

# Turbo Chameleon 64

The Core Developers Manual

Peter Wendrich  
pwsoft@syntiac.com

November 14, 2011

## Draft version!

### Contents

<b>1</b>	<b>Introducing the Turbo Chameleon 64</b>	<b>2</b>
1.1	Reconfigurable hardware . . . . .	2
<b>2</b>	<b>JTAG</b>	<b>2</b>
<b>3</b>	<b>I/O mux</b>	<b>2</b>
3.1	FPGA and MUX communication . . . . .	3
3.2	DMA . . . . .	3
3.3	R/W . . . . .	3
3.4	NMI, IRQ . . . . .	4
3.5	EXROM, GAME . . . . .	4
3.6	IOW and IOR . . . . .	4
3.7	Flash-ROM CS . . . . .	4
3.8	RTC CS . . . . .	4
3.9	MMC CS . . . . .	4
3.10	SPI MOSI, SPI CLK . . . . .	4
3.11	LEDs . . . . .	4
3.12	IEC . . . . .	5
3.13	PS/2 . . . . .	5
<b>4</b>	<b>FPGA I/O lines</b>	<b>5</b>
4.1	$\overline{IO_e}$ and $\overline{IO_f}$ . . . . .	5
4.2	$\overline{ROML}$ and $\overline{ROMH}$ . . . . .	5
4.3	Phi-2 . . . . .	5
4.4	Dot-Clock . . . . .	6
<b>5</b>	<b>Clockport</b>	<b>6</b>
<b>6</b>	<b>USB Debug interface</b>	<b>6</b>
6.1	Commands from USB to FPGA . . . . .	6
6.2	Commands from FPGA to USB . . . . .	7
<b>7</b>	<b>PS/2 Keyboard and Mouse</b>	<b>7</b>
7.1	PS/2 protocol . . . . .	7
7.2	Using a PS/2 keyboard . . . . .	7
7.2.1	PS/2 keyboard typematic . . . . .	8
7.2.2	PS/2 keyboard scan-codes . . . . .	8
7.3	Using a PS/2 mouse . . . . .	9
7.3.1	Mouse status packet . . . . .	10

7.3.2	Mouse movement packet . . . . .	10
<b>8</b>	<b>IR (CDTV remote)</b>	<b>11</b>
8.1	IR protocol . . . . .	11
<b>9</b>	<b>Docking Station</b>	<b>11</b>
9.1	Protocol . . . . .	11
9.2	Amiga keyboard LEDs . . . . .	12
9.3	Example code . . . . .	13
<b>10</b>	<b>Pins and Signals</b>	<b>13</b>

## 1 Introducing the Turbo Chameleon 64

The "Turbo Chameleon 64" is a multi-function expansion cartridge for the Commodore-64 home computer. The name is based on the multiple function aspects of the cartridge: VGA video, mass-storage, freezer and turbo. It can also emulate many classic cartridges, while providing the new functions at the same time. Some of the cartridges emulated are freezers, speeders, games and memory expansions. Finally it has a drive subsystem that emulates a complete 1541 diskdrive on a hardware level.

### 1.1 Reconfigurable hardware

Most of the functions in the cartridge are powered by a Cyclone III FPGA chip. This reconfigurable chip is the same as used on the C-One (the reconfigurable computer) expander. The Cyclone III FPGA has enough logic cells for 16 bit and even some 32 bit computer designs. A small 8 bit micro on the cartridge allows the FPGA to be reloaded with new cores under software control. This makes the cartridge a rather powerfull FPGA development platform as well. The 16 Mbyte flash chip on the cartridge offers space for 15 user designs.

## 2 JTAG

The PCB of Chameleon is prepared for a JTAG connection. This allows downloading new designs into the FPGA without flashing them first. The connection CN1 has 10 pins with a standard Altera JTAG compatible layout. Simply solder a pin-header to make the JTAG available.

Take note though that the JTAG connector might make it impossible to mount an ethernet card in the clock-port.

## 3 I/O mux

There are not enough I/O lines on the FPGA to control all the connectors and devices on the Chameleon. The I/O lines on the FPGA are also not 5 volt tolerant, which is necessary for interfacing to the Commodore 64 expansion port. For solving both issues, there is a CPLD on the board that performs I/O multiplexing.

The MUX has some restrictions. Some signals on the expansion port are input or output only. However it is designed in such a way that it can take over the bus with DMA, but also can function as a cartridge (emulator). Some of the signal required for cartridge mode are input only and directly connected to the FPGA. This is the case for the signals like  $IO_e$  and  $IO_f$  and  $ROM_L$  and  $ROM_H$ .

The default and safe value for all registers is '1', with one exception. The signal **RTC CS** should be 0 when not selected. This chip has an active-high chip select instead of the usual active-low for the

other SPI devices. Refer to the datasheet of the RTC component (PFC2123) for more information as it has some other issues to keep in mind, like a maximum SPI clockspeed limitation of 5 Mhz or lower (at 3.3V supply voltage).

The SPI bus is shared by the startup microcontroller and the CPLD. To make sure the micro can properly start system, the FPGA must notify the CPLD that it has been startup before it will drive any chip-selects on the SPI bus. The register  $C_h$  of the MUX must be selected atleast once by the FPGA before chip-selects and SPI lines can driven by the CPLD.

