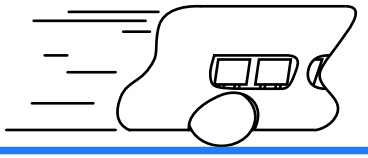
koreio



user manual

K-TEAM

version 1.0
january 2005

Documentation Author

Pierre Bureau for K-Team S.A.
Ch. de Vuasset, CP 111
1028 Préverenges
Switzerland

email: info@k-team.com
Url: www.k-team.com

Trademark Acknowledgments:

IBM PC: International Business Machine Corp.
Macintosh: Apple Corp.
SUN Sparc-Station: SUN Microsystems Corp.
LabView: National Instruments Corp.
MatLab: MathWorks Corp.
Webots: Cyberbotics
Khepera: K-Team and LAMI

Legal Notice:

- The content of this manual is subject to change without notice.
- All effort have been made to ensure the accuracy of the content of this manual. However, should any error be detected, please inform K-Team S.A.
- The above notwithstanding K-Team can assume no responsibility for any error in this manual.

Table Of Contents

1 Introduction

The KoreMotor controler board can be used as a KoreBot extension or as a standalone module that provides several inputs and ouputs features. All digital IO can be used as low frequency PWM outputs as well as standard IO. A set of 10 bits, timestamped analog inputs can be polled consitently. For communication with other devices, the KoreIO can be used as brigde to an I2C or CAN bus.

The KoreIO can receive commands from a serial RS232 link, an I2C bus or a KNet connection from a Koala or a KoreBot.

2.1 Overview

The KoreIO hardware is composed of a main microcontroller, a CAN bus controller, an analog outputs manager and a power outputs circuits.

The main microcontroller handles communication and most basic functions such a IO processing. A can controller is available to communicate with other CAN devices and 8bit analog outputs are managed with a dedicated chip. Another main module is the power outputs circuits with transistors able to drive 1 Ampere through the designed board outputs.

The required connections to the board are defined by the required functions for a given application. Dip switches control the board *running mode* and communication interface. The main connector blocks on each side of the board provides conenctions to all the board features. Communication can go through the serial connector, the KB-250 extension bus connectors or the KNet connector.

Figures 2.1 and 2.2 describe the board main components.

