FLYFISH TECHNOLOGIES

FF32 - USB Interface Chip

Datasheet

Created on: August 3, 2015, 15:03:28

Contents

1. Features 4
1.1 FF3x USB Interface Chips Family 5
1.2 References 5
2. Pin Diagram 6
2.1 Digital Outputs 7
2.2 Digital Inputs 7
2.3 PWM Generator 7
2.4 Analog Inputs 7
2.5 SPI Master Bus 8
2.6 I^2C Master Bus / TWI Master Bus 9
2.7 1-Wire / MicroLAN Master Bus 9
2.8 REF2 pin 9
2.9 CAP pin 9
3. Addressing and Identification 10
3.1 Addressing 10
3.2 Vendor and Product Identification 10
4. Typical application circuits 11
4.1 +5V power supply 11
4.2 +3.3V power supply 12
5. Initialization 12
6. Commands 13
6.1 Get chip information (0x71) 13
6.2 Set address (0x10) 13
6.3 Get address (0x11) 14
6.4 Set vendor (0x12) 14
6.5 Get vendor (0x13) 14
6.6 Set product (0x14) 15
6.7 Get product (0x15) 15
6.8 Set serial number (0x16) 15
6.9 Get serial number (0x17) 16
6.10 Set digital output (0x20) 16
6.11 Set block digital outputs (0x30) 17
6.12 Read digital input (0x21) 17

6.13 Read block digital inputs (0x31) 18
6.14 Set PWM output (0x22) 19
6.15 Get analog voltage (0x23) 19
6.16 Read block analog inputs (0x32) 20
6.17 Configure SPI bus (0x24) 21
6.18 Write to SPI device (0x25) 21
6.19 Read from SPI device (0x26) 22
6.20 Configure I^2C / TWI bus (0x27) 23
6.21 Write to I^2C / TWI device (0x28) 23
6.22 Read from I^2C / TWI device (0x29) 24
6.23 Configure 1-Wire / MicroLAN bus (0x2A) 24
6.24 Reset 1-Wire / MicroLAN devices (0x2D) 25
6.25 Write to 1-Wire / MicroLAN device (0x2B) 25
6.26 Write bit to 1-Wire / MicroLAN bus (0x2E) 26
6.27 Read from 1-Wire / MicroLAN device (0x2C) 26
6.28 Read bit from 1-Wire / MicroLAN bus (0x2F) 27
6.29 Set mode (0x40) 28
6.30 Set default item (0x41) 28
7. Codename :: Emona2000 29

Document Revisions Log:

Revision	Date	Reason for update
0.1	Mar 11, 2014	Initial version
0.2	Apr 8, 2014	Added References links, Fixed small typos
0.3	Apr 13, 2014	Added commands Set block digital outputs and Read block digital inputs
0.4	Aug 25, 2014	Added commands Reset 1-Wire / MicroLAN devices, Write bit to 1-Wire / MicroLAN bus and Read bit from 1-Wire / MicroLAN bus. Updated FF34 summary data, Fixed small typos
0.5	Nov 3, 2014	Added commands Set mode (0x40) and Set default item (0x41)
0.6	Aug 3, 2015	Added command Read block analog inputs (0x32)

1. Features

- 28-pin chip, 300-mil DIP body

- 18 Digital Outputs
- 18 Digital Inputs
- 6 PWM Outputs
- 12 Analog Inputs
- 4 SPI Master Buses
- 9 I^2C Master Buses / 9 TWI Master Buses
- 18 1-Wire Master Buses / 18 MicroLAN Master Buses

- 3.3V or 5V Power Supply
- Hot-Pluggable
- Core current consumption at full operating load <10mA
- High-current Output Pins, rated for 25mA
- Maximum 185mA sourced and sunk by all Output Pins
- Operating temperature range -40°C to +85°C

- USB 2.0 compliant
- Natively supported by various Operating Systems (Linux, Mac OS, Windows, BSD, etc.). No additional USB driver needed.

- Supported up to 127 chips attached to the host
- Programmable circuit's Vendor and Product ID strings, including serial number

- Upgradeable via embedded update feature (bootloader)

1.1 FF3x USB Interface Chips Family

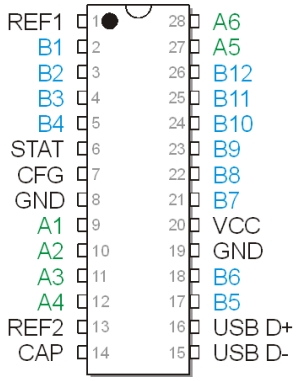
Chip ID	Pin count	Digital Outputs	Digital Inputs	PWM channels	Analog inputs	SPI buses	I^2C / TWI buses	1-Wire /MicroLAN	Status (November 2014)
FF31	14	8	8	4	4	2	4	8	Under investigation
FF32	**28**	**18**	**18**	**6**	**12**	**4**	**9**	**18**	**In production**
FF34	40/44	29	29	8	16	7	14	29	In production

1.2 References

Latest version of this document: http://www.flyfish-tech.com/FF32/FF32_datasheet.pdf

Errata document: http://www.flyfish-tech.com/FF32/FF32_errata.pdf

2. Pin Diagram



Pin	Name	Digital Output	Digital Input	PWM	Analog input	SPI(1)	I²C / TWI(1)	1-Wire /MicroLAN	Description
1	REF1								Reference pin, connect to Vcc via 10k resistor
2	B1	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
3	B2	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
4	B3	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
5	B4	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
6	STAT								Status LED output, 10mA maximum
7	CFG								Reserved, connect to STAT pin
8	GND								Power pin (Ground)
9	A1	Yes	Yes	Yes		Any	Any	Yes	General purpose pin
10	A2	Yes	Yes	Yes		Any	Any	Yes	General purpose pin
11	A3	Yes	Yes	Yes		Any	Any	Yes	General purpose pin
12	A4	Yes	Yes	Yes		Any	Any	Yes	General purpose pin
13	REF2								Connect either to Vcc or GND; *see **2.8 REF2 pin***
14	CAP								Capacitor pin; *see **2.9 CAP pin***
15	USB D-								USB port data signal
16	USB D+								USB port data signal
17	B5	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
18	B6	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
19	GND								Power pin (Ground)
20	VCC								Power pin (+3.3V or +5V)
21	B7	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
22	B8	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
23	B9	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
24	B10	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
25	B11	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
26	B12	Yes	Yes		Yes	Any	Any	Yes	General purpose pin
27	A5	Yes	Yes	Yes		Any	Any	Yes	General purpose pin
28	A6	Yes	Yes	Yes		Any	Any	Yes	General purpose pin

Note 1: The pin can be configured as any master bus' signal

2.1 Digital Outputs

Any digital output can source and sink currents up to 25mA. Maximum current sourced and sunk by all outputs is 185mA.

Voltage levels:

State	Power Supply	
	3.3V	5V
Output low	< 0.4V	< 0.6V
Output high	> 2.6V	> 4.3V

2.2 Digital Inputs

Digital inputs do not contain any pull-up resistors.

Voltage levels:

State	Power Supply	
	3.3V	5V
Input low	< 0.5V	< 0.8V
Input high	> 2.6V	> 4.0V

2.3 PWM Generator

The PWM generator has a resolution of 1% at the frequency of 1kHz.

