer configuration Slave#1 enable output is connected directly to the charger Enable Signal, whereas Slave#2 enable output is connected to the Slave#2 enable input in a daisy-chain way. The charger output is connected directly to both Slave Controllers.

The core of the arbitration process is to ensure only one Slave Controller will connect its 42V relay at a time. To achieve that goal a Slave Controller acting as a "master" will be in charge of controlling the access to the charger, and a Slave Controller acting as a "slave" will have to negotiate with the master in order to access the charger. Slave Controller #1 will be the arbitration "*master*", and Slave Controller #2 will be the arbitration "*slave*".

Between the master and slave there is a double communication link. One is implemented through the Enable Signal, and travels in the direction from #2 to #1. The other one is a bidirectional CAN communication. This link shares the CAN arbitration vectors with the priority arbitration algorithm.

Using these communication links, an algorithm called "Who am I" will determine whether a Slave Controller is #1 or #2. This algorithm works when both Slave Controllers get up from reset at the same time, but also when one of the boards gets up from reset while the other one is in operation and already knows who she is. To ensure this is possible, the "who am I" algorithm is designed not to interfere the normal operation of an already operating master or slave.

After the reset, the "Who am I" algorithm will consider the Slave Controller as Undefined, meaning the FW does not know to be #1 or #2.

While "Undefined", the board will try to determine its position (#1 or #2) as it will be explained later. In this state she can activate the enable signal, if needed to determine its number, but will NEVER activate the 42V relay.

If the board knows to be Slave Controller #1, then as it is the master it will consider to have the right to activate the relay whenever the charger is not in use. The "in use" flag is perfectly known by the master as it is his duty to keep track of it.

If the board knows to be Slave Controller #2, then prior to activate the relay, she has to ask for permission to the master. To do that, it will pulse the Enable signal for 200ms, and wait for the new CAN arbitration vector. The reception of this vector means “you can activate your relay, and if within a 1 second window you activate the enable again, I will forward this signal to the charger, and I will consider you are using the charger until you lower your enable and a minimum time has expired to ensure you have deactivated your relay”. The Slave Controller #2 will only activate its relay upon permission reception.

After having used the charger, that is, after lowering the enable signal, slaves will not raise the enable within a guard time window of 2 seconds. This is to ensure that enable wave form described in the Slave #2 negotiation will only appear in that case.

This is how the “Who am I” algorithm works:
To avoid risks, the Slave Controller number will be determined by physical ways after every reset, and will not be taken from any configuration register. This will avoid problems when, after servicing a docking point, both Slave Controller boards are swapped, or a Slave Controller from one DP has been interchanged by another board of a different DP.

The “who am I” algorithm is divided into two parts. The first part is always active. We can call it the “Permanent Part”. The second part will only execute when the Slave Controller does not know who she is. We can call it the “Undefined Number Part”

The “Permanent Part” behaves as follows:

> Every time the Slave Controller receives an enable pulse of 1.5 seconds (strictly speaking 1sec. < T < 2 sec.) it will transmit by CAN an special arbitration vector carrying the meaning “I’m #1”.
>
> Also every time The Slave Controller receives an special arbitration vector meaning “Transmit a 1.5 sec. enable pulse if you are #2”, it will do so.
>
> If receives “I’m #1” when being #1 will immediately disconnect the relay and set the slave number to undefined. This is to be protected against a multiple master scenario.
>
> If receives an enable pulse of 1.5 seconds when being #2 will immediately set the slave number to undefined. In case the Slave Controller is using the charger, any received enable pulse will cause immediate relay disconnection. This is to be protected against a multiple slave scenario.

The “Undefined Number Part”:

While a Slave controller does not know its number, it will constantly pulse its enable output for 1.5 seconds active and for 2 seconds inactive. At every pulse activation it will send by CAN a special arbitration vector to the peer slave meaning: “Transmit a 1.5 sec. enable pulse if you are #2”
During emitting the enable pulse by its output, and the CAN frame through its CAN transmitter, the Slave controller will also be monitoring its enable input and its CAN reception.