### 3.1 FPGA and MUX communication

The communication between the FPGA and the MUX uses 13 lines. The lines are unidirectional with nine of them going from FPGA to the MUX and only four lines going from the MUX to the FPGA.

- 1 **mux\_clk** clock line driven by the FPGA. The CPLD clocks data on the rising edge of this clock.
- 4 **mux** lines from the FPGA to the CPLD that select one out of 15 registers and mux states inside the CPLD. All ones ( $F_h$ ) selects no registers inside the CPLD and is therefore a safe startup/reset selection.
- 4 **muxd** lines from the FPGA to the CPLD that contain the data clocked into one of the 15 I/O registers on next rising edge of **mux\_clk**.
- 4 **muxq** lines from the CPLD to the FPGA with multiplexed I/O selected by **mux**. These outputs are not clocked and therefore need to be synchronized and deglitched inside the FPGA.

The layout of the registers inside the CPLD is as follows:

	FPGA to MUX clocked on rising edge of mux_clk				MUX to FPGA			
	muxd <sub>3</sub>	muxd <sub>2</sub>	muxd <sub>1</sub>	muxd <sub>0</sub>	muxq <sub>3</sub>	muxq <sub>2</sub>	muxq <sub>1</sub>	muxq <sub>0</sub>
0 <sub>h</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1 <sub>h</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>
2 <sub>h</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
3 <sub>h</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>
4 <sub>h</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>
5 <sub>h</sub>	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>
6 <sub>h</sub>	$\overline{\text{NMI}}$	$\overline{\text{IRQ}}$	$\overline{\text{DMA}}$	$\overline{\text{RESET}}$	$\overline{\text{NMI}}$	$\overline{\text{IRQ}}$	$\overline{\text{BA}}$	$\overline{\text{RESET}}$
7 <sub>h</sub>	A <sub>15-12</sub> $\overline{\text{OE}}$	A <sub>11-0</sub> $\overline{\text{OE}}$	D <sub>7-0</sub> $\overline{\text{OE}}$	R/ $\overline{\text{W}}$	1	1	BA	R/ $\overline{\text{W}}$
8 <sub>h</sub>	EXROM	GAME	$\overline{\text{IOW}}$	$\overline{\text{IOR}}$	1	1	1	1
9 <sub>h</sub>	-	-	-	-	1	1	1	1
A <sub>h</sub>	-	-	-	-	scl	id2	sda	id0
B <sub>h</sub>	FlashROM CS	RTC CS	green LED	red LED	IR eye	1	$\overline{\text{reset button}}$	reset_out
C <sub>h</sub>	USART RX	MMC CS	SPI MOSI	SPI CLK	1	1	1	1
D <sub>h</sub>	IEC ATN	IEC SRQ	IEC CLK	IEC DAT	IEC ATN	IEC SRQ	IEC CLK	IEC DAT
E <sub>h</sub>	mouse clk	mouse dat	keyb clk	keyb dat	mouse clk	mouse dat	keyb clk	keyb dat
F <sub>h</sub>	-	-	-	-	1	1	1	1

### 3.2 DMA

When DMA is made low the CPU inside the Commodore 64 computer is stopped. This allows the Chameleon cartridge to take control over the system bus. As extra protection against core programming errors, the DMA is automatically set to low when the signal A<sub>11-0</sub>  $\overline{\text{OE}}$  is made low. The DMA line is output only.

### 3.3 R/W

When R/W signal is low it signifies a write operation on the bus. The signal is driven together with the lower address lines A<sub>11-0</sub> by bringing the register A<sub>11-0</sub>  $\overline{\text{OE}}$  low. When not driven the signal is input and can be read through the MUX.

### 3.4 NMI, IRQ

The interrupt lines can be read (input) and controlled. Making the register low will pull the interrupt line low. Setting the register high will release the interrupt line. A pullup in the Commodore 64 machine will keep it at a defined state when not driven.

### 3.5 EXROM, GAME

These two signals are output only on the MUX. They control some parts of the memory layout inside the Commodore 64 machine. Refer to the "Commodore 64 Reference Manual" for more information about EXROM and GAME.

### 3.6 IOW and IOR

Read and write signals for the clockport. Both are low-active signals and should both be high when the clockport is not accessed.

### 3.7 Flash-ROM CS

This is the chip-select of the onboard flash-ROM. The line is active when low. To prevent conflicts RTC CS must be kept low and  $\overline{\text{MMC CS}}$  must be kept high when selecting the flash ROM. Before this register can drive the select line, register  $C_h$  of the MUX must be selected at least once by the FPGA.

### 3.8 RTC CS

This is the chip-select of the Real Time Clock chip. This line is active when high. To prevent conflicts the lines  $\overline{\text{FlashRom CS}}$  and  $\overline{\text{MMC CS}}$  must be kept high when accessing the Real Time Clock. Before this register can drive the select line, register  $C_h$  of the MUX must be selected at least once by the FPGA.

### 3.9 MMC CS

This is the chip-select of the MMC card. This line is active when low. To prevent conflicts the lines RTC CS must be kept low and  $\overline{\text{FlashRom CS}}$  must be kept high when accessing the MMC card.

### 3.10 SPI MOSI, SPI CLK

Shared SPI data-out (MOSI) and clock lines for MMC, FlashROM and Real Time Clock. The data from the SPI devices to the FPGA (MISO) is directly connected to a FPGA input only pin and is not routed through the MUX.