Any General Purpose Pin from “A block” can be independently set to generate PWM signal.

2.4 Analog Inputs

Analog to digital conversion is performed at the 10-bit resolution.

Input voltages must be DC and within the chip operating voltages (between GND and Vcc inclusive).

DO NOT try to measure mains voltage!

Special attention must be paid to the circuit power supply voltage. By the specification, USB port 5V voltage has +-5% tolerance. Be aware that the deviation of supply voltage has direct impact on the result accuracy. To increase the accuracy, it is recommended to use a precise 3.3V voltage regulator for supplying the FF32 chip.

To measure voltages above the FF32 supply voltage, a simple voltage divider can be added, made out of two resistors. Select resistors with small tolerances. It is not recommended to measure voltages greater than +25V.

2.5 SPI Master Bus

The bus supports all SPI slave devices capable to communicate at any speed above 340kHz (*virtually any SPI device*).

SPI Master bus operates in Mode 0 (CPOL=0, CPHA=0).

Any General Purpose Pin can be assigned to any signal of the SPI bus. Consult *Read block analog inputs (0x32)*

Command:	Read analog input of multiple pins
Introduced:	In version 0.8
Syntax:	`0x32` **`Pins_mask[2]`**
Parameters:	• `Pins_mask` - bitmask of applicable pins from block B, 2 bytes, valid values are between 0x0000 and 0x0FFF inclusive
Response:	`0x32 Vcc_value Result[24]` or `0x0F`
Return value:	• `Vcc_value` - this byte value is either 3 (when Vcc = 3.3V) or 5 (when Vcc = 5V) • `Result` - two-bytes result representation of the analog voltage for each block B pin * `0x0F` - unknown command (applicable to silicon versions older than 0.8)
Discussion:	With this command an analog voltage applied to the un-masked pins is measured. Mask parameter defines applicable pins. When a bit in the mask is 1, the related pin in included in the acquisition. When a bit in the mask is 0, the related pin is not included and its result is set to 0. The Mask parameter data is formatted in little endian order. The returned Result array is always 24-bytes long, its 1st and 2nd bytes belong to pin B1, 3rd and 4th bytes to pin B2, etc. The returned value per pin is relative to the Vcc voltage. Result numbers are in range between 0 and 1023 inclusive. 0 represents 0V and 1023 represents Vcc. The characteristic is linear. The returned Vcc_value is defined by *REF2 pin*. If the chip is powered with different voltage, this flag mechanism can be ignored and the calculation is to be performed relatively to actual Vcc voltage.
Examples:	Command: `0x32 0x01 0x03` (Get analog voltage on pins B9, B2 and B1) Response: `0x23 0x03 0x00 0x00 0x01 0xF4 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0F 0xFF 0x00 0x00 0x00 0x00 0x00 0x00` (B1=0V, B2=1.613V, B9=3.300V; there is a 3.3V system)

Configure SPI bus (0x24) command and related *Application notes* documents for details.

2.6 I²C Master Bus / TWI Master Bus

The bus supports Standard clock speed (100kHz), Fast mode (400kHz) and also High speed (3.4MHz) devices (*virtually any I²C / TWI device*). No configuration is required related to the attached devices' speed.

A single master mode is allowed (FF32). Clock stretching is not supported.

Any General Purpose Pin can be assigned to any signal of the I²C / TWI bus. Consult *Configure I2C / TWI bus (0x27)* command and related *Application notes* documents for details.

The bus SDA signal requires external 4k7 pull-up resistor towards Vcc. Additional pull-up resistor for SCL signal is NOT required. Furthermore, it is recommended not to use SCL signal pull-up resistor in order to decrease power consumption.

2.7 1-Wire / MicroLAN Master Bus

1-Wire devices can be powered using both Vdd or parasitic DQ modes.

The DQ signal requires external 4k7 pull-up resistor towards Vcc.

Any General Purpose Pin can be assigned to any signal of the 1-Wire / MicroLAN bus. Consult *Configure 1-Wire / MicroLAN bus (0x2A)* command and related *Application notes* documents for details.

2.8 REF2 pin

REF2 input pin is used to identify the power supply selection. When the FF32 chip is powered with 5V, this REF2 pin is to be connected to the GND. In case of 3.3V power supply, the pin is to be connected to the Vcc.

Misconfiguration doesn't cause any electrical issues. Its only impact is improper data value contained in Analog results. See *Get analog voltage (0x23)* command.

2.9 CAP pin

This pin is dedicated for connection of external 330nF capacitor. Use low-ESR capacitor, recommended are ceramic capacitors with X7R dielectric. Make sure that they are rated at least for 50V.

3. Addressing and Identification

The FF32 chip supports customization of addressing and identification parameters.

3.1 Addressing

Addressing mechanism is needed when more than one chip is connected to the USB host. The code needs to distinguish which chip is to be addressed to execute a command.

The address factory setting is 0. This is also a default address used in the provided code examples.

When more than one chip is attached to the host, each chip has to have a unique address assigned. The address stored in each chip must be aligned with the application. Address setting for a chip can be altered with *Set address (0x10)* command.