If it receives an enable pulse of 1.5 sec, it will consider to be Slave #1 and will exit the undefined number state. (Note that a spurious in enable input will have no effect).

If it receives a “I’m #1” CAN arbitration vector then it will consider to be Slave #2 and will exit the undefined number state.

If receiving inconsistent inputs, it will remain in undefined.

Let’s see some Single and Double Docking point considerations:
As remarked before, for the proposed arbitration procedure to avoid relay overlapping, it is crucial to have only a master, and very important: there must be only one master. The above described “Who am I” algorithm ensures that condition but requires always a double D.P. to work. This is so because a Slave Controller will always become a master upon slave intervention. The Slave Controller does not have any means to auto consider herself as a master.

This has one consequence in the case in a double D.P. in which suddenly Slave#2 fails. The Slave Controller #1 of that DP will still work, but if the power supply is disconnected and re-applied it won’t, since nobody will be there to tell that slave she is the master. The D.P. will need to be serviced.

In the case a 2015 Slave Controller is installed in a single DP, to make it a master will require proper stimulation of its enable input. This pin, now in open circuit, will need to be connected to the enable input. This may be done with a special three connector cable. The male connector is plugged to the power supply, the other two female connectors (white and black) are properly connected to the input and output of the slave controller. This cable is a hard-wired way to allow a Slave Controller to self become a master. The service personnel will be responsible not to install this cable in a double D.P. In case of installing the cable in the position #2 of a double D.P., the double master protection system will trigger and no relay overlapping will occur, although the DP will need to be serviced to be able to charge the bikes.

And also some considerations related to the Slave Controller main machine behavior:

While the “Who am I” algorithm considers the slave number is undefined, all charger operations initiated by the main state machine will receive an “aborted” as response. As a consequence the Slave Controller will enter the wait states, as if the charger would be in use by the peer slave. At start up, if a bike is present it will go to regular or unconditional charge. By the time it happens, the “Who am I” algorithm will have already had the time to determine the slave Controller position number.

Priority management: During the Slave Controller operation, the charger may be required to perform three different tasks: to identify a depleted bike applying 42V, to unconditionally charge a bike, or to performing a regular charge operation. If one Slave Controller (say Slave-A) is performing one of these tasks and the peer Slave Controller (say Slave-B) need to start its own task, to arbitrate the shared resource, each task is given a priority level as follows:

Identification of a depleted bike: High Priority.
Unconditional Charge Operation: Medium Priority.
Regular Charging Operation: Low Priority.

According to the hardware based arbitration, for Slave-B to gain the charger, Slave-A must stop using the resource. To induce this condition, Slave-B will send a request to its peer slave indicating the priority of the task it intends to perform. At reception, Slave-A will compare the informed priority with the priority level of its own task, and will determine whether to drop or not the charger. If the charger is freed within a certain time slot, it will be perceived by Slave-B either as a deactivation of its enable input signal, or by the fall of the charger output tension. Under these conditions Slave-B can use the charger. If not, it means the use of the charger has been denied.

When a Slave Controller that is performing a task of medium or low priority has to yield the charger to its peer Slave Controller, it enters a wait state that will be CHARGING_WAIT or UNCONDITIONAL_CHARGING_WAIT accordingly. In these states, the Slave Controller will monitor whether peer Slave has freed the charger in order to resume the unconditional or regular charging operations.

9.6.2 SLAVE CONTROLLER ARBITRATION 12V

To reduce the current consumption at 12V, the Slave Controller will never operate both solenoids simultaneously. Dock and undock operations activate the solenoids in sequence. If any dock or undock operation has to be performed while the heater is on (heating also uses solenoid coils), the heating will be suspended during the operation and resumed immediately after.

