µChameleon 2 User's Manual

Firmware Rev 4.0

1. General overview

1.1. Features summary

- Full-speed USB 2.0 communication bandwidth: 1Mbytes/s
- Drivers available for Windows 98/Me/2000/XP/Vista, Linux, MacOS
- USB CDC class communications
- 8 analog inputs with 12 bits of resolution
- 18 general purpose digital I/Os
- 4 timer channels with pulse width modulation
- 4 timer channels can also measure frequency and pulse width
- Embedded command interpreter
- Autonomous operation
- Powers from USB, or wall-mount transformer, switches automatically
- Screw connectors to quickly connect your external circuits
- Compact size

1.2. USB CDC communication drivers

The µChameleon 2 uses the USB CDC class to communicate with its host PC. What this means is that most operating systems can use built-in drivers, a situation similar to what happens with USB keys. The µChameleon 2 will appear like a legacy serial com port.
Compared to the first generation µChameleon, this will lead to an easier installation, and more robust and consistent performance, while maintaining the ease of use that came form virtual serial port communications.
Most applications written for the µChameleon 1 will work with the µChameleon 2, with little or no re-writing.
If your programming environnement doesn’t natively support serial ports, standard system libraries can be used to open and talk to the serial port, using the same techniques that would be used with legacy com ports.

1.3. Command interpreter

The firmware inside your µChameleon runs a command interpreter that understands full-text commands that let you access all of its hardware resources. The small command set was optimized to provide what is necessary for addressing real-world applications. It hides all the messy details so you can focus on end application, and makes for a short learning curve. No more complicated API that is difficult to learn, with lots of mandatory parameters and subtle and hard to distinguish variants.

1.4. Firmware upgrades

We constantly work to improve the possibilities of the µChameleon, and provide new features following customer input. New firmware can be downloaded from our web site, and with our simple Firmware Upgrader PC software, the newest features are just a click away. Additionally, the hardware protected boot block cannot be erased accidentally, meaning you cannot end up with a locked µChameleon. If something fails during upgrade, simply try again.

1.5. External connectors

The external connectors provide 18 I/O pins. All of them are individually programmable as digital inputs or outputs, and their state can be read or set. Additionally, some of them have specialized functions for use as analog inputs, analog outputs, frequency generation, pulse width modulation…
These special purpose pins are indicated on the label of your µChameleon as a quick reference when connecting to external devices. Here is a summary of these special function pins:

Special function	Applicable pin numbers
digital i/o	all pins
input pull-ups	9 to 16
analog inputs	1 to 8
analog outputs	9 to 12
pwm outputs	9 to 12
timers	9 to 12
spi	13 to 16
uart	17-18

1. Getting started

1.6. Device driver installation

When you first connect your µChameleon to your computer, you may be prompted for a driver, or a windows information file, so the system knows wich driver to use. Please insert the installation disk provided with the µChameleon, and choose the appropriate directory where the driver is located.

1.7. Test application: µChameleon control

1.7.1. Overview

On the install CD you received with your µChameleon 2 (also available as a download on our web site) you will find an utility called "µChameleon Control", that lets you perform simple tests with a graphical user interface, and that allows you to exert all the features of the µChameleon, as well as show you it is properly connected (if multiple µChameleons are connected to your computer, you will be able to select the one you want to talk to).

Another interesting feature is that for every action you perform in the user interface, the software shows you the command string that is sent to the µChameleon (along with the response if applicable) so you can learn along the process, making for a very short learning curve. In a few minutes, you'll understand how to perform most of the tasks in your own applications.

Additionally, an entry box will let you type random commands that will be sent as is to the µChameleon, again showing the eventual answer.

To install this software, simply launch the "setup.exe" in the "µChameleon Control" under "Utilities" directory of the install CD, or grab it on our "downloads" page on our web site.

This software full source code is available on the install CD and web site.

1.7.2. Automatic device detection

The frame called “Device selection” will show all µChameleon devices connected to your computer, and allow the selection of the device you want to talk to. Of course, if a single device is detected, it will be the default selection.

1.7.3. Activity led toggling

Two buttons, labelled “led on” and “led off” will turn on and off the activity led beside the usb connector of the µChameleon. This is one of the simplest things you can do with it. Additionally, you can use an 8bits word as a sequence for flashing effects, quickly showing a simple state information, without looking at the computer screen. Try typing the following command : “led pattern 5” (and press enter). You should see the led moslty off, with 2 brief flashes. To revert to default state, send: “led pattern 254”.