### 3.11 LEDs

The LEDs are lit when the register is 1 and are off when the register is 0.

### 3.12 IEC

Register  $D_h$  controls the IEC bus on the Chameleon. The signals are open-collector (open-drain) and can only drive a low level. Writing a 0 in one of the IEC registers drives the corresponding line low. Writing a 1 in the register turns it into an input with a pull-up holding it high. Each IEC device should have their own pull-up resistors on the IEC lines. All signals can be input or driven low by each device on the bus. This allows multiple devices to share the same bus. The Chameleon hardware makes it possible to be the master of the bus or emulate one or even multiple devices on the IEC bus as all lines are both input and output.

Due to pin limitation the Chameleon doesn't have a reset signal on its IEC bus. So external devices might need to be reset manually. This can be done by using an optional (IEC) reset switch or by toggling the power on the device(s) that need a reset.

### 3.13 PS/2

Register  $E_h$  controls the PS/2 connectors on the Chameleon (located on the break-out cable). The green connector is for a PS/2 mouse and the purple connector is for a PS/2 keyboard. See chapter 7 for details about the PS/2 protocol and commands for the various devices.

Example code for reading keyboard and mouse is available at [http://syntiac.com/vhdl\\_lib.html](http://syntiac.com/vhdl_lib.html) The required download file is [http://syntiac.com/zips/vhdl\\_io\\_ps2.zip](http://syntiac.com/zips/vhdl_io_ps2.zip)

## 4 FPGA I/O lines

As mentioned in the previous chapter. Some lines from the C64 go directly into the FPGA. This is the case for any clocklines and for some input only signals. As the FPGA isn't 5 volt tolerant it can't drive any C64 signals directly. So all signals that need to be output are routed through the CPLD based multiplexer.

### 4.1 $\overline{IO_e}$ and $\overline{IO_f}$

The lines  $\overline{IO_e}$  and  $\overline{IO_f}$  are combined into a single signal. The combined signal  $IO_{ef}$  is a logical NAND of the two select signals. It is high if any of the two I/O select lines are driven low by the C64. By inspecting the address line A8 it is possible to detect which of the two I/O spaces is actually accessed.

For the docking-station this pin carries the start sequence pulse (see chapter 9).

### 4.2 $\overline{ROML}$ and $\overline{ROMH}$

The lines  $\overline{ROML}$  and  $\overline{ROMH}$  are handled similar to the IO select lines. If one of  $\overline{ROML}$  or  $\overline{ROMH}$  is driven low the combined signal  $RomLH$  is high and otherwise low. The address lines can be used to determine which memory area is actually accessed.

For the docking-station this pin carries the data of the serial bitstream (see chapter 9).

### 4.3 Phi-2

System clock of the C64 is directly connected to one of the FPGA pins. Take note that the signal is inverted ( $\phi_{2.n}$ ). In standalone mode this pin is always high. If the docking-station is connected this signal is pulled low. Observing this signal is the preferred method to detect in which hardware configuration the core runs.

phi2.n	Chameleon configuration	Comments
Toggling below 1 Mhz	Chameleon plugged into a PAL C64	frequency is 0.985 Mhz
Toggling above 1 Mhz	Chameleon plugged into a NTSC C64	frequency is 1.02 Mhz
High	Standalone operation	
Low	Docking-station present	

Take note that the signal has a clean high to low transition, but a slow not well defined low to high transition. Therefore the FPGA should only use the high to low transitions of the Phi-2. As signal is inverted the stable transition represents a rising-edge on the actual FPGA pin (phi2.n).

#### 4.4 Dot-Clock

The 8 Mhz pixel clock of the C64 is directly connected to one of the FPGA pins. Take note that the signal is inverted (dotclock.n).

For the docking-station this pin carries the clock of the serial bitstream (see chapter 9).

## 5 Clockport

TODO

## 6 USB Debug interface

The USB microcontroller and FPGA communicate over a synchronous serial bus. The microcontroller generates the bit clock (at about 2 Mhz). Sample the data when clock line is high (or on the rising edge) and change data when clock is low (or on the falling edge). The data format is 9 bits with no parity. If the highest bit is set it is a command, otherwise it is a databyte.

The FPGA must inform the microcontroller of its existence by sending the command  $12A_h$  after startup. The microcontroller will respond with the flash slot number used during configuration ( $110_h-11F_h$ ). Using the slotnumber the FPGA can now load additional data from the flash-chip from the correct location.

Take note that the FPGA receives data from the microcontroller directly on I/O pins (including the synchronous clock). The line for transmission towards the microcontroller however runs through the CPLD multiplexer. Make sure the register is updated at 4 MHz or faster to ensure proper transmission of data. The serial transmission bit is combined with the SPI lines that are updated 16 times per C64 cycle (which is around 1 usec) in the Chameleon core. This allows for an 8 Mbit transfer rate on the SPI bus and is also fast enough to have stable synchronous serial communication with the micro.

### 6.1 Commands from USB to FPGA

Command	Action	Additional bytes						
$100_h$	Stop							
$101_h$	Write	$A_{31-24}$	$A_{23-16}$	$A_{15-8}$	$A_{7-0}$	bytes ...	$100_h$	
$102_h$	Read	$A_{31-24}$	$A_{23-16}$	$A_{15-8}$	$A_{7-0}$	$Len_{23-16}$	$Len_{15-8}$	$Len_{7-0}$
$110_h-11F_h$	Slot number							

## 6.2 Commands from FPGA to USB

Command	Action
12A <sub>h</sub>	I'm here
1F0 <sub>h</sub> –1FF <sub>h</sub>	Reconfigure request. Lowest four bits contain slot number.