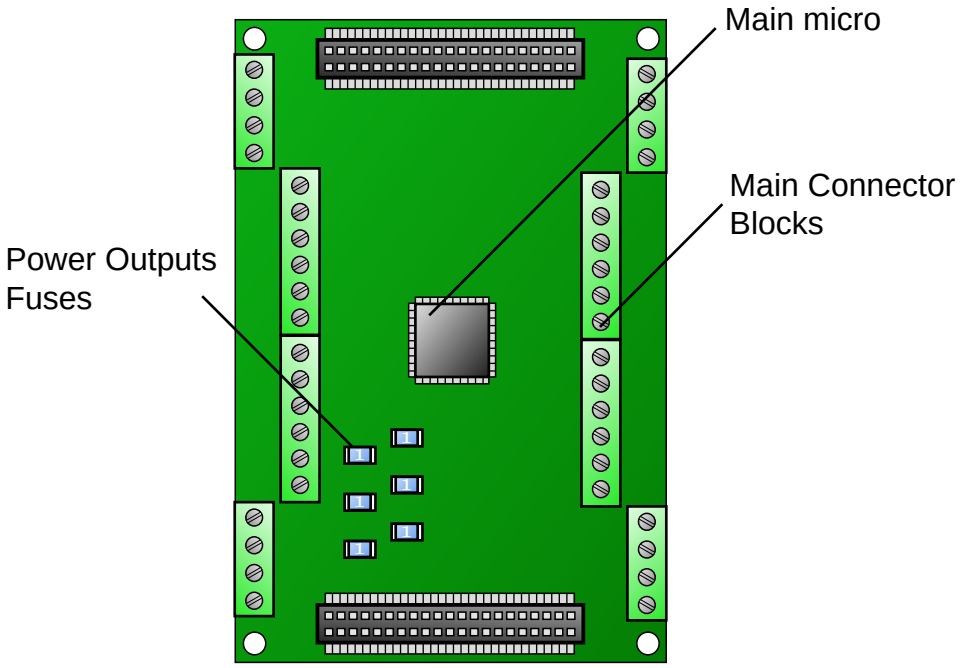


Figure 2.1: KoreIO hardware overview

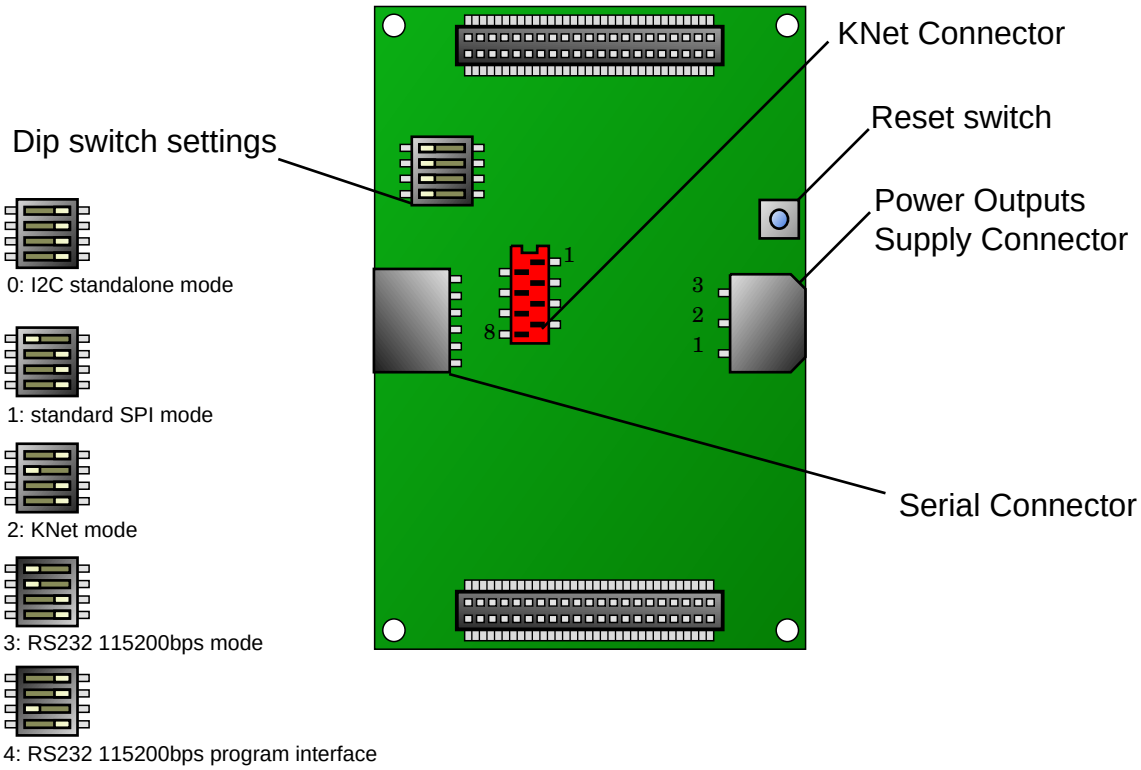


Figure 2.2: Dip switch settings

2.1.1 Dip Switch Settings

The running mode for the KoreIO is set with the dip swicth position at startup. The settings are described on figure 2.2. If the switch position is modified at run time, the mode will not change until the next reset.

I2C Standalone : That mode is used to control the board from an I2C bus. The I2C bus is directly connected to the main microcontroller, and the I2C address is detailled in section 2.1.2. The KoreIO can be connecter to any I2C master as a standalone I2C device, for instance a KoreBot can be configured as an I2C master for the KoreMotor.

Koala KNet : The Koala KNet mode is the classical KNet protocol implemented on the Koala robot and the Khepera robot. This mode is mainly designed to connect a KoreMotor as a Koala extension, refer to the Koala user manual for details about the KNet protocol.

RS232 User Interface (115200): User serial commands (see section 4.1) can be send to the KoreIO serial interface with this mode. The serial line settings should be 115200 bps, no parity, 8 data bits and 2 stop bits.

RS232 Program Interface (115200): Program serial commands (see section 4.4) can be send to the KoreIO serial interface in this mode. The serial line settings are the same as the RS232 User Interface settings.

2.1.2 Controller I2C Address

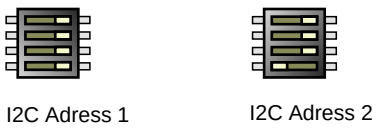


Figure 2.3: KoreIO I2C address

The last dip switch, as displayed on figure 2.3 is used to choose the I2C address for the controller. The address choice is usefull to stack two KoreIO together, using the same I2C bus.

Address 1	0x1C
Address 2	0x1D

2.1.3 Connector Blocks Connections

The KoreIO board provides 56 side connections to the various functions. Figure 2.4 shows how to connect a cable to a side connector. The function for each connection is writen on the board itself and explained in the following table.

Name	Function	Name	Function
AD0	Analog to Digital input 0	IO0	Digital IO 0
AD1	Analog to Digital input 1	IO1	Digital IO 1
AD2	Analog to Digital input 2	IO2	Digital IO 2
AD3	Analog to Digital input 3	IO3	Digital IO 3
AD4	Analog to Digital input 4	IO4	Digital IO 4
AD5	Analog to Digital input 5	IO5	Digital IO 5
AD6	Analog to Digital input 6	IO6	Digital IO 6
AD7	Analog to Digital input 7	IO7	Digital IO 7
AD8	Analog to Digital input 8	IO8	Digital IO 8
AD9	Analog to Digital input 9	IO9	Digital IO 9
AD10	Analog to Digital input 10	IO10	Digital IO 10
AD11	Analog to Digital input 11	IO11	Digital IO 11
		IO12	Digital IO 12
PW0	Power ouput 0	IO13	Digital IO 13
PW1	Power ouput 1	IO14	Digital IO 14
PW2	Power ouput 2	IO15	Digital IO 15
PW3	Power ouput 3		
PW4	Power ouput 4	DA0	Analog output 0
PW5	Power ouput 5	DA1	Analog output 1
		DA2	Analog output 2
CAN_H	Can Bus high	DA3	Analog output 3
CAN_L	Can Bus low	DA4	Analog output 4
I2C_SDA	I2C bus SDA	DA5	Analog output 5
I2C_SCL	I2C bus SCL	DA6	Analog output 6
		DA7	Analog output 7