3.2 Vendor and Product Identification

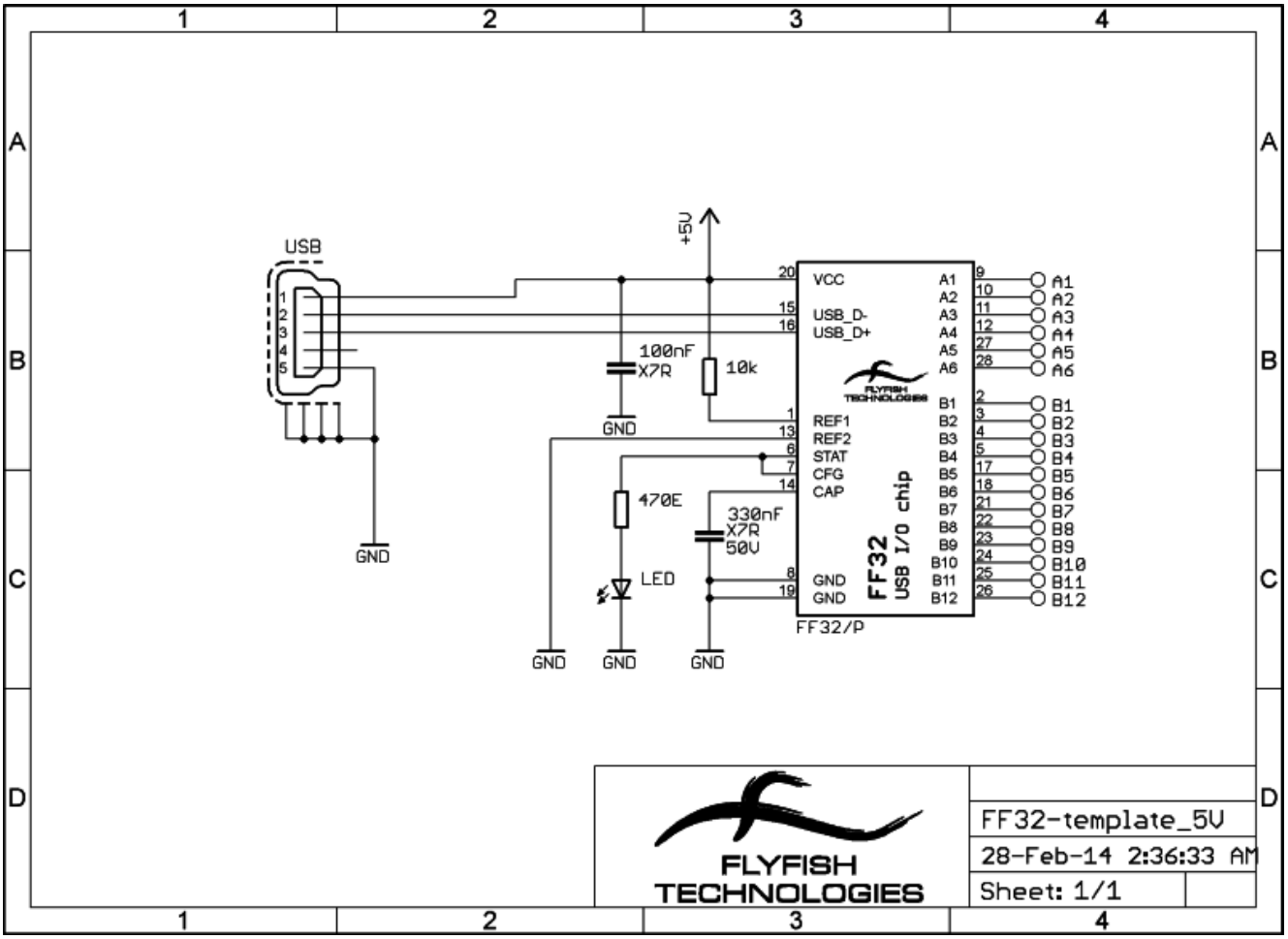
The chip can be programmed to identify the custom circuit it is part of. There are three identification strings which can be set: Vendor name, Product name and Product Serial number. Commands used to change the strings are *Set vendor (0x12)* , *Set product (0x14)* and *Set serial number (0x16)* respectively.

Content of these strings doesn't have any impact on the chip behavior. They are handled strictly by an application, when implemented.

4. Typical application circuits

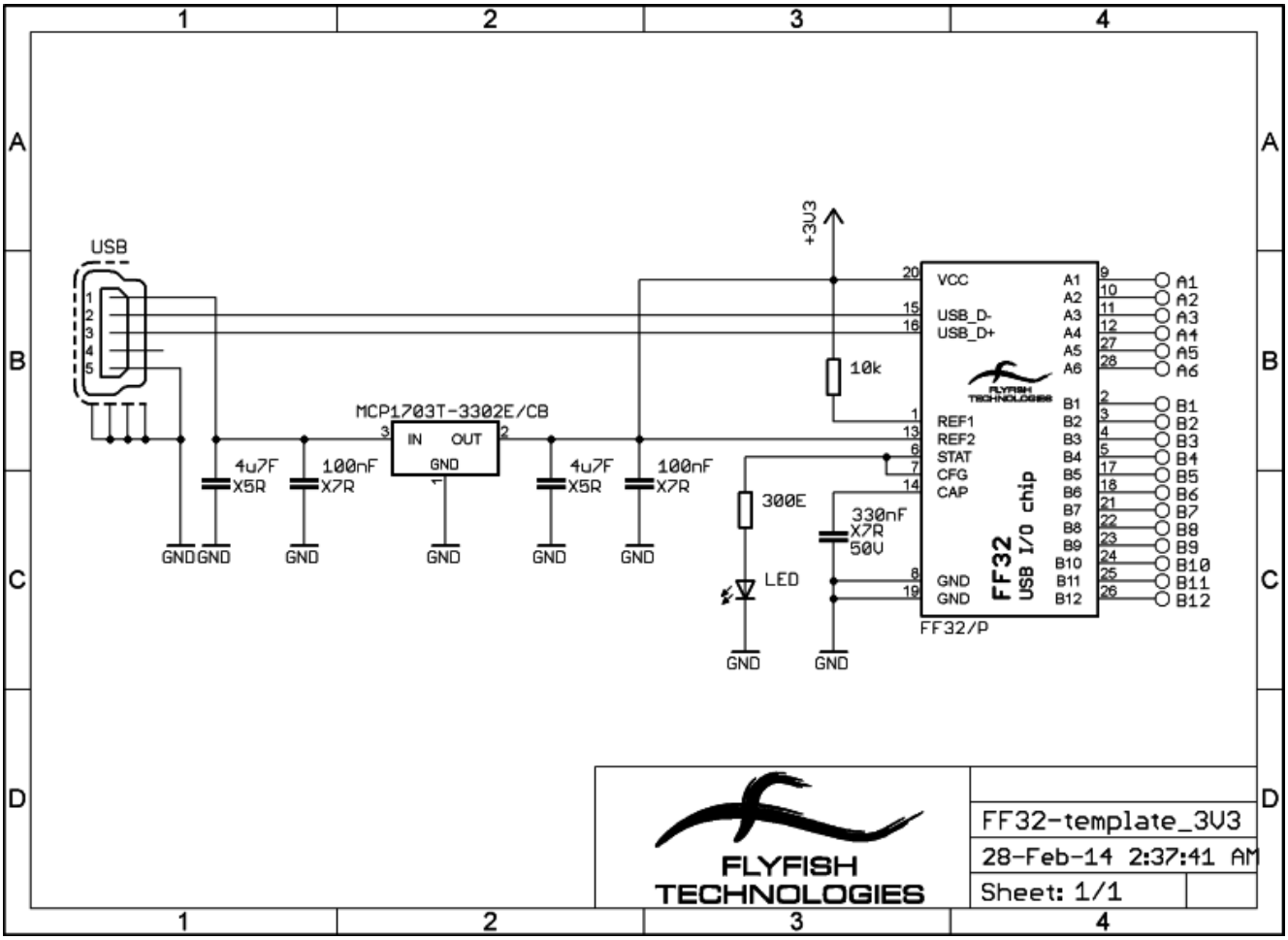
Most I2C and SPI chips are either 5V or 3.3V devices. To decrease number of cases where voltage level converters are required, you can design a circuit that both the FF32 chip and peripheral chips operate on the same voltage levels natively.

4.1 +5V power supply



Note: Some components attached to the FF32 chip (eg. sensors) could require additional elements to filter and stabilize circuit's power supply. Please consult the related components datasheets for their requirements.

4.2 +3.3V power supply



5. Initialization

After power-on, all general purpose pins are set as digital inputs. This state remains until changed by user code.

It is recommended to execute generic initialization custom code within host boot-up procedure. Since the general purpose pins do not contain any internal pull-up or pull-down resistors, all unused pins should be either:

- pulled to Vcc or GND by external 10k resistor or
- left unconnected and set as digital outputs by the initialization code.

6. Commands

Commands are initiated by the host. The chip responds to each command with a success or error message.