Apart from the care any individual Slave Controller may take to minimize current consumption, when two Slave Controllers are sharing the same power source and are linked via CAN, the Slave Controllers will exchange arbitration frames to divide time in slots of about 10 seconds in which “high current” operations are allowed. The heater operation will only be possible during the allowed slot for that Slave Controller. So the heating will work alternatively in both slaves. If any Dock/Undock operation is to be performed, it will be executed even out of the allowed slot, but at the same time an arbitration will take place to request immediate permission to consume, and that will have the immediate effect of stop the peer slave heater if on.

9.6.3 PEER-TO-PEER AND SINGLE CONFIGURATION

Collision and arbitration makes full sense when two Slave Controllers are connected peer-to-peer sharing one single charger resource. But the single configuration, that is one Slave Controller directly connected to one charger may also be a possible topology. The described arbitration procedure will also work for this setup, and no change in configuration will be required.

9.6.4 HEATER OPERATION

Heating will be automatically switched on when ambient temperature is detected to fall below LOW_TEMP_TRIGGER (CFG_Param 0x10). Heating consists of applying current to the locking solenoids, in the sense that keeps the bike in its present locked/unlocked condition, during HEATER_SEMIPERIOD (*CFG_Param 0x0E*).

Heating current wil be applied consecutively to each solenoid, and never simultaneously, to avoid excess current consumption. That means that each heating element is powered with a duty cycle of 50%.

Heating will only be automatically switched off when temperature goes LOW_TEMP_HYSTERESIS (*CFG_Param 0x11*) degrees above the tipping point (LOW_TEMP_TRIGGER). That is done to avoid turning heating on and off too fast.

Heating may also be turned on or off from the Host:

- Slave Controller must be set to State 0x44 (SLAVE_START_HEATING) to trigger heating on.
- Heating operation is compatible with Dock/Undock operations. Heating will be temporarily suspended to dock/undock
- Slave Status will not change, but Heater Status may be seen through Digital Input Status (most significant bit is set when heating is on).

9.7 ONLINE BEHAVIOR

The Slave Controller is connected to a CAN-bus network.

For maintenance purposes the Slave Controller can be connected to the CAN-bus master controller, so that the whole Docking Station can be monitored from a remote Host.

When a bicycle is docked, instructions from the bicycle's Tablet PC may also interact with the Slave Controller. This allows:

1.- Manage the undocking sequence.

2.- Retrieve log information and statistics.

3.- Perform FW upgrades and other maintenance tasks.

9.8 ENERGY MANAGEMENT

Since the Slave Controller is a node in a CAN-bus network, it must be active all the time. However, its current consumption is designed to be as low as possible.

Bicycle charging will be managed from the Slave Controller. A digital input nENABLE signal will be sent to the GWA Charger, or to the peer Slave Controller in order to start battery charging. That digital input will be active after the power relay has been engaged, and will be left inactive prior to disengaging the power relay. That will reduce stress and transient currents on the electronics system.

9.9 EVENT LOGGING

Docking and charging activity of the Slave Controller will be kept in the form of an event log, in the Slave Controller's EEPROM. A circular buffer of 127 event logs will be available.

Date and time information must be provided to each Slave Controller, so that accurate logging exists. Even though no Real-Time Clock exists on the Slave Controller, online instructions (Set Date and Time) allow any of the Tablet PC's that are docked, to periodically broadcast updated date and time information to all Slave Controllers simultaneously.

9.9.1 DATE TIMER FORMAT

Date and Time will be handled as a 4-byte value, with the following bitmap:

AAAA MMMM dddd dHHH HHmm mmmm ssss snnn

Where:

AAAA	Lower nibble of the year, after encoding it in binary format (example: year 2013 corresponds to 0x07DD, so we encode the last 0xD).
MMMM	Encoding for the Month, from 0x1 to 0xC (1 to 12).
dddd d	Encoding for the Day of the month, from 0x01 to 0x1F (1 to 31).
HHH HH	Encoding for the Hour of the day, from 0x00 to 0x17 (00 to 23).
mm mmmm	Encoding for the Minutes, from 0x00 to 0x3B (00 to 59).
ssss s	Encoding for 2xSeconds, from 0x00 to 0x1D (00 to 29). *Note that the least significant bit for the seconds is missing, and assumed to be zero!!* We can only encode even values for the seconds.
nnn	Encoding for the Day of the Week (0x1 for Monday, 0x0 or 0x7 for Sunday).