2.2 KoreIO Connections

The KoreIO requires a power supply connection and an external interface connection. The main external power supply may be necessary for power outputs and the connection is detailled in section 2.2.5. Another supply may be necessary for the electonics on the board. This supply is provided from the KoreBot or the Koala when the KoreIO is used as an extension, but it is required for standalone use.

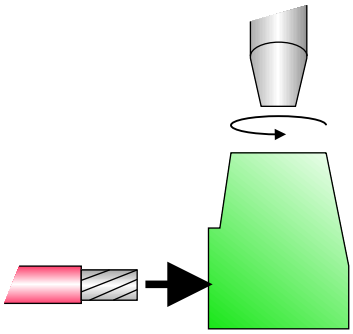


Figure 2.4: Block Connections

The external interface depends on the chosen dip switch settings. The possibilities are:

- KoreBot connection for I2C or KNet 2.0 interface
- Koala connection for KNet interface
- Standalone serial connection for both RS232 interface
- Custom connection for a standalone I2C interface

2.2.1 Standalone Serial Connection

The standalone serial connection is used for both serial protocol mode. A KoreConnect option, which provides the RS232 transceiver, is necessary to use the KoreIO in standalone serial mode. A custom made RS232 transceiver can be used as well, schematics are available on request at support@k-team.com. The serial cable can then be connected to any PC. A 5V supply voltage is required for the board electronic components (see section 2.2.1).

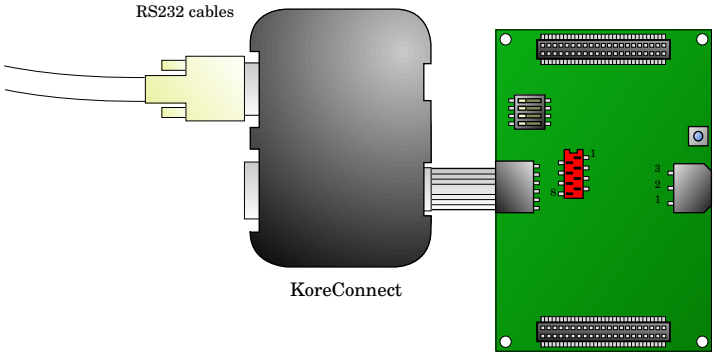


Figure 2.5: Standalone serial connection

How to Supply 5V for Electronics

The 5V supply for the electronics on the board can be derivated from the power output supply using the embedded 5V regulator. The regulator must be activated by adding a 0 ohm bridge on the KoreIO as displayed on figure 2.6. Once activated the only required supply is the main power suppy (section 2.2.5).

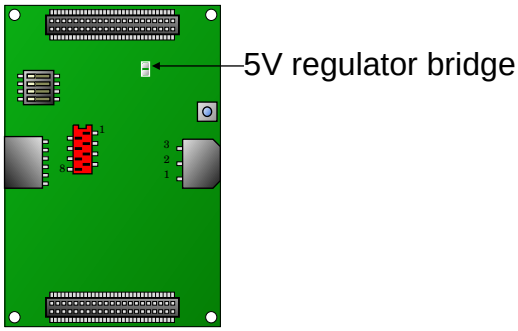


Figure 2.6: 5V regulator bridge

2.2.2 KoreBot Connections

The KoreBot connection is pretty straightforward, the boards should be simply stacked together. The KoreBot will provide the electonics 5V supply, and only the power ouput supply should be added if necessary.

Dip switch setting should be standalone I2C to use the I2C bus for communications between the KoreBot and the KoreIO.

2.2.3 Koala Connections

The Koala must be connected to the KoreIO using the red KNet connector (see section 2.1). A cable should be provided with the KoreIO to be connected with the Koala K-Bus connector (please refer to the Koala user manual for more details).

The electronics supply is provided from the Koala, the power output supply can be connected to an external source or to the Koala 12V outputs.

Dip switches should be set to the regular KNet mode, the Koala does not support the KNet 2.0 protocol.

2.2.4 Standalone I2C Connections

The KoreIO can be connected to an external I2C bus and controlled as a standard I2C devices (see section 2.1.1 for details). Three signals (SDA,

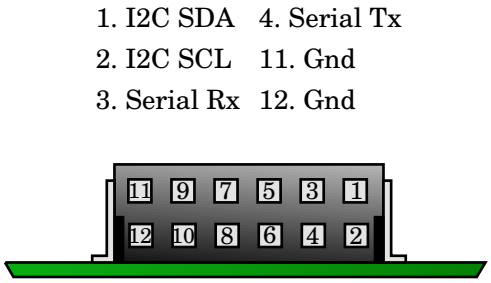


Figure 2.7: KoreIO serial connector and I2C connections

SCL and GND) are necessary to connect the I2C bus, they are displayed on figure 2.7 with the relevant connections on the serial connector.