## 7 PS/2 Keyboard and Mouse

The PS/2 is an open-collector (or open-drain) bus with pull-up resistors. This allows both the host (computer) and the device (mouse or keyboard) to send and receive data. Only zeros drive the pin low and ones let it float (and turn in into an input). The pull-up resistors make sure that a pin is high when nothing pulls it low.

### 7.1 PS/2 protocol

Each byte on the interface is transmitted LSB first in a frame of 11 or 12 bits. The clock line during frame transfer is always generated by the device (keyboard or mouse) at 10 to 17 Khz. When transferring data from device to the host the data line changes when clock is high and is read by the host when the clock line is low. When transferring data from the host to the device the data line changes when the clock is low and is read by the device when the clock is high. The acknowledge bit is send by the device so the data line is pulled low on ack by the device when the clock is high. Then one more clock pulse is generated by the device and the transfer is finished.

bit	purpose
1	start bit, always 0
2	LSB of byte
3-8	bits 1 to 6
9	MSB of byte
10	parity bit (odd parity)
11	stop bit, always 1
12	acknowledge bit (host to device communication only)

The parity bit is 1 if there is a even number of ones in the byte to be send and 0 if there is a odd number of ones in the byte. The total ones in the parity and 8 data bits together is therefore always an odd number (that is why it is called odd parity).

If the host wants to send something to the device it pulls clock low for atleast 100 microseconds. This request can happen at any time even during frame transfer. After the 100 microseconds the data line is pulled low and the clock line is released, this is the begin of the start bit.

### 7.2 Using a PS/2 keyboard

For every key pressed the keyboard, it sends one or more scan-codes to the host. Sending data to the keyboard is necessary when you want to change one of the LEDs for Num-Lock, Caps-Lock or Scroll-Lock. Also the repeat rate (and delay before repeating) can be modified by sending commands to the keyboard. After each byte is send from the host, the keyboard responds with FA<sub>h</sub>.

byte	command	comments
ED <sub>h</sub>	Set/Reset LEDs	This command takes one argument byte bit0 Scroll-Lock LED (0=off, 1=on) bit1 Num-Lock LED (0=off, 1=on) bit2 Caps-Lock LED (0=off, 1=on) bit7..3 should be set to 0
EE <sub>h</sub>	Echo	Keyboard responds with EE <sub>h</sub>
F3 <sub>h</sub>	Set typematic rate	Set key-repeat rate and delay before repeat starts. This command takes one argument byte (see next table).
F4 <sub>h</sub>	Enable	Enable keyboard (after it was disabled with F6 <sub>h</sub> )
F5 <sub>h</sub>	Disable	Disables scanning and loads defaults
F6 <sub>h</sub>	Set Defaults	Load default settings (10.9cps / 500ms key repeat) and selects scancode set 2
FE <sub>h</sub>	Resend	Last byte is resend
FF <sub>h</sub>	Reset	Load default settings and performs a self-test. Keyboard sends 0xAA after self-test completes or FC <sub>h</sub> when there is an error.

### 7.2.1 PS/2 keyboard typematic

Following table show the bit allocations of the argument byte for the F3<sub>h</sub> command.

bits4..0	Rate (cps)	bits4..0	Rate (cps)	bits4..0	Rate (cps)	bits4..0	Rate (cps)	bits6..5	delay
00 <sub>h</sub>	30.0	08 <sub>h</sub>	15.0	10 <sub>h</sub>	7.5	18 <sub>h</sub>	3.7	00	250 ms
01 <sub>h</sub>	26.7	09 <sub>h</sub>	13.3	11 <sub>h</sub>	6.7	19 <sub>h</sub>	3.3	01	500 ms
02 <sub>h</sub>	24.0	0A <sub>h</sub>	12.0	12 <sub>h</sub>	6.0	1A <sub>h</sub>	3.0	10	750 ms
03 <sub>h</sub>	21.8	0B <sub>h</sub>	10.9	13 <sub>h</sub>	5.5	1B <sub>h</sub>	2.7	11	1 second
04 <sub>h</sub>	20.0	0C <sub>h</sub>	10.0	14 <sub>h</sub>	5.0	1C <sub>h</sub>	2.5	bit 7 should be 0	
05 <sub>h</sub>	18.5	0D <sub>h</sub>	9.2	15 <sub>h</sub>	4.6	1D <sub>h</sub>	2.3		
06 <sub>h</sub>	17.1	0E <sub>h</sub>	8.6	16 <sub>h</sub>	4.3	1E <sub>h</sub>	2.1		
07 <sub>h</sub>	16.0	0F <sub>h</sub>	8.0	17 <sub>h</sub>	4.0	1F <sub>h</sub>	2.0		

### 7.2.2 PS/2 keyboard scan-codes

There are three different scancode sets that can be configured on PS/2 keyboards. The first set is for XT compatibility. The following tables describe the scancode set 2, which is the default set on all PS/2 keyboards. The third scancode set is almost never used and is therefore not described in this document.