1.7.4. Digital I/Os panel

This panel shows a representation of the µChameleon, where every connector has two clickable items. One is a square box with a letter I or O, and sets the direction on the pin, as input or output, respectively. The other mimics a led, and will show the state of the pin. When programmed as an input, it will be light or dark green, corresponding to the high and low state, respectively. When programmed as an output, the colours will be light and dark red, indicating a high or low output.

1.7.5. Direct command input

This entry box allows you to directly type commands with your keyboard, and send it to your µChameleon. The command is displayed in the logging text box, as well as the answer, if applicable.
Check boxes allow you to select the type of string termination used.

2. Programming reference

This section describes all commands supported by the µChameleon firmware. It is possible to try issuing commands, as a learning exercise, by using the "µChameleon Control" application. Actions in the graphical interface will also show (in the log text) the command corresponding to each action, and that was actually sent. It is also possible to type commands by hand, in the 'direct command input' box.
Note: In the rest of this manual, when we talk about sending a command, we mean that the corresponding string is sent, followed by a LineFeed, or CarriageReturn, or Cr-Lf combination.

Note: Commands to be sent will be shown in italics, like this: *led on,* as well as replies.

The full source of the "µChameleon Control" being available, it is a great place to start learning how to interact with the µChameleon, using proven code.
It can be used as a basis for your own applications, so feel free to cut/paste code from it.

1.8. Communication basics

The µChameleon USB CDC class will create a 'Virtual Com Port', that means that everything happens like if you where talking to a device connected to a legacy serial port on your pc. Most programming environments will support that feature, either through built-in functions, or the win32 api.
Although most programming examples will be provided using 'Visual Basic' style, it is generally straightforward to translate them to other programming languages.

1.8.1. Opening communication

Before actually sending commands, it's necessary to open the communications port, and this will depend on your programming environment, but in Visual Basic, would simply be:

```
MSComm1.PortOpen = True
```

1.8.2. Checking for device presence

Although it is not necessary, you might want to check if your µChameleon is functioning properly and ready to accept commands. It can be done by sending *id*, which the device should respond to by returning the two words : *id µChameleon2*.

1.8.3. Checking for firmware version

It is possible to check the version of the firmware currently present in the µChameleon with the following command:

firmware -> *firmware 4.0*

1.9. Activity led

Besides the USB connector of the µChameleon, there is a led that turns on at power-up, with a small off flash, indicating the firmware is up and running, and waiting to receive commands. It is also possible to act on this led by software.

1.9.1. Turning the led on or off

Turing the led on:
led on
led 1

Turn the led off:
led off
led 0

1.9.2. Setting a led flashing pattern

led pattern <n>

The led can be driven by a sequence of 8 'on' and 'off' states, each state corresponding to the state of a bit in the parameter byte. For example, if you want the led to be mostly off, with 3 small flashes, the parameter value can be: 21 (1 + 4 + 16).

Try by sending: *led pattern 21*

1.10. Digital inputs – outputs

1.10.1. Setting pin direction

Setting the nth pin as an input:

pin <n> input
pin <n> in

Setting the nth pin as an output:

pin <n> output
pin <n> out

1.10.2. Reading pin state

Reading the nth pin state:

pin <n> state -> *pin <n> 0 | 1*

1.10.3. Setting pin state

Setting the nth pin high:

pin <n> high
pin <n> hi

Setting the nth pin low:

pin <n> low
pin <n> lo

1.10.4. Activating pin pull-up

Activating pull-up on pin n:

pin <n> pullup on
pin <n> pullup 1

Deactivating pull-up on pin n:

pin <n> pullup off
pin <n> pullup 0

Note: The typical pull-up resistor value is around 47kOhms.

1.10.5. Monitoring pin activity

This is an alternative method to monitor pin states without having to use a timer in your pc application to periodically poll for the state of pins.

Activating pin monitoring:

pin <n> monitor on | *1*
pin <n> mon on | *1*

Deactivating pin monitoring :

pin <n> monitor off | *0*
pin <n> mon off | *0*

When pin monitoring is active, the µChameleon will constantly check for transitions on the selected pin, and report once per transition by sending the same string that would be returned by the *pin <n> state* command.