6.1 Get chip information (0x71)

Command:	Get chip identification and version
Introduced:	In version 0.1
Syntax:	`0x71`
Parameters:	None
Response:	`0x71 0x0F 0x0B` **`Ver_hi Ver_lo`**
Return values:	• `0x0F 0x0B` - constant value identifying FF32 chip • `Ver_hi` - silicon version, high byte • `Ver_lo` - silicon version, low byte
Discussion:	The command returns information about the chip identification and version. The version data can be used as a reference for checking features and updates listed in related documentation.
Example:	Command: `0x71` Response: `0x71 0x0F 0x0B 0x00 0x01` (FF32 chip version 0.1)

6.2 Set address (0x10)

Command:	Set chip USB address
Introduced:	In version 0.1
Syntax:	`0x10` **`Addr`**
Parameters:	• `Addr` - address, 1 byte, valid values are between 0 (factory default setting) and 127 (0x7F) inclusive
Response:	`0x0E` or `0x03`
Return value:	• `0x0E` - success status * `0x03` - address out of range
Discussion:	The address data is non-volatile (remains set also after power-off). When a single chip is attached to the host, this address value can remain unchanged (factory setting = 0). In case multiple FF32 chips are present on a host, an application needs to distinguish which chip is addressed via software API call. This is performed by addressing. Each chip attached to the host must have unique address. It is recommended to set address for each chip as an initial configuration step by attaching one chip at a time and assigning unique address to it by executing this command.
Examples:	Command: `0x10 0x07` (Set address to 7) Response: `0x0E` (OK, address has been set) Command: `0x10 0xF5` (Set address to 245) Response: `0x03` (Error, address out of range)

6.3 Get address (0x11)

Command:	Get chip USB address
Introduced:	In version 0.1
Syntax:	`0x11`
Parameters:	None
Response:	`0x11` **`Addr`**
Return value:	• `Addr` - current address, 1 byte
Discussion:	The command returns address set by *Set address* command. Factory default address value is 0.
Example:	Command: `0x11` Response: `0x11 0x09` (Address is 9)

6.4 Set vendor (0x12)

Command:	Set vendor string
Introduced:	In version 0.1
Syntax:	`0x12` **`Vendor[32]`**
Parameters:	• `Vendor` - name string, 32-byte array
Response:	`0x0E`
Return value:	• `0x0E` - success status
Discussion:	The vendor string is non-volatile (remains set also after power-off). The placeholder is 32-bytes long. Anything can be stored, including Unicode text. It is recommended that the string is zero-terminated.
Example:	Command: `0x12 'A' 'A' 'A' ' ' 'L' 't' 'd' '.' 0x00 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` (Set vendor to “AAA Ltd.”) Response: `0x0E` (OK, vendor string has been set)

6.5 Get vendor (0x13)

Command:	Get vendor string
Introduced:	In version 0.1
Syntax:	`0x13`
Parameters:	None
Response:	`0x13` **`Vendor[32]`**
Return value:	• `Vendor` - name string, byte array
Discussion:	The returned vendor array is 32-bytes long. Default value is *FLYFISH TECHNOLOGIES d.o.o.*
Example:	Command: `0x13` Response: `0x13 'A' 'A' 'A' ' ' 'L' 't' 'd' '.' 0x00 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` (Vendor is AAA Ltd.)

6.6 Set product (0x14)

Command:	Set product string
Introduced:	In version 0.1
Syntax:	`0x14` **`Product[32]`**
Parameters:	• `Product` - name string, 32-byte array
Response:	`0x0E`
Return value:	• `0x0E` - success status
Discussion:	The product string is non-volatile (remains set also after power-off). The placeholder is 32-bytes long. Anything can be stored, including Unicode text. It is recommended that the string is zero-terminated.
Example:	Command: `0x14 'D' 'e' 'm' 'o' 'B' 'o' 'a' 'r' 'd' 0x00` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20` (Set product to “DemoBoard”) Response: `0x0E` (OK, product string has been set)

6.7 Get product (0x15)

Command:	Get product string
Introduced:	In version 0.1
Syntax:	`0x15`
Parameters:	None
Response:	`0x15` **`Product[32]`**
Return value:	• `Product` - name string, byte array
Discussion:	The returned product array is 32-bytes long. Default value is *http://www.flyfish-tech.com/FF32*
Example:	Command: `0x15` Response: `0x15 'D' 'e' 'm' 'o' 'B' 'o' 'a' 'r' 'd' 0x00` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20` (Product is “DemoBoard”)

6.8 Set serial number (0x16)

Command:	Set product serial number
Introduced:	In version 0.1
Syntax:	`0x16` **`Serial[32]`**
Parameters:	• `Serial` - serial number string, 32-byte array
Response:	`0x0E`
Return value:	• `0x0E` - success status
Discussion:	The serial string is non-volatile (remains set also after power-off). The placeholder is 32-bytes long. Anything can be stored, including Unicode text. It is recommended that the string is zero-terminated.
Example:	Command: `0x16 '2' '0' '1' '4' 'X' 'Z' '7' '3' 'U' 0x00` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20` (Set serial to “2014XZ73U”) Response: `0x0E` (OK, product string has been set)

6.9 Get serial number (0x17)

Command:	Get product serial number
Introduced:	In version 0.1
Syntax:	`0x17`
Parameters:	None
Response:	`0x17` **`Serial[32]`**
Return value:	• `Serial` – serial number string, byte array
Discussion:	The returned serial number array is 32-bytes long. Default value is *00000000*
Example:	Command: `0x17` Response: `0x17 '2' 'Y' '6' '9' 'G' '-' 'D' 'e' 'v' 0x00` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20` `0x20 0x20 0x20 0x20` (Serial number is "2Y69G-Dev")

6.10 Set digital output (0x20)

Command:	Set pin digital value
Introduced:	In version 0.1
Syntax:	`0x20` **`Pin_block Pin_number State`**
Parameters:	• `Pin_block` - pin block identifier, 1 byte, valid values are 'A' and 'B' • `Pin_number` - pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `State` - output pin state, 1 byte, valid values are 0 and 1
Response:	`0x0E` or `0x01`
Return value:	• `0x0E` - success status * `0x01` - improper pin name
Discussion:	With this command a voltage of selected pin is set according to the given State parameter value. Its value 0 represents low level (GND) and value 1 is high level (Vcc). If the pin was not set as digital output previously, the command sets it prior to setting the state.
Examples:	Command: `0x20 0x0A 0x04 0x01` (Set pin A4 to digital 1) Response: `0x0E` (OK) Command: `0x20 0x0B 0x0C 0x00` (Set pin B12 to digital 0) Response: `0x0E` (OK) Command: `0x20 0x0B 0x10 0x01` (Set pin B16 to digital 1) Response: `0x01` (Error, improper pin name - pin B16 doesn't exist)