key	make	break	key	make	break	key	make	break
A	1C <sub>h</sub>	F0 <sub>h</sub> 1C <sub>h</sub>	Esc	76 <sub>h</sub>	F0 <sub>h</sub> 76 <sub>h</sub>	Space	29 <sub>h</sub>	F0 <sub>h</sub> 29 <sub>h</sub>
B	32 <sub>h</sub>	F0 <sub>h</sub> 32 <sub>h</sub>	F1	05 <sub>h</sub>	F0 <sub>h</sub> 05 <sub>h</sub>	Enter	5A <sub>h</sub>	F0 <sub>h</sub> 5A <sub>h</sub>
C	21 <sub>h</sub>	F0 <sub>h</sub> 21 <sub>h</sub>	F2	06 <sub>h</sub>	F0 <sub>h</sub> 06 <sub>h</sub>	BkSp	66 <sub>h</sub>	F0 <sub>h</sub> 66 <sub>h</sub>
D	23 <sub>h</sub>	F0 <sub>h</sub> 23 <sub>h</sub>	F3	04 <sub>h</sub>	F0 <sub>h</sub> 04 <sub>h</sub>	Tab	0D <sub>h</sub>	F0 <sub>h</sub> 0D <sub>h</sub>
E	24 <sub>h</sub>	F0 <sub>h</sub> 24 <sub>h</sub>	F4	0C <sub>h</sub>	F0 <sub>h</sub> 0C <sub>h</sub>	Caps	58 <sub>h</sub>	F0 <sub>h</sub> 58 <sub>h</sub>
F	2B <sub>h</sub>	F0 <sub>h</sub> 2B <sub>h</sub>	F5	03 <sub>h</sub>	F0 <sub>h</sub> 03 <sub>h</sub>	L Shift	12 <sub>h</sub>	F0 <sub>h</sub> 12 <sub>h</sub>
G	34 <sub>h</sub>	F0 <sub>h</sub> 34 <sub>h</sub>	F6	0B <sub>h</sub>	F0 <sub>h</sub> 0B <sub>h</sub>	L Ctrl	14 <sub>h</sub>	F0 <sub>h</sub> 14 <sub>h</sub>
H	33 <sub>h</sub>	F0 <sub>h</sub> 33 <sub>h</sub>	F7	83 <sub>h</sub>	F0 <sub>h</sub> 83 <sub>h</sub>	L Win	E0 <sub>h</sub> 1F <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 1F <sub>h</sub>
I	43 <sub>h</sub>	F0 <sub>h</sub> 43 <sub>h</sub>	F8	0A <sub>h</sub>	F0 <sub>h</sub> 0A <sub>h</sub>	L Alt	11 <sub>h</sub>	F0 <sub>h</sub> 11 <sub>h</sub>
J	3B <sub>h</sub>	F0 <sub>h</sub> 3B <sub>h</sub>	F9	01 <sub>h</sub>	F0 <sub>h</sub> 01 <sub>h</sub>	R Shift	59 <sub>h</sub>	F0 <sub>h</sub> 59 <sub>h</sub>
K	42 <sub>h</sub>	F0 <sub>h</sub> 42 <sub>h</sub>	F10	09 <sub>h</sub>	F0 <sub>h</sub> 09 <sub>h</sub>	R Ctrl	E0 <sub>h</sub> 14 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 14 <sub>h</sub>
L	4B <sub>h</sub>	F0 <sub>h</sub> 4B <sub>h</sub>	F11	78 <sub>h</sub>	F0 <sub>h</sub> 78 <sub>h</sub>	R Win	E0 <sub>h</sub> 27 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 27 <sub>h</sub>
M	3A <sub>h</sub>	F0 <sub>h</sub> 3A <sub>h</sub>	F12	07 <sub>h</sub>	F0 <sub>h</sub> 07 <sub>h</sub>	R Alt	E0 <sub>h</sub> 11 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 11 <sub>h</sub>
N	31 <sub>h</sub>	F0 <sub>h</sub> 31 <sub>h</sub>	0	45 <sub>h</sub>	F0 <sub>h</sub> 45 <sub>h</sub>	Apps	E0 <sub>h</sub> 2F <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 2F <sub>h</sub>
O	44 <sub>h</sub>	F0 <sub>h</sub> 44 <sub>h</sub>	1	16 <sub>h</sub>	F0 <sub>h</sub> 16 <sub>h</sub>	Num	77 <sub>h</sub>	F0 <sub>h</sub> 77 <sub>h</sub>
P	4D <sub>h</sub>	F0 <sub>h</sub> 4D <sub>h</sub>	2	1E <sub>h</sub>	F0 <sub>h</sub> 1E <sub>h</sub>	KP 0	70 <sub>h</sub>	F0 <sub>h</sub> 70 <sub>h</sub>
Q	15 <sub>h</sub>	F0 <sub>h</sub> 15 <sub>h</sub>	3	26 <sub>h</sub>	F0 <sub>h</sub> 26 <sub>h</sub>	KP 1	69 <sub>h</sub>	F0 <sub>h</sub> 69 <sub>h</sub>
R	2D <sub>h</sub>	F0 <sub>h</sub> 2D <sub>h</sub>	4	25 <sub>h</sub>	F0 <sub>h</sub> 25 <sub>h</sub>	KP 2	72 <sub>h</sub>	F0 <sub>h</sub> 72 <sub>h</sub>
S	1B <sub>h</sub>	F0 <sub>h</sub> 1B <sub>h</sub>	5	2E <sub>h</sub>	F0 <sub>h</sub> 2E <sub>h</sub>	KP 3	7A <sub>h</sub>	F0 <sub>h</sub> 7A <sub>h</sub>
T	2C <sub>h</sub>	F0 <sub>h</sub> 2C <sub>h</sub>	6	36 <sub>h</sub>	F0 <sub>h</sub> 36 <sub>h</sub>	KP 4	6B <sub>h</sub>	F0 <sub>h</sub> 6B <sub>h</sub>
U	3C <sub>h</sub>	F0 <sub>h</sub> 3C <sub>h</sub>	7	3D <sub>h</sub>	F0 <sub>h</sub> 3D <sub>h</sub>	KP 5	73 <sub>h</sub>	F0 <sub>h</sub> 73 <sub>h</sub>
V	2A <sub>h</sub>	F0 <sub>h</sub> 2A <sub>h</sub>	8	3E <sub>h</sub>	F0 <sub>h</sub> 3E <sub>h</sub>	KP 6	74 <sub>h</sub>	F0 <sub>h</sub> 74 <sub>h</sub>
W	1D <sub>h</sub>	F0 <sub>h</sub> 1D <sub>h</sub>	9	46 <sub>h</sub>	F0 <sub>h</sub> 46 <sub>h</sub>	KP 7	6C <sub>h</sub>	F0 <sub>h</sub> 6C <sub>h</sub>
X	22 <sub>h</sub>	F0 <sub>h</sub> 22 <sub>h</sub>	‘ ~	0E <sub>h</sub>	F0 <sub>h</sub> 0E <sub>h</sub>	KP 8	75 <sub>h</sub>	F0 <sub>h</sub> 75 <sub>h</sub>
Y	35 <sub>h</sub>	F0 <sub>h</sub> 35 <sub>h</sub>	- _	4E <sub>h</sub>	F0 <sub>h</sub> 4E <sub>h</sub>	KP 9	7D <sub>h</sub>	F0 <sub>h</sub> 7D <sub>h</sub>
Z	1A <sub>h</sub>	F0 <sub>h</sub> 1A <sub>h</sub>	=	55 <sub>h</sub>	F0 <sub>h</sub> 55 <sub>h</sub>	KP .	71 <sub>h</sub>	F0 <sub>h</sub> 71 <sub>h</sub>
; :	4C <sub>h</sub>	F0 <sub>h</sub> 4C <sub>h</sub>	\	5D <sub>h</sub>	F0 <sub>h</sub> 5D <sub>h</sub>	KP /	E0 <sub>h</sub> 4A <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 4A <sub>h</sub>
’ ”	52 <sub>h</sub>	F0 <sub>h</sub> 52 <sub>h</sub>	[ {	54 <sub>h</sub>	F0 <sub>h</sub> 54 <sub>h</sub>	KP *	7C <sub>h</sub>	F0 <sub>h</sub> 7C <sub>h</sub>
, <	41 <sub>h</sub>	F0 <sub>h</sub> 41 <sub>h</sub>	] }	5B <sub>h</sub>	F0 <sub>h</sub> 5B <sub>h</sub>	KP -	7B <sub>h</sub>	F0 <sub>h</sub> 7B <sub>h</sub>
. >	49 <sub>h</sub>	F0 <sub>h</sub> 49 <sub>h</sub>	/ ?	4A <sub>h</sub>	F0 <sub>h</sub> 4A <sub>h</sub>	KP +	79 <sub>h</sub>	F0 <sub>h</sub> 79 <sub>h</sub>
						KP Enter	6B <sub>h</sub>	F0 <sub>h</sub> 6B <sub>h</sub>