For example, if you when to monitor pin 3, you will send:
pin 3 monitor on
and whenever a low to high transition will be detected, you will receive:
pin 3 1
or when a high to low transition happens, you will receive:
pin 3 0

Pin monitoring is supported by all 18 pins, and can be active simultaneously on any pins combination.

1.11. Analog inputs

1.11.1. Reading pin voltage

Read voltage on pin n:

adc <n> -> *adc <n> <v>*

Pins 1 to 8 support the analog to digital conversion of a 0-5volts input to a 12 bit number. The firmware always responds by repeating the *adc <n>,* this allows easy and unambiguous demultiplexing, without waiting for answers each time a command is sent. The <v> value will be in the range [0;4095] with 0 for 0volts and 4095 for 5volts.

1.12. Analog outputs - PWM - Frequency generation

1.12.1. PWM applications

The four timer channels of your µChameleon can be used to output signals of controlled frequency and duty cycle, and each of them can be used for a variety of purposes, including generation of analog voltages.

A simple R-C filter will generate a clean, linearly varying analog voltage, and by changing this voltage at regular intervals, it is also possible to generate arbitrary waveforms.

This makes it very simple to generate a programmable control voltage, for example, the output voltage of a programmable power supply, the frequency or a vco, in short, anything that can be controlled with an analog voltage.

Is is also possible to use the pwm outputs without any filtering, the pwm output connected directly, for example to control the brightness of a led, of driving a power transistor or bridge, controlling the speed and direction of a dc motor.

1.12.2. PWM commands summary

Here is a summary of the available commands:

pwm <n> on
pwm <n> off
pwm <n> period
pwm <n> width
pwm <n> polarity
pwm <n> prescaler
pwm <n> counter

The following sections give details on using these commands.
Note: these features are available on pins 9-10-11-12.

1.12.3. Pwm channel on or off

Turn on the pwm feature on pin n:

pwm <n> on

Turn off the pwm feature on pin n:

pwm <n> off

Note: these commands can be sent only once at the beginning and end of an application, although it can be a good idea to set various parameters like period and width before turning the pwm on.

1.12.4. Pwm output frequency

Set the signal period of pin n:

pwm <n> period <p>
pwm <n> per <p>

The period parameter is in timer clock cycle units, with a range [0;65535]. Nominal frequency is 24MHz. For example, *pwm 9 period 1000* will program the timer channel on pin 9 for a 24kHz frequency. Note: this is true unless you have prescaled down the clock input – see prescaler section 1.12.7.

1.12.5. Pwm duty cycle

Set the signal duty cycle of pin n:

pwm <n> width <w>
pwm <n> wid <w>

The width parameter is in timer clock cycle units, with a range [0;65535]. For example, if you want to generate a 30% duty cycle signal on pin 9 with a 10kHz frequency, you will send:

```
pwm 9 period 2400
pwm 9 width 720
```

Also, if you want to generate a variable frequency, but with a constant 50% duty cycle, the period being in a variable called myvar, you will send:

```
pwm 9 period myvar
pwm 9 width myvar/2
```

Note: this syntax is a simplification, because myvar and myvar/2 should be sent as string, and because other commands might be necessary, notably turning the pwm channel on if this is the first action.

1.12.6. Pwm polarity

It is possible to control the polarity of the logic level of a pwm channel, which will affect the meaning of the width parameter (a 30% duty cycle would now become 70%).

Normal polarity: (default)

```
pwm <n> polarity 0
pwm <n> pol 0
```

Inverted polarity:

```
pwm <n> polarity 1
pwm <n> pol 1
```

Note: pwm channels 9 and 10, as well as 11 and 12 share their clock, so you can easily generate complementary signals by programming two channels identically, and just inverting the polarity of one of then.

1.12.7. Pwm prescaler

The period and width parameters of the pwm feature are expressed in terms of cycle counts, with a default frequency of 24MHz, that is a resolution just under 42ns (41.667ns). This means that the minimum possible frequency is about 366Hz with no prescaling. It is also possible to prescale the clock of pwm channels by the following divider amounts: 1,2,4,8,16,32,64,128. Default: 1.

```
pwm <n> prescaler <p>
pwm <n> pre <p>
```

1.12.8. Pwm counter

The free-running counter current value can be requested with the following command:

```
pwm <n> counter      ->      pwm <n> counter <c>
pwm <n> cnt
```

1.13. The Wait instruction: timing is everything

One of the requirements when interacting with the real-world is the ability to make events occur at specific times. One way to achieve that on the µChameleon is to use the wait instruction set.
It enables defining actions, like setting a pin hi or low, with precise intervening times, in a way that does not depend on individual instructions execution time. The wait time is defined as an additional time lapse referenced to a specific zero time that is defined once. This is unlike having a pause, that would accumulate small errors due to execution times. Also, times are defined by a free running timer inside one of the four timer channels 9 to 12.