The default year code will be encoded in EEPROM, in order to avoid confusion.

Time managed by Slave Controller is meant to be UTC. That means, all applications dealing with time or logs must convert appropriately to local time.

9.9.2 EVENT LOG FORMAT

Event log records have a total length of 8 bytes:

- Slave State: 1 byte indicates one of the Slave Controller's states shown on Table 10.
- Date and Time: 3 most-significant bytes of the current Date and Time (see format in Subsection 9.4.1., above).
- The last 4 bytes may vary its meaning, depending on the Slave State. Normally, they correspond to:
- Heater Status
  - Digital Input Status
  - ADC Channel: one ADC channel is reported, depending on the status recorded.
  - ADC Reading: corresponding to the ADC channel reported.
- In case of Dock or Undock logs, the last two bytes correspond to the two least-significant bytes of the BikePCB's EUI-64 (instead of an ADC reading).
- In case of State 0xD1 (SLAVE_RTC_SYNC_OK), the last four bytes correspond to the date and time value that has been replaced. That allows converting logs recorded with a previous (usually not realistic) time stamp.

9.9.3 STATISTICAL INFORMATION

Based on the logs recorded, counters for the different events are kept. This is a fast way to assess the performance of a docking module, without the needs to retrieve all the log database.

- Number of successful dockings
- Number of unsuccessful dockings
- Number of successful undockings
- Number of unsuccessful undockings
- Number of broadcast errors received: corresponding to [App_ID, Node_ID] different from [0x02, 0x00]

If any of the counters reaches its top value (0xFFFF), all counters will stop until reset by the user.

9.10 SLAVE CONTROLLER CONFIGURATION

The configuration of the Slave Controller is stored in EEPROM memory, which gives it around 10,000 read/write cycles. This has to be taken into account when managing the configuration parameters. Default configuration values should be the right ones for Slave Controller operation, anyway.

9.10.1 PARAMETER ARRAY

The Slave Controller is configured from the following configuration parameters that are sent using command WriteCFG_Byte and can also be read using command ReadCFG_Byte. All the device's parameters are listed below:

Table 11: List of parameters of the Slave Controller

Num	Description	Default Value
0x01	CFG_UART_HOST – Determines the baud rate of communication with the Servicing Host. 0x20: 9600bps 0x21: 19600bps	0x21
0x02	ICHAR_TMO_HOST – Determines the maximum standby time between two consecutive characters of communication with the Host to be interpreted for the same command frame. Allows any value between 0x04 and 0xFF, and the standby time of the value of this byte is multiplied by 5 ms.	0x0A
0x03 to 0x05	RFU - Reserved for future use.	
0x06	CHARGER_VOLTAGE_THRESHOLD Configures the voltage level of the line voltage comparator. Any tension above this value will be considered a charging voltage.	0x60
0x07	CFG_DIN_ACTIVE_STATE – Specifies whether Digital Inputs are Active-High or Active-Low A bitmap tells which digital inputs are active-high (bit to 1), which are active-low (bit to 0).	0x0C
0x08	CFG_SLAVE_MODES – Allows to enable or disable certain functionalities Certain functionalities can be enabled: - Bit 0 (LSB): RFU - Bit 1: RFU	0x43