The electronics supply must be provided separately, and the dip switches should be set to standalone I2C mode.

2.2.5 Power Output Supply Connector

The supply is common for all power outputs thus they will provide the same voltage, and the maximum current per channel is 1A. For instance to provide 12V on each power output, the board should be connected to a 12V DC supply. The overall current that can be supplied by all the power outputs is of course limited by the external power supply capacity.

The main power connector will supply all the outputs, the voltage should match the application requirements, and stay within the accepted voltage range for the transistors. The connector pinout is detailed on figure 2.8.

Voltage range	0-20V
Max global current	5A
Max current per channel	1A

2.3 Hardware Protection

2.3.1 Power Output Fuses

The power outputs are protected from overcurrent with dedicated fuses. Each channel is protected with a 1 Amps fuse, and the board is protected with a general 5 Amps fuse. Higher peak currents are normaly supported by the board and all its components, but continuous operations should respect these limits.

Please contact your K-Team dealer for support if a fuse replacement is required.

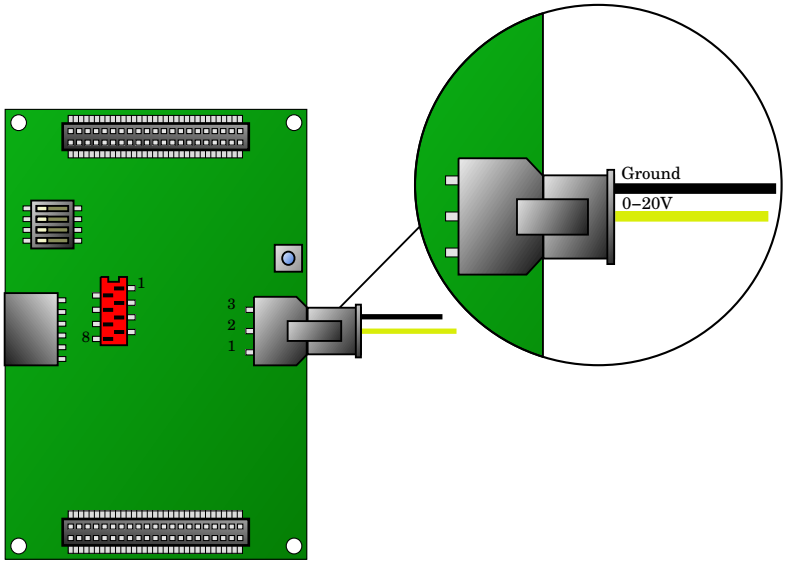


Figure 2.8: Power output supply connection

3.1 Digital Inputs and Outputs

The KoreIO board provides 16 digital IO, that can be individualy configured as outputs, inputs or PWM outputs. Each channel must be configured before being used, and the initial default configuration for all channels is as input.

3.1.1 Input Mode

This is the default mode for all channels. In this mode, a channel state can be read at any time. A zero will be returned if the applied voltage on this input is under 4.5V, and a one will be returned if the voltage is above this level. The board microcontroller might be damaged if a voltage over 5.5V is applied on any input pin.

Voltage threshold	4.5V
Maximum voltage	5.5V
Minimum voltage	0V

3.1.2 Output Mode

In this mode the voltage of any channel can be driven to 5V or forced to ground. The channel are TTL signals and cannot be used to supply power to external devices. The state of each channel can be reset to zero, set to one, or switched (refer to section 4.4).

High voltage	5V
Low voltage	0V

LED State Control

One of the LED on the KoreIO is user controlled. The LED state can be changed using the virtual IO 16, which is always configured as an ouput. This virtual IO can be accessed as any other IO, using the ouput control commands.

3.1.3 PWM Output Mode

Each channel can be configured to be used as a software emulated PWM generator. All the PWM mode channels will share the same frequency but the duty cycle can be set separately for each channel.

The signal frequency in Hertz can be changed by setting a 16 bit register, the minimum supported frequency is 20Hz and the maximum is 5000Hz. High frequencies might become less accurate because of the microcontroller limited computation speed.

The duty cycle ratio for each channel can be set from 1% to 100% with a 1% resolution. The duty cycle ratio value should be directly written to the corresponding 8bit register.

Max Frequency	5000 Hz
Min Frequency	20Hz
Resolution	1%

Servo Motor Control Mode

PWM channels can be configured in a special mode to control servo motors. If the PWM frequency is set to zero, the servo control mode will be enabled for all channels. The PWM frequency is forced to 50Hz for all channels, and the duty cycle ratio can then be set between 1% and 10% with a 0.04% resolution.

		period	**register value**
PWM frequency	50 Hz	20ms	NA
Min ratio	1%	0.2ms	1
Max ratio	10%	2ms	255

3.2 Analog Inputs

The KoreIO provides 12 analog inputs that are connected to a Analog to Digital converter. Each channel is connected to a 10bit DAC, and the values are timestamped to reconstitute the acquired signal properly. The timestamp is a 32bit value wich gives the number of milliseconds since the last timer restart.