6.11 Set block digital outputs (0x30)

Command:	Set digital values for multiple pins.
Introduced:	In version 0.3
Syntax:	`0x30` **`Pins_block Pins_mask[2] States[2]`**
Parameters:	• `Pins_block` - pins block identifier, 1 byte, valid values are 'A' and 'B' • `Pins_mask` - bitmask of applicable pins, 2 bytes, valid values are between 0x0000 and 0x003F inclusive for block A and between 0x0000 and 0x0FFF inclusive for block B • `States` - output pins state, 2 bytes, valid values are between 0x0000 and 0x003F inclusive for block A and between 0x0000 and 0x0FFF inclusive for block B
Response:	`0x0E` or `0x05` or `0x0F`
Return value:	• `0x0E` - success status * `0x05` - improper pins block identifier * `0x0F` - unknown command (applicable to silicon versions older than 0.3)
Discussion:	This command sets digital output values for all pins from one block. Mask parameter defines applicable pins. When a bit in the mask is 1, the related pin output is set according to an associated value in the States parameter. When a bit in the mask is 0, the related pin remains untouched. The Mask and States data is formatted in little endian order.
Examples:	Command: `0x30 0x0A 0x00 0x01 0x00 0x01` (Set pin A1 to digital 1) Response: `0x0E` (OK) Command: `0x30 0x0A 0x00 0x03 0x00 0x02` (Set pins A2 to digital 1 and A1 to digital 0) Response: `0x0E` (OK) Command: `0x30 0x0B 0x01 0x00 0x00 0x00` (Set pin B9 to digital 0) Response: `0x0E` (OK) Command: `0x30 0x0B 0x0F 0xFF 0x00 0x00` (Set all pins from B block to digital 0) Response: `0x0E` (OK)

6.12 Read digital input (0x21)

Command:	Read pin as digital input
Introduced:	In version 0.1
Syntax:	`0x21` **`Pin_block Pin_number`**
Parameters:	• `Pin_block` - pin block identifier, 1 byte, valid values are 'A' and 'B' • `Pin_number` - pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B
Response:	`0x21 State` or `0x01`
Return value:	• `State` - state is 0 or 1 * `0x01` - improper pin name
Discussion:	The command returns selected pin digital input value. GND voltage on the input is represented by 0 and Vcc voltage as 1. If the pin was not set as a digital input previously, the command sets it prior to reading the state.
Examples:	Command: `0x21 0x0A 0x01` (Get pin A1 input state) Response: `0x21 0x01` (State is 1 - Vcc is applied to the input)

	Command: 0x21 0x0B 0x0A (Get pin B10 input state) Response: 0x21 0x00 (State is 0 - the pin is at GND potential) Command: 0x21 0x0A 0x08 (Get pin A8 input state) Response: 0x01 (Error, improper pin name - pin A8 doesn't exist)

6.13 Read block digital inputs (0x31)

Command:	Read digital input of multiple pins
Introduced:	In version 0.3
Syntax:	0x31 **Pins_block Pins_mask[2]**
Parameters:	• Pins_block - pins block identifier, 1 byte, valid values are 'A' and 'B' • Pins_mask - bitmask of applicable pins, 2 bytes, valid values are between 0x0000 and 0x003F inclusive for block A and between 0x0000 and 0x0FFF inclusive for block B
Response:	0x31 States[2] or 0x05 or 0x0F
Return value:	• States - output pins state, 2 bytes, valid values are between 0x0000 and 0x003F inclusive for block A and between 0x0000 and 0x0FFF inclusive for block B * 0x05 - improper pins block identifier * 0x0F - unknown command (applicable to silicon versions older than 0.3)
Discussion:	This command returns pins digital input value according to the given bitmask. For enabled inputs (bit set to 1) in the given mask, a GND voltage on the input is represented by 0 and Vcc voltage as 1. When a bit in the mask is 0, the related pin remains untouched and the returned associated status bit is set to 0. If the enabled pins were not set as digital inputs previously, the command sets them prior to reading the state.
Examples:	Command: 0x31 0x0A 0x00 0x05 (Get pins A1 and A3 input state) Response: 0x31 0x00 0x01 (Pin's A1 state is 1, pin's A3 state is 0) Command: 0x31 0x0B 0x01 0x82 (Get pins B9, B8 and B2 input state) Response: 0x31 0x01 0x00 (Pin's B9 state is 1, pins' B8 and B2 state is 0) Command: 0x31 0x0B 0x0F 0xFF (Get input state of all pins from B block) Response: 0x31 0x0A 0x00 (Pins' A12 and A10 state is 1, remaining are 0)

6.14 Set PWM output (0x22)

Command:	Set pin PWM ratio
Introduced:	In version 0.1
Syntax:	`0x22 0x0A` **`Pin_number Ratio`**
Parameters:	• `0x0A` - constant value, pin from block A • `Pin_number` - pin number, 1 byte, valid values are between 1 and 6 inclusive • `Ratio` - PWM ratio, 1 byte, valid values are between 0 and 100
Response:	`0x0E` or `0x01`
Return value:	• `0x0E` - success status * `0x01` - improper pin name
Discussion:	This command sets the selected pin as PWM output and a ratio is set according to the given Ratio parameter value. Its value 0 represents constant low level (GND) and value 100 constant high level (Vcc). If the Ratio value is higher than 100, the output is set on constant high level (Vcc).
Examples:	Command: `0x22 0x0A 0x02 0x32` (Set pin A2 to 50% PWM) Response: `0x0E` (OK) Command: `0x22 0x0A 0x04 0x5A` (Set pin A4 to 90% PWM) Response: `0x0E` (OK) Command: `0x22 0x0B 0x01 0x05` (Set pin B1 to 5% PWM) Response: `0x01` (Error, improper pin name - pin B1 is not PWM-capable)

6.15 Get analog voltage (0x23)

Command:	Read analog voltage on a pin
Introduced:	In version 0.1
Syntax:	`0x23 0x0B` **`Pin_number`**
Parameters:	• `0x0B` - constant value, pin from block B • `Pin_number` - pin number, 1 byte, valid values are between 1 and 12 inclusive
Response:	`0x23 Vcc_value Result[2]` or `0x01`
Return values:	• `Vcc_value` - this value is either 3 (when Vcc = 3.3V) or 5 (when Vcc = 5V) • `Result` - two-bytes result representation of the analog voltage * `0x01` - improper pin name
Discussion:	With this command an analog voltage applied to the given pin is measured. The returned value is relative to the Vcc voltage. Result numbers are in range between 0 and 1023 inclusive. 0 represents 0V and 1023 represents Vcc. The characteristic is linear. The returned Vcc_value is defined by *REF2 pin*. If the chip is powered with different voltage, this flag mechanism can be ignored and the calculation is to be performed relatively to actual Vcc voltage.
Examples:	Command: `0x23 0x0B 0x02` (Get analog voltage on pin B2) Response: `0x23 0x03 0x01 0xF4` (Voltage is 1.613V; there is a 3.3V system) Command: `0x23 0x0B 0x03` (Get analog voltage on pin B3) Response: `0x23 0x05 0x01 0xF4` (Voltage is 2.444V; there is a 5V system) Command: `0x23 0x0B 0x00` (Get analog voltage on pin B0) Response: `0x01` (Error, improper pin name - pin B0 doesn't exist) Command: `0x23 0x0A 0x02` (Get analog voltage on pin A2) Response: `0x01` (Error, improper pin name - pin A2 is not Analog-capable)