	- Bit 2: RFU - Bit 3: RFU - Bit 4: RFU - Bit 5: if set, Offline Heating is enabled while bike is docked. - Bit 6: Determines 2-to-1 configuration mode. - Bit 7: Determines 2-to-1 configuration mode.	
0x09	CFG_ECHO It defines the events that are reported to the Host. •Bit 0:If set, service UART is enabled, else CAN bus is enabled. •Bit 1: If set, events are reported to CAN or UART Host by default (e.g, at startup), else to BikePCB. •Bit 2: If set, event is sent to default Host at startup. •Bit 3: If set, KSP Associate Request is sent to default Host at startup. •Bit 4: If set, FSK beacons are echoed to Host •Bit 5: Reserved. •Bit 6: If set, event is echoed during docking or charging status. Bit 7: If set,e vent provides additional information.	0x01
0x0A	DIN_VALIDATION_TMR – Configures the maximum time elapse between docking sensors right/left activation for the Slave Controller to consider a bike has been docked. (units: 5 ms)	0x64
0x0B	CHARGING_DELAY – Configure the delay between docking and automatically start charging (units: seconds). If value is 0x00 or 0xFF, no automatic charging at all. Actual delay may increase up to 3.2 seconds, as a function of EUI-64 of each Slave Controller. That flattens out peak inrush current, in case of a mains 220V dip.	0x0A
0x0C	CHARGE_MAX_RISE_TIME – Time interval before which 42V must be detected on the Slave Controller, at 42V start-up (units: 5 ms)	0xA0
0x0D	CHARGE_MAX_FALL_TIME – Time interval before which 42V must be detected on the Slave Controller, at 42V start-up (units: 5 ms)	0x60
0x0E	HEATER_SEMIPERIOD – Heater pulse activation time for each solenoids, after this span the other solenoid will be used to heat-up the docking point. (units: 5 milliseconds).	0x64
0x0F	SOLENOID_TIME_A – Docking solenoid activation time (units: 5 milliseconds). Solenoid activation for the bicycle dock is limited by a timeout value, configured here (units: 5 milliseconds). If all hardware is correct, activation terminates as soon as the digital input detects	0x32

	the end of the solenoid movement.	
0x10	LOW_TEMP_TRIGGER – Temperature sensor reading, below which solenoid heating is enabled. Heating mode will be activated as soon as temperature goes below the temperature stated here (4ºC by default).	0x36
0x11	LOW_TEMP_HYSTERESIS – Temperature sensor reading to add to LOW_TEMP_TRIGGER, before solenoid heating is disabled. Heating mode will be deactivated as soon as temperature goes above (LOW_TEMP_TRIGGER + LOW_TEMP_HYSTERESIS), in order to avoid switching on and off continuously (Default hysteresis is 3ºC, yielding a switch-off temperature of 4+3=7ºC).	0x03
0x12	RFU	0xD3
0x13	UNDOCK_TIMEOUT – After the bicycle has been released from the solenoids, the user has a few seconds allowed withdrawing it. If the value here configured (units: 1 sec) expires, the bike will still be detected, and the solenoids will lock it again.	0x14
0x14	MIN_SOLENOID_CURRENT – Solenoid detection considers Open Circuit (no solenoid connected) if current is below this limit at Self-Test. Default 100mA equals to 20 counts.	0x14
0x15	MAX_SOLENOID_CURRENT – Solenoid detection considers Short Circuit (solenoid failure) if current is above this limit at Self-Test. Default 1A equals to 200 counts.	0xF6
0x16	RFU.	0x04
0x17	RFU	0xF6
0x18	MAX_12V_CURRENT – Maximum current limit when measuring powerline impedance with 12V supply.	0x2D
0x19	Reserved	
0x1A	MIN_42V_CURRENT – Charging is considered to have ended if current is below this limit. A zero indicates charge will not stop due to low current.	0x00

0x1B	MAX_42V_CURRENT – Protection against short-circuits or overload, for 42V supply. Default 4.4A equals 220 counts.	0xF0
0x1C	RFU - Reserved for future use.	0xE6
0x1D	FSK_PROBE_ERR_RETRY_DELAY Delay between every 3 FSK_PROBE cycles in FSK_PROBE_ERROR in minutes (exactly 61.4sec). Minimum value 1 min.	0x1E
0x1E	RFU - Reserved for future use.	0x01
0x1F to 0x24	RFU - Reserved for future use.	