The analog measurements are buffered and refreshed sequentialy every millisecond. For any given channel, a new measurement is available every 8 ms.

The maximum voltage which can be applied on a given input is 5V, any voltage over 5.5V applied on any input can seriously damage the microcontoller.

Max Voltage	5V
Min Voltage	0V
Resolution	0.005V

3.2.1 Analog Measurement Timer

Analog measurements are always timestamped using a dedicated timer 32 bit value. The timestamp give the elapsed time in mS since the last timer restart. The timer can be stopped, started and reset from the application.

3.3 Analog Outputs

The KoreIO board provide 8 analog ouputs using a 8bit Digital to Analog converter. Values for these outputs can be set with the corresponding 8bit register. The minimum ouput voltage is $-5.03V$ and the maximum output is $4.39V$, so that the resolution is $0.035V$.

Max Voltage	4.39V
Min Voltage	-5.03V
Resolution	0.035V

The ouput voltage is set to zero when 139 is written in the corresponding register. Figure 3.1 shows the ouput value for a given register value.

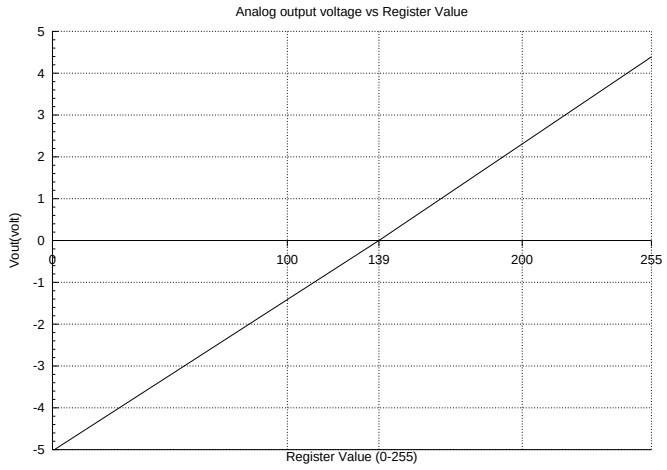


Figure 3.1: Analog output diagram

3.3.1 How To Shutdown the DAC Controller

The Digital to Analog Converter can be shutdown to lower the board power consumption. The DAC activity can be controlled using the analog virtual channel number 8. It is active when the virtual channel value is set to 1, and shutdown when the virtual channel value is reset to 0. Power consumption of the shutdown DAC is nearly null. The converter is active at startup by default.

3.4 Power Outputs

The KoreIO provides 6 power outputs. These outputs are digital and are controlling a transistor to drive up to 1A per channel, as displayed on figure 3.2.

Each channel can be switched on to supply an external device with the KoreIO supply as connected on the power output supply (see section 2.2.5). The channel can be switched off as well to become the equivalent of an open circuit.

A load should be connected between the KoreIO supply, which means any VCC signal on the connector blocks, and a PW channel. If the load is non reversible, the negative side should be connected to the PW channel and the positive side to the VCC signal.

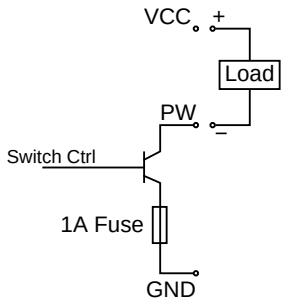


Figure 3.2: Power output diagram

3.5 I2C Bus Controller

The KoreIO can receive commands from an I2C bus (see section 2.2.4), but it can act as an I2C master at the same time using its secondary I2C controller. Figure 3.3 displays the connections between the KoreIO and a secondary I2C bus. With such a configuration, I2C devices can be controlled from a KoreIO receiving serial commands from a PC or KNet commands from a Koala. The KoreIO provides routines to read and write on the I2C bus, and a function to scan the bus for responding peripherals.

The write routine will use a device address, a register address and a variable amount of data bytes. All the data bytes will be written to the same device and register address one by one. Depending on the running mode, the maximum amount of data bytes can vary, refer to section 4.4 for details.

The read routine will use a device address, a register address and a data length parameter. The given number of bytes will be read from the given device at the given register address. The device must support multiple byte

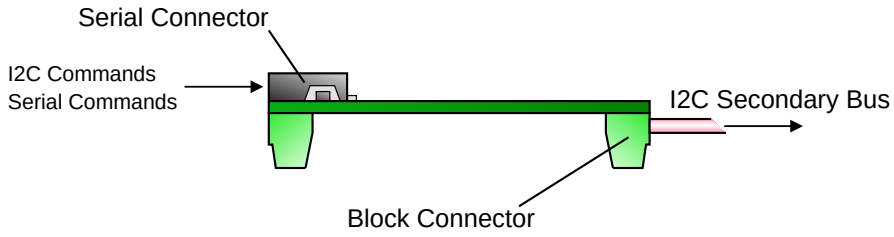


Figure 3.3: I2C Secondary Bus Connection

sequential read from the same register address to enable multiple bytes reading. If the device do not support multiple bytes read, inconsistent behavior will occur when data length is longer than one.