A typical application will consist of an instruction sequence having:

- first, an initialisation instruction to define time zero.
- then, normal instructions alternated with wait instructions.

1.13.1. Creating a time zero reference

The following instruction initialises a timer channel for use as a timing source for the wait instruction, and sets the current timer as a zero reference:

wait init <channel>

1.13.2. Waiting for a specific time

The following instruction inserts a time interval referenced from the previous wait instruction, so an accurately timed sequence of actions can be performed:

wait time <time>

The time parameter is specified in timer counter units.

1.13.3. Wait sequence sample code

For an example that produces 3 impulsions of 1ms, separated by 1ms intervals (one pulse every 2ms), please load the 'wait pulses.txt' demo code provided with the 'µChameleon Control' test application.

1.14. Frequency and pulse duration measurements

All four timer channels on the µChameleon can be used to measure the frequency of an incoming signal, or the duration of a pulse.
The edge detection logic can be programmed to start measurement on a rising (low to high - default) on falling (high to low) edge of the incoming signal.
Default resolution for these measurements is 42ns, but that depends on the timer prescalers, so this can be changed with the pwm prescaler commands.

1.14.1. Measuring a full cycle length

The following instruction will wait for a transition of the input signal, start counting using the hardware in the timer channels, and stop on the same signal transition. The time interval is then reported.

timer <pin> capture cycle

The µchameleon responds with:

timer <pin> capture cycle <count>

1.14.2. Measuring a single pulse length

The following instruction will wait for a transition of the input signal, start counting using the hardware in the timer channels, and stop on the opposite polarity transition. The time interval is then reported.

timer <pin> capture pulse

The µchameleon responds with:

timer <pin> capture pulse <count>

1.14.3. Changing edge sensitivity

The following instruction sets the sensitivity of the capture instruction, that is the edge that starts the capture feature:

timer <pin> capture edge <polarity>

0 is negative going edge, 1 is positive (low to high - default).

1.15. SPI – Serial Peripheral Interface

1.15.1. Introduction

The SPI feature of the µChameleon implements the three wire synchronous communication found on many peripheral chips. It is compatible with most devices on the market. The µChameleon will always work as the master SPI port.

Synchronous serial port signal are assigned as follows:

pin 13 : SCK	(Serial Clock)
pin 14 : MOSI	(Master Out Slave In – TX)
pin 15 : MISO	(Master In Slave Out – RX)

1.15.2. Turning the SPI on and off

Before being able to use the SPI, it should be turned on because the pins are shared with regular I/O pins 13, 14 and 15.

Syntax: *spi on*
spi off

Turning the spi of will revert the associated pin to their general purpose I/O function.

1.15.3. Setting clock rate

The clock rate of the SPI can be varied with a prescaler that divides the master clock (24MHz) of the µChameleon, by values of 2, 4, 8, 16, 32, 64, 128, or 256.
Divide by 2 is the default, providing 12Mbit/s transmission.

Syntax for setting the prescaler:

spi prescaler <p>
spi pre <p>

Example:

spi prescaler 32 (set the clock generator to 750kHz)

1.15.4. Setting clock phase and polarity

To adapt to the variety of peripheral devices one can have to connect to the µChameleon, it is possible to set the polarity and phase of the SPI with the following commands:

Normal polarity (normaly low, active high - default)

spi polarity 0
spi pol 0

Inverted polarity (normally high, active low)

spi polarity 1
spi pol 1

Normal phase (first clock edge begins in middle of data bit – default)

spi phase 0
spi pha 0

Inverted phase (clock edge and data change simultaneously)

spi phase 1
spi pha 1

1.15.5. Sending data

Data transmission is initiated with the following command:

Syntax: spi out <byte>

As soon as the command is received, the spi starts shifting the data at the MOSI output and cycling the SCK output.