9.10.2 CONFIGURATION OF COMMUNICATION WITH THE HOST

The Slave Controller features a Service UART (TTL levels) where a USB Virtual COM port adapter may be connected. That allows a direct connection to the Slave Controller from a laptop PC, thus bypassing both the FSK link to BikePCB and the CAN-bus network. The behavior of this UART port is defined from parameters 0x01-CFG_UART_HOST, 0x02- ICHAR_TMO_HOST and 0x09- CFG_ECHO of the configuration:

CFG_ECHO parameter allows us to define a start-up event, either to the Service Host through Virtual COM, or to the Master Controller through the CAN bus, or even to BikePCB and the TabletPC (FSK).

9.10.3 SELF-TEST AND PROBING

Self-Test is performed at start-up, or more specifically, after every SLAVE_START state. Self-Test ir responsible for detecting the solenoids.Depending on the status reported by the sensors, current through the solenoids is injected in one sense or the other, to make sure that they are not moving. Open connections or short circuits are reported as errors (right or left accordingly).

APPENDIX A. COMMUNICATIONS EXAMPLES

A.1 DIRECT COMMUNICATION (HOST – BIKEPCB)

The simplest scenario for KSP occurs between the Host and the BikePCB.

In the following example, the Host asks for the FW version to the BikePCB (KSP Address: 0x00.00) directly, through KSP. Communication will be as follows:

Table 16: KSP Direct Communication sequence

Host	AppID, NodeID	KSP OpC	Label	Instruction /*HSK Type*	BikePCB
Tx >>	0x0000	0xF8	0x01	0x02	>> Rx
Rx <<	0x0000	*0xF0*	0x01	*0x0300*	<< Tx
Rx <<	0x0000	0xF8	0x01	0x827D3A	<< Tx
Tx >>	0x0000	*0xF0*	0x01	*0x0300*	>> Rx

The KSP stack on the Host side must be prepared to handle the following scenarios:

- If no Handshake is received, the sender will retry up to three times, approximately every 1 second (exact time may be configurable on the Host side).
- If no Handshake is received after three times, an error must be reported to the upper application layers.
- If End-Point ACK is received (0x0300, as shown above), transmission has been successfully completed.
- If End-Point NACK (0x04FF) is received, KSP transmission must be cancelled, and an error reported to upper application layers.
- If End-Point WAIT (0x0420 or similar) is received, the Host must wait for the specified time in tenths of second (in the example, 3.2 seconds), then restart transmission.

Symmetrically, the End Node will resend Response up to three times more, if no ACK is received from the Host.

A.2 RELAYED COMMUNICATION (HOST – BIKEPCB – SLAVE CONTROLLER)

Whenever a Mid-Point (BikePCB) comes into play, the Host stops resending as soon as the Mid-Point ACK is received. The Mid-Point is then responsible for routing the KSP data frame to the End-Point:

Table 17: KSP Relayed Communication sequence

Host	**[KSP_ID][OpC] [Label][Instr.]**	**BikePCB**	**[KSP_ID][OpC] [Label][Instr.]**	**Slave Controller**
Tx >>	[0x0200][0xF8] [0x01][0x02]	>> Rx		
Rx <<	[0x0000][*0xF0*] [0x01][*0x0100*]	<< Tx		
		Tx >>	[0x0200][0xF8] [0x01][0x02]	>> Rx
		Rx <<	[0x0200][*0xF0*] [0x01][*0x0300*]	<< Tx
Rx <<	[0x0200][*0xF0*] [0x01][*0x0300*]	<< Tx		
		Rx <<	[0x0200][0xF8] [0x01][0x827D3A]	<< Tx
		Tx >>	[0x0200][*0xF0*] [0x01][*0x0100*]	>> Rx
Rx <<	[0x0200][0xF8] [0x01][0x827D3A]	<< Tx		
Tx >>	[0x0200][*0xF0*] [0x01][*0x0300*]	>> Rx		

Note that the Host will stop resending the first instruction as soon as Mid-Point ACK is received.