key	make	break
Print Screen	E0 <sub>h</sub> 12 <sub>h</sub> E0 <sub>h</sub> 7C <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 7C <sub>h</sub> E0 <sub>h</sub> F0 <sub>h</sub> 12 <sub>h</sub>
Scroll	7E <sub>h</sub>	F0 <sub>h</sub> 7E <sub>h</sub>
Pause	E1 <sub>h</sub> 14 <sub>h</sub> 77 <sub>h</sub> E1 <sub>h</sub> F0 <sub>h</sub> 14 <sub>h</sub> F0 <sub>h</sub> 77 <sub>h</sub>	– (no break code!)
Power	E0 <sub>h</sub> 37 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 37 <sub>h</sub>
Sleep	E0 <sub>h</sub> 3F <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 3F <sub>h</sub>
Wake up	E0 <sub>h</sub> 5E <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 5E <sub>h</sub>
Insert	E0 <sub>h</sub> 70 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 70 <sub>h</sub>
Delete	E0 <sub>h</sub> 71 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 71 <sub>h</sub>
Home	E0 <sub>h</sub> 6C <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 6C <sub>h</sub>
End	E0 <sub>h</sub> 69 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 69 <sub>h</sub>
Page Up	E0 <sub>h</sub> 7D <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 7D <sub>h</sub>
Page Down	E0 <sub>h</sub> 7A <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 7A <sub>h</sub>
Up	E0 <sub>h</sub> 75 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 75 <sub>h</sub>
Left	E0 <sub>h</sub> 6B <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 6B <sub>h</sub>
Down	E0 <sub>h</sub> 72 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 72 <sub>h</sub>
Right	E0 <sub>h</sub> 74 <sub>h</sub>	E0 <sub>h</sub> F0 <sub>h</sub> 74 <sub>h</sub>