The scan routine can scan the bus between a given start address and a given last address. To scan the bus, the KoreIO controller will simply write each address on the bus and wait for the Acknowledge bit. A device is considered present if the acknowledge bit is set. The routine will return the number of answering device and the list of corresponding addresses.

3.6 CAN Bus Controller

The KoreIO can read and write frames on a CAN bus, using a dedicated CAN bus controller. This document's pupose is not to explain in details the CAN bus protocol, please refer to more specific documentation if required.

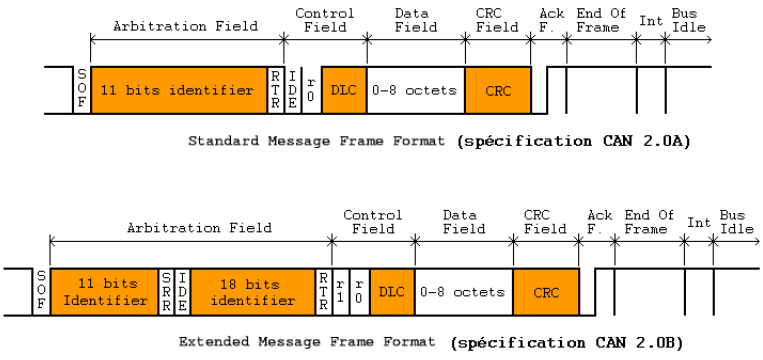


Figure 3.4: I2C Secondary Bus Connection

Figure 3.4 shows a CAN frame structure and the various fields within a frame. Any message sent on the bus must be compliant with this specifi-

cation. The KoreIO controller will handle arbitration, error detection, bit stuffing, and other bus related issues. The application should simply set the frame fields properly before sending messages on the bus.

SOF : The Start Of Frame bit indicates a new frame.

Identifier : The 11bit message identifier. This field sets priority for arbitration. The highest priority is identifier 0.

RTR : The Remote Transmission Request bit specifies if the frame is a data frame or a message request frame.

IDE : Specify if the frame format is standard or extended.

DLC : The four Data Length Code bits specify the number of data bytes in the frame.

Data Field : This field contains between 0 and 8 data bytes.

CRC : 15 bits Cyclic Redundancy Code which is checked by receiver to generate a CRC error.

ACK : Acknowledge bits.

EOF : The seven End Of Frame bits must be set to identify the end of a frame.

Before sending a CAN message, the identifier, data length, control bits and data bytes should be set properly. The serial command 'H' will set all the fields and then send a message, returning a CAN bus status. To send a message using I2C control, several registers must be writen before sending the message. The Data Length Control register must be writen last to trigger the message transmission.

Some registers, such as the canData register or the canIdentifier register, can be written several times to store several bytes. For instance to send a four byte long frame, each data byte should be written sequentially in the canData register.

3.6.1 CAN control bits

The CAN control bits should be set before sending a message on the bus.

bit0 (RTR): 0:Data frame 1:Message request frame

bit1 (EXT): 0:Standard format 1:Extended format

bit2-3: Set priority of the message from 0 to 3. 0 being the highest priority. The defautl priority is 1.

The KoreIO controller will read messages on the CAN bus when available and keep them in an internal buffer for the application to read them. When using serial commands, the first message in the buffer will be returned. When using I2C control, the application can check the number of available messages and must clear each message before reading the next one. Using the KoreIO internal timer, the application can also check the moment when the message was received.

Some I2C registers can be used to read more than a single byte. For instance all the CAN data bytes must be read from the canData register. All the bytes must be read using a multiple byte read and *not* using several sequential single byte read. If the I2C master does not support multiple byte read, it will not be able to read all the data.

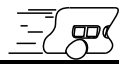
3.6.2 CAN status bits

The CAN status byte indicates the following:

bit0: Bus overflow

bit4: Hardware buffer full

bit6: 0:Standard format 1:Extended format

4 Serial Communication Protocol 

4.1 Generic User Syntax

The KoreIO supports various high level serial commands to control the board from a host Personnal Computer. The serial command syntax is similar to the Khepera syntax with some enhancements.

Each command is a letter eventually followed by numerical arguments. Arguments format is either 8bit (default), 8bit hexadecimal, or 32bit integer. The format for each argument is given in the command description (ie value.format), but is 8bit by default. A prefix must be added to an argument with a specific format as described in the following table:

32bit integer	l
16bit integer	d
8bit hexadecimal	0x

This syntax is designed for an easier human interface, when the board is controlled manually from a terminal. Programs can use the *Program Syntax mode* (see section 2.1.1) for a faster interface to the KoreIO.
Some user command syntax examples:

```
Read Analog channel 1:  A,1

Set PWM frequency:  F,d400

Set the Digital ouput 5: O,5,1
```

4.2 Read Configuration Format

The digital io configuration read command returns four bytes that describe the current IO modes. Each IO mode is coded with 2 bits using the following table:

00	input mode
01	ouput mode
10	pwm mode
11	n.a.