In general, the Mid-Point will route either End-Point ACK or End-Point NACK to the Host. End-Point ACK does not imply any further action by the Host, whereas End-Point NACK should be reported as an error to the upper application layers.

If no handshake is generated by the End-Point even after a total of 4 retries by the Mid-Point, the Mid-Point will generate an End-Point NACK towards the Host, so that the error condition can be reported to the upper application layers.

APPENDIX B. APPLICATION INFORMATION

B.1 HOW TO CONNECT THE CAN BUS

Even if no Master Controller is installed, the CAN bus can be used as a backchannel. That is, Slave Controllers will broadcast some critical information (essentially, error conditions), which will be logged into all the Slave Controllers that share the CAN bus.

That allows detecting error conditions on any Slave Controller of the Docking Station, as soon as one bike is docked at any docking point and event logs from that particular docking point are retrieved. Also, date and time information can be regularly broadcast from the Tablet PC, to make up for the absence of Real-Time Clock on the Slave Controller (which requires that information for a correct event logging).

In order to connect the CAN bus, the following must be done:

1. Use connectors J6 (Input, White) and J5 (Output, Black).
2. Power the CAN bus: The CAN Bus is powered with the 12V provided from the power supply. It's common for all CAN net.
3. Terminate the CAN bus: The Salve Controller build-in a terminate resistor (150 ohm). To remove it, take out R67 (0 ohm).

Broadcast messages can be sent from the Tablet PC, and every Slave Controller will collect all relevant broadcast messages sent by any Slave Controller. That provides some amount of information, even if no Master Controller is present.

B.2 LOCK OPERATION

The AXA Lock can be operated from BikePCB. In order to do so, we must know how to connect it and how to operate it.

B.2.1 CONNECTION BETWEEN BIKEPCB AND AXA LOCK

Connection must be as follows (see also Section 3.5):

Table 18;: AXA Lock connection

AXA Lock		**BikePCB**	
Pin Description	**Number**	**Number**	**Pin Description (J4)**
8-12V	1	1	Lock Power
Input A	2	2	Lock - Control Signal A
GND	3	5	Lock Detect - GND
Input B	4	3	Lock - Control Signal B
Lock Status	5	4	Lock Detect - Digital Input

Please notice:

Connector to be used at cable's end is: Molex 50-37-3053.

B.2.2 OPERATION AND ERROR HANDLING

1. We must first deploy the kickstand if we want to lock the bike: locking is only allowed from the STANDING state.
2. Locking and unlocking may only be triggered by the Host Application.
3. After the Host Application triggers MAKE_LOCK state, AXA lock will let its lever loose, so that the user can manually lock the bike. In the meantime, BikePCB is waiting (USER_LOCKS).
4. User operation can be detected by a change in the Digital Input status, from 0x01 (Standing) to 0x03 (Standing and Locked). At 0x03, we getLOCKED.
5. Only the Host Application may take the bike out of LOCKED. Two error conditions may exist:

- Digital Input Status goes back to 0x01 (or 0x00): in that case, we must assume that the lock has been disconnected, or somehow tampered with (ERROR_LOCK_ TAMPER). The Host Application can just wait until the situation reverts by itself, or force UNLOCK.

- A voltage at PWR+ or association to Slave Controller are detected: that would mean that the bike has been lifted from the ground and inserted into a docking point. If a voltage is detected, BikePCB will reset. After Reset, association will be detected and ERROR_DOCKED_WHILE_LOCKED event will be issued. The Host Application should then force UNLOCK or somehow manage the exception.

6. In normal condition, the Host Application will trigger UNLOCK as a consequence of the user having introduced the correct PIN number to unlock the bike. Then, state transition will go to STANDING, which might immediately switch to RUNNING, if the kickstand has been folded before unlocking the bike.

7. Current sensing is enabled, so that if the lock does not activate its motor (maybe because it’s not there) the bike is reverted to its previous stable state (either STANDING or LOCKED). If that happens when trying to lock, STANDING event is issued instead of USER_LOCKS.