1.15.6. Receiving data

Data reception is initiated with the following command:

Syntax: spi in -> spi in <byte>

As soon as the command is received, the spi starts cycling the SCK output, shifting the data from the MISO input. After 8 clock cycles, the µChameleon sends the data back with the above syntax.

1.15.7. Setting the dummy byte

Due to the inherently full-duplex nature of the SPI, when receiving data, the MOSI output remains idle, as if it where sending a zero byte.

This default behaviour can be changed with:

spi dummy <byte>

1.15.8. Full-duplex transmit – receive

It is possible to combine a transmit and receive operation with the following command:

spi swap <tx byte> -> *spi in <rx byte>*

When the spi swap command is received by the µChameleon, the spi starts cycling the SCK output, shifting out the data on the MOSI input, and simultaneously shifting in the data form the MISO input.

1.16. UART Support

1.16.1. Introduction

The µChameleon features a hardware uart (universal asynchronous receiver transmitter) that is similar to the legacy serial ports found on many devices, including PCs and instrumentation. Uart pins on the µChameleon use logic level voltages. For voltage levels compatible with RS-232, an external voltage translator should be used (contact us for availability). The 4.0 firmware only supports hardware flow control (DTR/CTS).

1.16.2. Initialisation

Uart is initialised in a fixed mode after reset at a default 9600 bauds, 8 bits, 1 stop bit, no parity, with hardware flow control on pins 16 (output-RTS) and 15 (input-CTS). TX is pin 18, RX is pin 17.

In order to enable the uart, including flow control pins, the following command should be sent:

```
uart on
```

In order to disable the uart, and return its pins, including flow control pins, to general purpose I/O operation, the following command should be sent:

```
uart off
```

1.16.3. Sending data

Data is sent out with:

```
uart send <n>  <…data…>
```

where <n> is a text string representing the number of bytes to send, followed by the actual bytes to send. The presence of the byte count makes it possible to have the data contain not only text strings, but include any 8-bit binary value.

Example

```
uart send 5 hello
```

Note: this command does not require a carriage return of line feed at the end.

1.16.4. Receiving data

To receive data, simply send the "uart receive" command:

uart receive

If then µChameleon has no received bytes in its internal buffer, it will simply answer:

uart receive 0

In the more general case, the µChameleon will respond by returning the "uart receive" string, followed by a space character, then a number indicating the bytes that will follow, then the actual bytes. The format is the same as the send command. The data bytes can contain text or binary data.

Example:

uart receive 5 hello

Note: a termination will be sent after the command, but the byte count will always reflect the number of bytes that were actually received, with no data filtering performed, in order to maintain transparent text/binary compatibility.

1.16.5. Asynchronous Notification

It is sometimes desirable to be notified of incoming data without having to regularly poll the µChameleon with “uart receive” commands.

Turning on the notification feature is done with:

uart notify <bytes>

Anytime the internal buffer has gathered sufficient data (at least a certain byte count threshold), the µChameleon will automatically send data using the “uart receive” format as defined in the previous section.

If necessary, it is possible to revert to explicit polling with:

uart notify 0

1.17. Variables and arithmetic

1.17.1. Dim

The dim statement create a new variable, stores its name and type, and allocates memory for it. The variable name and type are stored permanently in the flash memory, but the value is not retained when power is lost.

Syntax: *dim <varname> as <type>*

Example: *dim myvar as int*

Notes: It is possible to define up to 32 variables. Variable names may have up to 12 characters. Only the 'int' type (32 bit signed integer) is defined in firmware 4.0.

1.17.2. Let

The let statement assigns a value to a variable, the right side of the equal sign being either a single variable or constant, or an arithmetic operation combining two variables or constants.

Syntax:

let <variable> = <value>
let <variable> = <value> <operator> <value>

Examples:

let x = 0
let counter = counter + 3
*let speed = speed * accel*

Available operators:

operator	arithmetic performed
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

1.17.3. Increment and decrement

Instead of writing:	*let var = var + 1*
One can write:	*increment var*
Or:	*incr var*

Instead of writing:	*let var = var - 1*
One can write:	*decrement var*
Or:	*decr var*

This is generally more convenient, results in more compact code, and is also faster.

1.17.4. Print

It is possible to query the value of a variable with the print statement.

Syntax: *print <varname>*

This command will send the variable content to the USB communication port.

1.17.5. The '?' special variable

Some former commands, like 'adc' or 'pin', have their last value stored in a special variable that can be accessed with the '?' sign.