Each of the four returned bytes contain four IO configuration code:

bit	7 6	5 4	3 2	1 0
io1	IO3	IO2	IO1	IO0
io2	IO7	IO6	IO5	IO4
io3	IO11	IO10	IO9	IO8
io4	IO15	IO14	IO13	IO12

4.3 Error Code Return

Most commands will return a status code that will take special values in case of error. Some commands have different status codes which are specified in the command description. The standard error codes are:

0	Success
2	IO not properly configured
3	Wrong Channel specified
4	Wrong Mode or State specified

4.4 Generic Program Syntax

As it is easier for a program to handle raw data transaction, and as the method is globaly faster, a binary serial protocol is also supported. This protocol does not use comma nor format prefix and all data bytes must be transfered sequentially.

The byte sequence to send for each command is the same which is sent for serial mode, except that no coma nor end of line character should be added and that the first byte of the sequence should be the number of bytes in the command. The command sequences must be send without any charaters between each command.

For example, to send the "B" command, the following sequence should be sent:

```
1B
```

And the module will answer:

```
3b10
```

The command "C,0,2" will be formated as:

```
3C02
```

and the answer will be something like:

```
2c0
```

A Register Summary

A.1 Master Write Registers

Address	Valid Data	Description
0x01	0-100	Change PWM channel 0 ratio
0x02		Change PWM channel 0 ratio
0x03		Change PWM channel 0 ratio
0x04		Change PWM channel 0 ratio
0x05		Change PWM channel 0 ratio
0x06		Change PWM channel 0 ratio
0x07		Change PWM channel 0 ratio
0x08		Change PWM channel 0 ratio
0x09		Change PWM channel 0 ratio
0x0A		Change PWM channel 0 ratio
0x0B		Change PWM channel 0 ratio
0x0C		Change PWM channel 0 ratio
0x0D		Change PWM channel 0 ratio
0x0E		Change PWM channel 0 ratio
0x0F		Change PWM channel 0 ratio
0x10		Change PWM channel 0 ratio
0x11	0-255	Change the Analog output 0
0x12		Change the Analog output 1
0x13		Change the Analog output 2
0x14		Change the Analog output 3
0x15		Change the Analog output 4
0x16		Change the Analog output 5
0x17		Change the Analog output 6
0x18		Change the Analog output 7
0x19	0-15	Configure the given digital IO to input mode
0x1A	0-15	Configure the given digital IO to ouput mode
0x1B	0-15	Configure the given digital IO to pwm mode
0x1C	0-15	Reset to zero the given digital IO
0x1D	0-15	Set to one the given digital IO
0x1E	0-15	Change the state of the given digital IO

Address	Valid Data	Description
0x1F 0x20	0-255 0-255	Change the PWM frequency LSB Change the PWM frequency MSB
0x22	0-5	Reset to zero the given Power output
0x23	0-5	Set to one the given Power output
0x24	0-5	Change the state of the given Power output
0x25	0-2	Analog measurement timer ctrl 0: reset the timer 1: stop the timer 2: start the timer
0x26	0-255	Set the canIdentifier. Each byte of the CAN identifier must be writen sequentially starting with the Most Significant Byte. 4 bytes should be written for extended id, and only 2 bytes for standard id.
0x27	0-255	Set the canData to send. Each data byte must be written sequentially in this register.
0x28	0-8	DLC sets the number of bytes in the message then send the message.
0x29	0-255	Set the CAN control bits
0x32	0-255	Write the given byte on the I2C busr, using the DeviceWriteAddress (0x38) and the RegisterWriteAddress(0x39)
0x33	0-255	Read the given register one or more times. The device to read is the DeviceReadAddress(0x34) and Nbytes(0x35) sets the number of times it will be read. See the master read table to get the returned values.
0x34	0-255	DeviceReadAddress. Set the device address for I2C bus read.
0x35	0-4	Nbytes. Set the number of times to read a given I2C register
0x37	-	Start the I2C bus starting from address 0 to address 128. See the master read table to get the returned value.
0x38	0-255	DeviceWriteAddress. Set the device address for I2C bus write.
0x39	0-255	RegisterWriteAddress. Set the register write address for I2C bus write.

A.2 Master Read Registers

Address	Answer	Description
0x00	0-255	Read Firmware Version
0x01	0-1	Read Digital Input 0
0x02		Read Digital Input 1
0x03		Read Digital Input 2
0x04		Read Digital Input 3
0x05		Read Digital Input 4
0x06		Read Digital Input 5
0x07		Read Digital Input 6
0x08		Read Digital Input 7
0x09		Read Digital Input 8
0x0A		Read Digital Input 9
0x0B		Read Digital Input 10
0x0C		Read Digital Input 11
0x0D		Read Digital Input 12
0x0E		Read Digital Input 13
0x0F		Read Digital Input 14
0x10		Read Digital Input 15
0x11	6 bytes	Read Analog Input 0
0x12	refer to	Read Analog Input 1
0x13	section 3.2	Read Analog Input 2
0x14		Read Analog Input 3
0x15		Read Analog Input 4
0x16		Read Analog Input 5
0x17		Read Analog Input 6
0x18		Read Analog Input 7
0x19		Read Analog Input 8
0x1A		Read Analog Input 9
0x1B		Read Analog Input 10
0x1C		Read Analog Input 11
0x20	0-20	Number of available CAN messages.
0x21	0-255	Read the canIdentifier (4 bytes).
0x22	0-8	Read the number of data bytes.
0x23	0-255	Read the data bytes (1-8 bytes).
0x24	0-255	Read the CAN status byte.
0x25	0-255	Read the message timing (4bytes).
0x26	0-4	Read this register to clear the current CAN message and access the next one.