LIST OF REVISIONS

Version No.	**Date**	**Description**
Version 0.50	February 28, 2013	First Draft
Version 0.60	March 14, 2013	
Version 0.70	April 2, 2013	Baud rate changed to 19200 for Host USB communication. Tablet PC input changed from nRST to nWAKE. Removed security code to unlock bike .Added two BikePCB . Added logging functionality for Slave Controller.
Version 0.75	April 17, 2013	More ADC Channels available (MAX valuesAdded two Slave Controller instructions)
Version 0.76	April 23, 2013	New ADC conversion factor.
Version 0.80	June 10, 2013	
Version 0.90	August 28, 2013	Added section where self-test and 12V probe are explained. Also, configuration parameters on List of parameters of the BikePCB and List of parameters of the Slave Controller has been updated
Version 0.91	September 5, 2013	Updated pinout description for Slave PCB v1.1.Added descriptions on offline operation and error handling both for BikePCB and Slave Controller.
Version 0.92	September 26, 2013	BikeID changed to *Bike_Serial_Number*
Version 0.93	October 7, 2013	Corrected address ranges for Slave Controller.
Version 0.94	October 23, 2013	BikePCB v1.1 included in **¡Error! No se encuentra el origen de la referencia.**. Modified Bike and Slave state diagrams (including BIKE_SLAVE_UNAVAILABLE).
Version 0.95	October 25, 2013	Added Bike States 0xE6, 0xE7 to BikePCB States
Version 0.96	October 31, 2013	Added BIKE_FSK_PROBE and related syntax for setting.
Version 0.97	November 6, 2013	Added more detailed information about Lock operation. More configuration parameters on List of parameters of the BikePCB.

		Service_Mode has a Time-Out now.
Version 0.98	November 22, 2013	Added Heater functionality. Digital Input Status is showing bitmap assignment .
Version 0.99	December 23, 2013	- SLAVE_START may be set by the Host, Unconditional Charging states defined. Also in BikePCB (BikePCB States) - Unconditional Charging explained. - New instructions added for BikePCB) - New configuration parameters added (List of parameters of the BikePCB, List of parameters of the Slave Controller)
Version 1.00	January 21, 2014	- New state 0x29 (and a few more, for future use) defined for BikePCB, on BikePCB States. - Explained means to wake up . - 4 last bytes of log records . - Some provisional comments updated.
Version 1.01	March 18, 2014	- Specify that heater operation in Offline mode is available when bike is not docked, but left to MA when docked. - Added tolerance to Sensors' Error (Slave CFG Param 0x08)
Version 1.10	May 14, 2014	Slave FW 2.2: Parameter 0x1E, STOP_CHARGING_TMO, introduced New state 0x4A, SLAVE_SERVICE_ UNDOCK introduced.
Version 1.11	May 29, 2014	Sync Timer Value explained.
Version 1.20		
Version 1.21	Apr 15, 2015	
Version 1.22	Apr 23, 2015	Slave CFG P.1B default value changed (0xDC -> 0xF0).
Version 1.23	June 9, 2015	Slave Controller Identification Module description updated.
Version 1.24	June 17, 2015	Slave Controller Identification Module description updated. CFG parameter P.1D and State 0xD2 included.
Version 1.25	July 13, 2015	Bike PCB 42V states review
Version 1.26	Oct 6, 2015	Bike PCB. 0x15-CFG_PAR default value changed
Version 1.27	Dec 7, 2015	Lost Voltage condition detailed in Slave "Charging" states. 42V Arbitration algorithm changed. Bike PCB. 0x14-CFG_PAR default value changed Bike PCB. 0x15-CFG_PAR default value changed
Version 1.28	Dec 17, 2015	Added Log Opcode SLAVE_WHOAMI

Version 1.29	Oct 17, 2016	Slave and Bike states corrected.
Version 1.30	July 25, 2017	SLAVE_MAKE_UNDOCK_ONE_SENSOR instruction added. Lock Supply control instruction added.

NOTES