Example:

```
adc 1
let voltage = ?
```

the 'voltage' variable (that we suppose was created previously with 'dim') now contains the result of the last analog to digital conversion.

Note: the '?' sign can be used in any place a variable or constant would be used.

1.17.6. Erasing variables

It is possible to erase all variables, and their definitions, as well as freeing the memory they use with the erase command:

Syntax: *erase dims*

1.18. Conditional execution

1.18.1. If … then and relational operators

If statements provide the ability to conditionally execute commands based on comparison between values.

The general form it takes is the following:

```
if <value A> <operator> <value B> then <command>
```

The value parameters can be any variable or constant. The operator can be any of the following:

operator	comparison performed
=	equal to
<>	not equal to
>	strictly greater than
>=	greater or equal
<	strictly less than
<=	less or equal

Examples:

if counter >= 1024 then let counter = 0
if voltage < 128 then pin 3 low
if alert = 1 then led pattern 5

1.19. Event handlers

1.19.1. Introduction

Event handlers provide a way to store a list of instructions to be executed when a specific event occurs. They are similar to functions or procedures in most programming languages except they don't take parameters.

There are currently two defined events: 'reset' and 'background'.

1.19.2. The reset event

As the name implies, the reset event occurs once at power-up and every time the µChameleon is reset, either via software, by pressing the reset button, or when power is cycled.

1.19.3. The background event

The background event is a periodic event triggered by an internal oscillator running at approximately 20Hz. It can be turned on or off (default after reset is off).

Syntax for turning on the periodic event generator:

background on
back on

Syntax for turning off the periodic event generator:

background off
back off

1.19.4. Background event clock source

It is possible to use the hardware timer pwm generated clocks as sources for the background events, instead of the default internal 20Hz clock.
This way, higher background code frequency, and/or more precise clock generation is possible.
One will start by generating a specified clock frequency using a pwm channel on any of the 4 timer channels as explained in the pwm section, and then linking the background event to that specific channel with the following command:

background clock <n>
back clk <n>

1.19.5. Defining event handlers

Syntax for defining event handlers:

```
onevent <event name>
<instruction line 1>
<instruction line 2>
.
.
.
<instruction line n>
endevent
```

Example:

```
onevent reset
pin 2 output
background on
endevent

onevent background
adc 1
if ? > 135 then pin 2 low
if ? < 126 then pin 2 high
endevent
```

Note: The preceding example show how simple it is to implement a simple standalone temperature controller with hysteresis.

The memory footprint available for event handlers is 1024 bytes for the reset event, and 7168 bytes for the background event.

2. Hardware Information

2.1. Inputs / Outputs

All inputs are very high impedance CMOS inputs (typically greater than 10Mohms.)
All I/Os have current limiting drivers. This enables a direct connection to LEDs, opto-couplers, power transistors, miniature relays, piezo buzzers, small loudspeakers… with a typical current output of 20mA.

2.2. Power supply circuitry

2.2.1. Power circuitry overview

- Powers from USB
- Powers from wall-mount transformer
- Powers from local regulated +5 Volts
- Switches automatically between its power sources
- Provides power to your external circuitry
- Multiple protection schemes ensure high reliability

The power circuitry of the µChameleon is extremely flexible, and has been designed to adapt to as many real-world situations as possible.
First, it can be powered directly by the USB port of your computer, which is the default configuration most users are satisfied with.
In some instances however, for example when using an bus powered hub that is not capable of providing enough current, or to conserve the battery of a laptop, or to get an accurate 5 Volts level for some analog applications, the µChameleon has an internal linear regulator.

2.2.2. Power circuitry protections

The power circuitry is protected against the following situations:

- Polarity inversion of wall-mount connection
- Board power short circuit
- Current consumption over 500mA (protects your computer)
- Over-temperature of linear 5Volts regulator

2.2.3. Using an external wall-mount transformer

Simply connect a standard wall-mount transformer, with a typical output voltage between 9Volts and 12Volts to the black connector beside the usb connector. The external voltage will be internally regulated to a clean 5Volts, and the µChameleon will switch from USB power automatically.

2.2.4. Thermal considerations

When powered by an external transformer, the region near the power connector can feel warm to the touch. This is normal. However, it is suggested not to apply more than 16 Volts to the external power input, especially at high currents, as this will increase the heat produced by the internal linear regulator.