0x33	0-255	Read the returned value for the first read on the I2C bus.
0x34	0-255	Read the returned value for the second *sequential* read on the I2C bus.
0x35	0-255	Read the returned value for the third *sequential* read on the I2C bus.
0x36	0-255	Read the returned value for the fourth *sequential* read on the I2C bus.
0x37	0-255	Read the number the of device found during the I2C bus scan.
0x38	0-255	Read the list of addresses found during a scan. Refer to section 3.5. All the addresses must be read with a single i2c read.

B Serial Commands Summary

A Read Analog Input

Command:	A, channel
Answer:	a, value.16, time.32
Effect:	Read the current voltage level on the given analog input. The 10bit value from the ADC is returned. The time gives the 32bit amount of milliseconds since the last timer restart. If an invalid channel is given the returned value will be set to 0xFFFF.

B Read Software Version

Command:	B
Answer:	b, Version, Revision
Effect:	Give the software version for the KoreIO firmware.

C Configure IO

Command:	C, channel, mode
Answer:	c, status
Effect:	Configure the given IO channel. The mode can be set to 0: input, 1: ouput, or 2: pwm.

D Read the IO Configuration

Command:	D
Answer:	d, 0xio1, 0xio2, 0xio3, 0xio4
Effect:	Read the configuration for all the digital IO. Please refer to section 4.2 for details about the format.

E Unused

F Change PWM Frequency

Command:	F, frequency.16
Answer:	f, status
Effect:	Change the 16bit common frequency for PWM channels. Any frequency over 5000 Hz will be limited to 5000. Any frequency under 20Hz will be set to 20. If the frequency is set to 0 the controller will switch to the Servo Motor control mode (see section 3.1.3).

G Change PWM Ratio

Command:	G, channel, ratio
Answer:	G, status
Effect:	Change the given PWM channel duty cycle ratio. The ratio can be set to any value between 0 and 100. When using the servo motor control mode, ratio setting can take any value between 0 and 255 (see section 3.1.3).

H Write CAN Message

Command:	H, identifier.32, dataLength, data0, data1, ..., dataN, control
Answer:	h, status
Effect:	Send a message on the CAN bus. The control bits are defined in section 3.6.

I Read Digital IO

Command:	I, channel
Answer:	i, state
Effect:	Read the given digital IO state.

J Read CAN Message

Command:	J
Answer:	j, status, id.32, dataLength, data0, data1, ..., dataN, canStatus
Effect:	Read a message from the CAN bus. If no message is available, the status is set to 1 and the other bytes are set to 0. If a message is available, the status is reset. The canStatus byte is described in section 3.6.

K Unused

L Unused

M Unused

N Unused

O Set Digital IO

Command:	O, channel, state
Answer:	o, status
Effect:	Change the given digital output state. State can be set to 0: clear IO, 1: set IO, 2: change IO.

P Set Power Output

Command:	P, channel, state
Answer:	p, status
Effect:	Change the given Power output state. State can be set to 0: clear IO, 1: set IO, 2: change IO.

Q Unused

R Read I2C Bus

Command:	R, address, register, dataLength
Answer:	r, data0, data1, ..., dataN
Effect:	Read *dataLength* bytes on the I2C bus, all bytes are read from the same device address, and the same register. The I2C device must support multiple read to allow data length longer than 1. The maximum data length for KNet mode is 4 bytes, the maximum data length for serial mode is 20.

S Set Analog Output

Command:	S, channel, value
Answer:	s, status
Effect:	Change the given analog output value. Refer to section 3.3 for details about the valid value range. Virtual channel 8 can be used to shutdown and wake-up the DAC controller. The controller is active when the channel 8 value is set to 1, and is shutdown when the channel 8 value is reset to 0.

T Analog Measurement Timer

Command:	T, mode
Answer:	t, status
Effect:	Control the Analog measurement timer. The mode can be set to 0: reset timer, 1: stop timer, 2: start timer.

U Scan I2C Bus

Command:	U, firstAddr, lastAddr
Answer:	u, nDevice, 0xdeviceAddr0, ..., 0xdeviceAddrN
Effect:	Scan the I2C bus from firstAddr to lastAddr. The number of answering devices is returned as well as all the device adresses.

V Unused

W Write I2C Bus

Command:	W, address, register, data0, ..., dataN
Answer:	w, status
Effect:	Write the given data to the I2C bus using the given device address and register. All the bytes are writen to the same address, one by one. Maximum 20 bytes can be writen with the same command. Only one byte at a time can be writen in KNet mode.

X Unused

Y Unused

Z Unused