6.16 Read block analog inputs (0x32)

Command:	Read analog input of multiple pins
Introduced:	In version 0.8
Syntax:	`0x32` **`Pins_mask[2]`**
Parameters:	• `Pins_mask` - bitmask of applicable pins from block B, 2 bytes, valid values are between 0x0000 and 0x0FFF inclusive
Response:	`0x32 Vcc_value Result[24]` or `0x0F`
Return value:	• `Vcc_value` - this byte value is either 3 (when Vcc = 3.3V) or 5 (when Vcc = 5V) • `Result` - two-bytes result representation of the analog voltage for each block B pin * `0x0F` - unknown command (applicable to silicon versions older than 0.8)
Discussion:	With this command an analog voltage applied to the un-masked pins is measured. Mask parameter defines applicable pins. When a bit in the mask is 1, the related pin in included in the acquisition. When a bit in the mask is 0, the related pin is not included and its result is set to 0. The Mask parameter data is formatted in little endian order. The returned Result array is always 24-bytes long, its 1st and 2nd bytes belong to pin B1, 3rd and 4th bytes to pin B2, etc. The returned value per pin is relative to the Vcc voltage. Result numbers are in range between 0 and 1023 inclusive. 0 represents 0V and 1023 represents Vcc. The characteristic is linear. The returned Vcc_value is defined by *REF2 pin*. If the chip is powered with different voltage, this flag mechanism can be ignored and the calculation is to be performed relatively to actual Vcc voltage.
Examples:	Command: `0x32 0x01 0x03` (Get analog voltage on pins B9, B2 and B1) Response: `0x23 0x03 0x00 0x00 0x01 0xF4 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0F 0xFF 0x00 0x00 0x00 0x00 0x00 0x00` (B1=0V, B2=1.613V, B9=3.300V; there is a 3.3V system)

6.17 Configure SPI bus (0x24)

Command:	Assign SPI bus pins for next SPI transactions
Introduced:	In version 0.1
Syntax:	`0x24` **`CSPin_block CSPin_number SCKPin_block SCKPin_number MOSIPin_block MOSIPin_number MISOPin_block MISOPin_number`** `0x00`
Parameters:	• `CSPin_block` - CS pin block identifier, 1 byte, valid values are 'A' and 'B' • `CSPin_number` - CS pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `SCKPin_block` - SCK pin block identifier, 1 byte, valid values are 'A' and 'B' • `SCKPin_number` - SCK pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `MOSIPin_block` - MOSI pin block identifier, 1 byte, valid values are 'A' and 'B' • `MOSIPin_number` - MOSI pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `MISOPin_block` - MISO pin block identifier, 1 byte, valid values are 'A' and 'B' • `MISOPin_number` - MISO pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `0x00` - constant value
Response:	`0x0E` or `0x01`
Return value:	• `0x0E` - success status * `0x01` - at least one improper pin name
Discussion:	This command sets SPI bus pins for next SPI transaction(s). The setup remains valid until changed again with this command. These settings are volatile (are lost after power-off). Default power-on pins configuration: • CS pin = A1 • SCK pin = A2 • MOSI pin = A3 • MISO pin = A4
Examples:	Command: `0x24 0x0B 0x02 0x0B 0x04 0x0B 0x05 0x0A 0x01 0x00` (CS=B3, SCK=B4, MOSI=B5, MISO=A1) Response: `0x0E` (OK) Command: `0x24 0x0B 0x02 0x0B 0x04 0x0B 0x05 0x0A 0x09 0x00` (CS=B3, SCK=B4, MOSI=B5, MISO=A9) Response: `0x01` (Error, improper pin name - pin A9 doesn't exist)

6.18 Write to SPI device (0x25)

Command:	Write data to SPI device
Introduced:	In version 0.1
Syntax:	`0x25` **`Data_len Data[]`**
Parameters:	• `Data_len` - Number of applicable bytes in Data parameter array, 1 byte, valid values are between 0 and 60 inclusive • `Data` – Data to be written to the SPI device. Maximum array length is 60 bytes.
Response:	`0x0E` or `0x02`
Return value:	• `0x0E` - success status * `0x02` - improper Data_len parameter value
Discussion:	This command writes data values to the SPI bus.
Examples:	Command: `0x25 0x04 0x02 0x12 0x34 0x41` (Write "A" in the SPI EEPROM at address 0x1234)

	Response: `0x0E` (OK) Command: `0x25 0x03 0x41 0x42 0x43` (Write "ABC" to the SPI bus) Response: `0x0E` (OK) Command: `0x25 0x06` (Write 6 undefined bytes to the SPI device) Response: `0x0E` (OK) Command: `0x25 0xBB 0x00 0x00` (Write 187 bytes to SPI device) Response: `0x02` (Error, improper length)

6.19 Read from SPI device (0x26)

Command:	Read data from SPI device
Introduced:	In version 0.1
Syntax:	`0x26` **`WRData_len RDData_len WRData[]`**
Parameters:	• `WRData_len` - Number of applicable bytes in WRData parameter array, 1 byte, valid values are between 0 and 60 inclusive • `RDData_len` - Number of bytes to be read from the device, 1 byte, valid values are between 0 and 60 inclusive • `WRData` – Data to be written to the SPI device before reading from it. Maximum array length is 60 bytes.
Response:	`0x26 RDData_len RDData[]` or `0x02`
Return value:	• `RDData_len` - number of bytes in RDData array • `RDData` - data read * `0x02` - improper WRData_len or RDData_len parameter value
Discussion:	This command writes and reads data values to and from the SPI device. When WRData_len is 0, just read step is performed. If RDData_len is 0, this command is equivalent to the *Write to SPI device (0x25)* command.
Examples:	Command: `0x26 0x03 0x05 0x03 0x12 0x34` (Read 5 bytes from SPI EEPROM starting at address 0x1234) Response: `0x26 0x05 0x41 0x42 0x43 0x44 0x45` ("ABCDE") Command: `0x26 0x00 0x03` (Read 3 bytes from the SPI bus) Response: `0x26 0x03 0x31 0x32 0x33` ("123") Command: `0x26 0x00 0xBA` (Read 186 bytes from SPI bus) Response: `0x02` (Error, improper length)

6.20 Configure I²C / TWI bus (0x27)

Command:	Assign I²C / TWI bus pins for next transactions
Introduced:	In version 0.1
Syntax:	`0x27` **`SCLPin_block SCLPin_number SDAPin_block SDAPin_number`**
Parameters:	• `SCLPin_block` - SCL pin block identifier, 1 byte, valid values are 'A' and 'B' • `SCLPin_number` - SCL pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B • `SDAPin_block` - SDA pin block identifier, 1 byte, valid values are 'A' and 'B' • `SDAPin_number` - SDA pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B
Response:	`0x0E` or `0x01`
Return value:	• `0x0E` - success status * `0x01` - at least one improper pin name
Discussion:	This command sets I²C / TWI bus pins for next transaction(s). The setup remains valid until changed again with this command. These settings are volatile (are lost after power-off). Default power-on pins configuration: • SCL pin = A5 • SDA pin = A6
Examples:	Command: `0x27 0x0B 0x08 0x0B 0x0C` (SCL=B8, SDA=B12) Response: `0x0E` (OK) Command: `0x27 0x0B 0x02 0x0A 0x08` (SCL=B2, SDA=A8) Response: `0x01` (Error, improper pin name - pin A8 doesn't exist)