### 7.3 Using a PS/2 mouse

The mouse sends its data in packets. Such a packet is standard three bytes in size. However after reprogramming (with something called a knocking sequence) some mice can also transmit four or

five byte long packets. These extra bytes can give information about the scroll-wheel and extra buttons on the mouse. Here only the standard 3 byte protocol is described. Before the mouse sends any data to the host it needs to be in stream mode with data reporting enabled. The two commands  $EA_h$  and  $F4_h$  can be used to switch the mouse in this mode. However if the current state of the mouse is unknown it might be better to send the reset command  $FF_h$  first. After reset the mouse is already in stream mode so sending the  $EA_h$  command is unnecessary. The host should wait for the self-test complete ( $AA_h$ ) and ID ( $00_h$ ) response codes before sending any additional commands.

byte	command	comments
$E6_h$	Set scaling 1:1	Default scaling. No processing is done on the X and Y deltas.
$E7_h$	Set scaling 1:2	Alternate scaling value. Some processing is done on the X and Y deltas, which implements speed dependent scaling.
$E9_h$	Status Request	Mouse responds with $FA_h$ followed by a status packet. (See chapter 7.3.1)
$EA_h$	Set Stream Mode	Mouse responds with $FA_h$ , resets its counters and enters stream mode.
$EB_h$	Read Data	Request a movement packet. Mouse responds with $FA_h$ followed by a movement packet. (See chapter 7.3.2)
$F0_h$	Set Remote Mode	Mouse responds with $FA_h$ , resets its counters and enters remote mode.
$F4_h$	Enable Data Reporting	Mouse acknowledges with $FA_h$ and starts sending packets when mouse is moved or buttons are pressed.
$F5_h$	Disable Data Reporting	Mouse acknowledges with $FA_h$ and stops transmitting movement data.
$F6_h$	Load Defaults	Load default values into the mouse.
$FE_h$	Resend	Request mouse to resend last data packet.
$FF_h$	Reset	Mouse responds with $FA_h$ and resets. After reset it send $AA_h$ (self-test complete) and its ID (normally $00_h$ ). If the mouse detects an problem during self-test it responds with $FC_h$ instead of $AA_h$ .

### 7.3.1 Mouse status packet

The following packet is send when requested with command  $E9_h$ .

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	always 0	Mode	Enable	Scaling	always 0	Left Button	Middle Button	Right Button
Byte 2				Resolution				
Byte 3				Sample Rate				

**L,M,R Buttons** is 1=button pressed and 0=not pressed.

**Scaling** is 1=scaling 2:1, 0=scaling 1:1

**Enable** is 1=Data Reporting Enabled, 0=Data Reporting Disabled

**Mode** is 1=Remote Mode, 0=Stream Mode

### 7.3.2 Mouse movement packet

The following packet is send when the mouse is moved and data reporting is enabled (command  $F4_h$ ) or when it is requested with command  $EB_h$ .

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	always '1'	Middle Button	Right Button	Left Button
Byte 2					X Movement			
Byte 3					Y Movement			

**L,M,R Buttons** is 1=button pressed and 0=not pressed.

## 8 IR (CDTV remote)

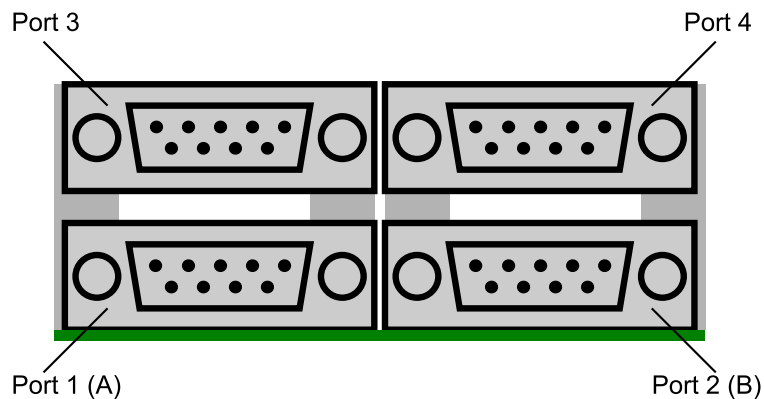
The Chameleon has a IR-eye designed (40 Khz) for an Amiga CDTV remote.

### 8.1 IR protocol

TODO

## 9 Docking Station

The docking-station adds four joystick connectors and two different keyboard connectors to the Chameleon. This supports the system with more flexibility and options when running in standalone mode. The docking-station is driven by a 8051 style micro-controller from STC. The firmware in the micro-controller performs some pre-processing and scanning without support of the FPGA. For example the docking-station will scan the eight columns of C64 keyboard autonomously. It also decodes the data-stream from the Amiga keyboard including the handling of data acknowledge and retransmissions. The FPGA will receive the pre-decoded data as a fixed sequence of octets (bytes).



### 9.1 Protocol

The docking-station transfers all data in a sequence of 13 octets. The data is transferred using a synchronous serial port. Before the sequence starts it pulses the word signal low for a few microseconds. The word signal is connected to IOe/IOf and IRQ line on the Chameleon. The IOe/IOf are directly connected to the FPGA allowing fast response. The IRQ line is used to send pulses back to the docking-station to control LEDs on an Amiga keyboard. Take note that the IOe/IOf lines come in inverted on the FPGA.