6.21 Write to I²C / TWI device (0x28)

Command:	Write data to I²C / TWI device
Introduced:	In version 0.1
Syntax:	`0x28` **`Data_len Data[]`**
Parameters:	• `Data_len` - Number of applicable bytes in Data parameter array, 1 byte, valid values are between 0 and 60 inclusive • `Data` - Data to be written to the I²C / TWI device. Maximum array length is 60 bytes.
Response:	`0x0E` or `0x02` or `0x04`
Return value:	• `0x0E` - success status * `0x02` - improper Data_len parameter value * `0x04` - absent acknowledge from the device
Discussion:	This command writes data values to the I²C / TWI device.
Examples:	Command: `0x28 0x04 0xA2 0x12 0x34 0x41` (Write "A" in the I²C EEPROM at address 0x1234) Response: `0x0E` (OK) Command: `0x28 0x03 0xA2 0x12 0x34` (Set I²C EEPROM internal address pointer for next read command to 0x1234) Response: `0x0E` (OK) Command: `0x28 0xB9 0x00 0x00` (Write 185 bytes to I²C device) Response: `0x02` (Error, improper length)

	Command: `0x28 0x03 0x41 0x42 0x43` (Write "ABC" to the I²C bus) Response: `0x04` (Error, absent acknowledge from the device)

6.22 Read from I²C / TWI device (0x29)

Command:	Read data from I²C / TWI device
Introduced:	In version 0.1
Syntax:	`0x29` **`WRData_len RDData_len WRData[]`**
Parameters:	• `WRData_len` - Number of applicable bytes in WRData parameter array, 1 byte, valid values are between 0 and 60 inclusive • `RDData_len` - Number of bytes to be read from the device, 1 byte, valid values are between 0 and 60 inclusive • `WRData` – Data to be written to the I²C / TWI device before reading from it. Maximum array length is 60 bytes.
Response:	`0x29 RDData_len RDData[]` or `0x02` or `0x04`
Return value:	• `RDData_len` - number of bytes in RDData array • `RDData` - data read * `0x02` - improper WRData_len or RDData_len parameter value * `0x04` - absent acknowledge from the device
Discussion:	This command writes and reads data values to and from the I²C / TWI device. When WRData_len is 0, just read step is performed. If RDData_len is 0, this command is equivalent to the *Write to I2C / TWI device (0x28)* command.
Examples:	Command: `0x29 0x01 0x05 0xA3` (Read 5 bytes from I²C EEPROM starting at current value of EEPROM's internal address pointer) Response: `0x29 0x05 0x41 0x42 0x43 0x44 0x45` ("ABCDE") Command: `0x29 0x00 0xBC` (Read 188 bytes from I²C bus) Response: `0x02` (Error, improper length) Command: `0x29 0x00 0x03` (Read 3 bytes from the I²C bus) Response: `0x04` (Error, absent acknowledge from the device)

6.23 Configure 1-Wire / MicroLAN bus (0x2A)

Command:	Assign 1-Wire / MicroLAN bus pin for next transactions
Introduced:	In version 0.1
Syntax:	`0x2A` **`DQPin_block DQPin_number`**
Parameters:	• `DQPin_block` - DQ pin block identifier, 1 byte, valid values are 'A' and 'B' • `DQPin_number` - DQ pin number, 1 byte, valid values are between 1 and 6 inclusive for block A and between 1 and 12 inclusive for block B
Response:	`0x0E` or `0x01`
Return value:	• `0x0E` - success status * `0x01` - improper pin name
Discussion:	This command sets 1-Wire / MicroLAN bus pin for next transaction(s). The setup remains valid until changed again with this command. These settings are volatile (are lost after power-off). Default power-on DQ pin is B1.
Examples:	Command: `0x2A 0x0B 0x07` (DQ=B7) Response: `0x0E` (OK) Command: `0x2A 0x0B 0x0F` (DQ=B15) Response: `0x01` (Error, improper pin name - pin B15 doesn't exist)

6.24 Reset 1-Wire / MicroLAN devices (0x2D)

Command:	Reset 1-Wire / MicroLAN devices
Introduced:	In version 0.5
Syntax:	`0x2D`
Parameters:	None
Response:	`0x2D` **`Detected`** or `0x0F`
Return value:	• `Detected` – Flag if any device was detected on the bus, 1 byte. Value 0 indicates that no device is present on the bus; value 1 indicates that one or more devices have responded. * `0x0F` - unknown command (applicable to silicon versions older than 0.5)
Discussion:	This command performs the 1-Wire / MicroLAN bus reset sequence and checks for a response from devices attached.
Examples:	Command: `0x2D` Response: `0x2D 0x01` (One or more devices are present) Command: `0x2D` Response: `0x2D 0x00` (No device present)

6.25 Write to 1-Wire / MicroLAN device (0x2B)

Command:	Write data to 1-Wire / MicroLAN device
Introduced:	In version 0.1
Syntax:	`0x2B` **`Data_len Data[]`**
Parameters:	• `Data_len` - Number of applicable bytes in Data parameter array, 1 byte, valid values are between 0 and 60 inclusive • `Data` - Data to be written to the 1-Wire / MicroLAN device. Maximum array length is 60 bytes.
Response:	`0x0E` or `0x02` or `0x04`
Return value:	• `0x0E` - success status * `0x02` - improper Data_len parameter value * `0x04` - absent acknowledge from the device
Discussion:	This command writes data values to the 1-Wire / MicroLAN device.
Examples:	Command: `0x2B 0x02 0xCC 0x44` (Convert temperature of DS18B20) Response: `0x0E` (OK) Command: `0x2B 0xBD 0x00` (Write 189 bytes to 1-Wire device) Response: `0x02` (Error, improper length) Command: `0x2B 0x03 0x43 0x42 0x41` (Write “CBA” to the 1-Wire bus) Response: `0x04` (Error, absent acknowledge from the device)

6.26 Write bit to 1-Wire / MicroLAN bus (0x2E)

Command:	Write one bit to 1-Wire / MicroLAN bus
Introduced:	In version 0.5
Syntax:	`0x2E` **`DataBit`**
Parameters:	• `DataBit` – The value to be written to the 1-Wire / MicroLAN bus, 1 byte. Non-zero value causes bit '1' write.
Response:	`0x0E` or `0x0F`
Return value:	• `0x0E` - success status * `0x0F` - unknown command (applicable to silicon versions older than 0.5)
Discussion:	This command writes one bit to the 1-Wire / MicroLAN bus. If the DataBit value equals to 0, then '0' timeslot is created. For any other value the '1' timeslot is created.
Examples:	Command: `0x2E 0x00` (Write bit '0' to the bus) Response: `0x0E` (OK) Command: `0x2E 0x01` (Write bit '1' to the bus) Response: `0x0E` (OK) Command: `0x2E 0x6A` (Write bit '1' to the bus) Response: `0x0E` (OK)

6.27 Read from 1-Wire / MicroLAN device (0x2C)

Command:	Read data from 1-Wire / MicroLAN device
Introduced:	In version 0.1
Syntax:	`0x2C` **`WRData_len RDData_len WRData[]`**
Parameters:	• `WRData_len` - Number of applicable bytes in WRData parameter array, 1 byte, valid values are between 0 and 60 inclusive • `RDData_len` - Number of bytes to be read from the device, 1 byte, valid values are between 0 and 60 inclusive • `WRData` – Data to be written to the 1-Wire / MicroLAN device before reading from it. Maximum array length is 60 bytes.
Response:	`0x2C RDData_len RDData[]` or `0x02` or `0x04`
Return value:	• `RDData_len` - number of bytes in RDData array • `RDData` - data read * `0x02` - improper WRData_len or RDData_len parameter value * `0x04` - absent acknowledge from the device
Discussion:	This command writes and reads data values to and from the 1-Wire / MicroLAN device. When WRData_len is 0, just read step is performed. If RDData_len is 0, this command is equivalent to the *Reset 1-Wire / MicroLAN devices (0x2D)* command.
Examples:	Command: `0x2C 0x02 0x09 0xCC 0xBE` (Read Scratchpad of DS18S20) Response: `0x2C 0x09 0x2F 0x02 0x00 0x00 0x3F 0xFF 0x0C 0x10 0x7D` Command: `0x2C 0x00 0xC1` (Read 193 bytes from 1-Wire bus) Response: `0x02` (Error, improper length) Command: `0x2C 0x00 0x03` (Read 3 bytes from the 1-Wire bus) Response: `0x04` (Error, absent acknowledge from the device)

6.28 Read bit from 1-Wire / MicroLAN bus (0x2F)

Command:	Read one bit from 1-Wire / MicroLAN bus
Introduced:	In version 0.5
Syntax:	`0x2F`
Parameters:	None
Response:	`0x2F DataBit` or `0x0F`
Return value:	• `DataBit` - level present on the bus at scan timestamp * `0x0F` - unknown command (applicable to silicon versions older than 0.7)
Discussion:	This command returns value of the read timeslot on the 1-Wire / MicroLAN bus. DataBit value equals to 0 if the level was low and equals to 1 if the level was high.
Examples:	Command: `0x2F` Response: `0x2F 0x00` (bit '0') Command: `0x2F` Response: `0x2F 0x01` (bit '1')

6.29 Set mode (0x40)

Command:	Set various modes
Introduced:	In version 0.6
Syntax:	`0x40 Switch`
Parameters:	• `Switch` – Mode item, 1 byte, valid values: o 0x00 – enable 1-Wire implicit reset within commands *Write to 1-Wire device (0x2B)* and *Read from 1-Wire device (0x2C)*; **default** o 0x01 – disable 1-Wire implicit reset within commands *Write to 1-Wire device (0x2B)* and *Read from 1-Wire device (0x2C)*
Response:	`0x0E` or `0x06` or `0x0F`
Return value:	• `0x0E` - success status * `0x06` - invalid Switch value * `0x0F` - unknown command (applicable to silicon versions older than 0.6)
Discussion:	This modes change command is volatile, applicable till next power-on.
Examples:	Command: `0x40 0x01` Response: `0x0E` (1-Wire implicit Reset command is not executed from now on) Command: `0x40 0x10` Response: `0x06` (Error, invalid Switch value)

6.30 Set default item (0x41)

Command:	Set various default items
Introduced:	In version 0.7
Syntax:	`0x41 Item [Parameters]`
Parameters:	• `Item` – 1 byte, valid values: o 0x01 – set default 1-Wire DQ pin. Parameters are pin name, 2 bytes
Response:	`0x0E` or `0x01` or `0x06` or `0x0F`
Return value:	• `0x0E` - success status * `0x01` - invalid Parameter (pin specified) * `0x06` - invalid Item value * `0x0F` - unknown command (applicable to silicon versions older than 0.7)
Discussion:	This command stores values in non-volatile memory. Values are applicable at power-on.
Examples:	Command: `0x41 0x01 0x0B 0x09` Response: `0x0E` (1-Wire default DQ pin is B9) Command: `0x41 0x01 0x0A 0x09` Response: `0x01` (Error, invalid DQ default pin A9) Command: `0x41 0x13` Response: `0x06` (Error, Item value)

7. Codename :: Emona2000

FLYFISH TECHNOLOGIES headquarters is located in Ljubljana, Slovenia.

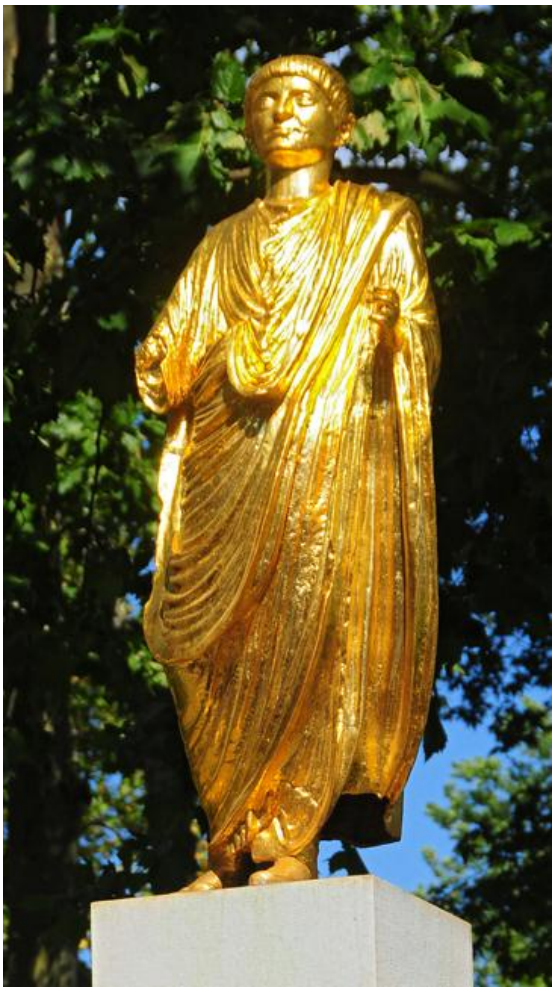
In 2014 city of Ljubljana celebrates 2000 years of the Roman City of Emona, situated on the site of today's Ljubljana, the capital city of Slovenia.

In 14 AD, the Roman colony of Emona (Colonia Iulia Aemona) unquestionably already stood on the site of the present-day city centre of Ljubljana. This is evidenced by an inscription about a donation that the city received from the emperors Augustus and Tiberius.

Due to its geographical position, Emona played an important role in the Roman Empire's system of defense and flourished from the 1st to the 5th century AD. It was part of the Roman province of Italy, as its easternmost city.

Emona was built on a Roman model and was governed in accordance with Roman political and religious principles. It is believed to have been inhabited by around 5,000 colonists, whose occupations were agriculture, trade, and crafts.

Emona was also an important Early Christian centre with its own goddess, Equrna.



Ancient statue of Emona resident, Photo: Dunja Wedam

Numerous remains have been excavated, such as parts of the Roman wall, residential houses, statues, tombstones, several mosaics, parts of the paleochristian baptistry, etc. that can be still seen today.
Some of the remains of ancient Emona have survived in Ljubljana's streets and squares, others are on display at the City Museum of Ljubljana and the National Museum of Slovenia.

Sources:
http://en.wikipedia.org/wiki/Emona
http://www.visitljubljana.com/en/activities/culture-art/49296/detail.html

What life was like in Emona 2000 years ago; animation: http://www.youtube.com/watch?v=8YrqEQs0PRg