The data comes in on the ROM-L/H lines with the LSB first. Take note that the combined ROM-L/H line is inverted on the FPGA pin.

The clock is connected to dotclock\_n (again inverted). The data is output by the docking-station on the falling edge, so the data should be captured by the FPGA on the rising edge. As the dotclock\_n input is inverted this sample point is a falling-edge on the actual FPGA pin.

See the following table for the purpose of each of the thirteen octets send by the docking-station.

Byte	bits	purpose	
Status	0	0	Set if Amiga keyboard has send scancode
	1	1	C64 Restore status (low active)
	2	2	Amiga reset line status (low active)
	3-7	3-7	Reserved for future use (always 0)
Scancode	0-6	8-14	Scancode of Amiga keyboard
	7	15	Key make and break flag. Bit is clear on make and set on break/release.
C64 col0	0-7	16-23	Status of row lines when column 0 is low
C64 col1	0-7	24-31	Status of row lines when column 1 is low
C64 col2	0-7	32-39	Status of row lines when column 2 is low
C64 col3	0-7	40-47	Status of row lines when column 3 is low
C64 col4	0-7	48-55	Status of row lines when column 4 is low
C64 col5	0-7	56-63	Status of row lines when column 5 is low
C64 col6	0-7	64-71	Status of row lines when column 6 is low
C64 col7	0-7	72-79	Status of row lines when column 7 is low
P0	0	80	Joystick 2 Up
	1	81	Joystick 2 Down
	2	82	Joystick 2 Left
	3	83	Joystick 2 Right
	4	84	Joystick 2 Fire Button
	5	85	Joystick 2 Second Fire Button
	6	86	Joystick 4 Second Fire Button / Joystick 2 Third Fire Button
P1	7	87	Joystick 4 Fire Button
	0	88	Joystick 4 Right
	1	89	Joystick 4 Left
	2	90	Joystick 4 Down
	3	91	Joystick 4 Up
	4	92	Joystick 3 Right
	5	93	Joystick 3 Left
P2	6	94	Joystick 3 Down
	7	95	Joystick 3 Up
	0	96	Joystick 1 Up
	1	97	Joystick 1 Down
	2	98	Joystick 1 Left
	3	99	Joystick 1 Right Button
	4	100	Joystick 1 Fire Button
5	101	Joystick 1 Second Fire Button	
6	102	Joystick 3 Second Fire Button / Joystick 1 Third Fire Button	
7	103	Joystick 3 Fire	

## 9.2 Amiga keyboard LEDs

The docking-station can control the two LEDs on the Amiga keyboard. For this the word signal is used. The word signal is sampled between bits 47 and 48 and if high the Power LED is lit. The word signal is also sampled between bits 79 and 80 and if high the Drive LED is lit. As the word signal is driven by both the docking-station controller and the Chameleon it should be open-collector/drain output. For this reason the word signal is connected to the IRQ line on the Chameleon edge connector.

As the word signal is only weakly pulled-up there is a extra "pull high" action sixteen bit-clocks after sampling the word signal (after sending bit 64 and bit 96). The Chameleon should switch

the IRQ high before this time as not to pull with the IRQ signal against the micro-controller. It is recommended to set the word signal eight clocks before the sample period and release it eight clocks after the sample time. This makes sure the setup and hold times are correct and the release time not critically close to the "pull high" time.

For the receiving side the word signal should be ignored in the bit-clock range 32 to 96 as not trigger on the word signal while driven low to control the LEDs. The logic inside Chameleon should be in sync with the micro-controller (atleast seen and acted on one word-signal) before pulling the IRQ low. This again to guard against the Chameleon pulling low while the micro-controller pulses high.

### 9.3 Example code

Example code in VHDL for using the docking-station is included in the Chameleon hardware test version 2. The required download file is [http://syntiac.com/zips/chameleon\\_hwtest2.zip](http://syntiac.com/zips/chameleon_hwtest2.zip)

## 10 Pins and Signals

Clocks				
Name	FPGA	CPLD	C64	comments
clk8	25	-	-	8 Mhz system clock.
mux_clk	87	?	-	CPLD clock generated by FPGA.
sd_clk	44	-	-	SDRAM clock
phi2_n	88	-	E	C64 system clock (inverted)
dotclock_n	89	-	6	C64 pixel clock (inverted)

### FPGA to CPLD connection

Name	FPGA	CPLD	C64	comments
mux_clk	87	?	-	CPLD clock generated by FPGA
mux[0]	119	?	-	CPLD register selection (bit 0)
mux[1]	115	?	-	CPLD register selection (bit 1)
mux[2]	114	?	-	CPLD register selection (bit 2)
mux[3]	113	?	-	CPLD register selection (bit 3)
mux.d[0]	125	?	-	Data bit 0 from FPGA to CPLD
mux.d[1]	121	?	-	Data bit 1 from FPGA to CPLD
mux.d[2]	120	?	-	Data bit 2 from FPGA to CPLD
mux.d[3]	132	?	-	Data bit 3 from FPGA to CPLD
mux.q[0]	126	?	-	Data bit 0 from CPLD to FPGA
mux.q[1]	127	?	-	Data bit 1 from CPLD to FPGA
mux.q[2]	128	?	-	Data bit 2 from CPLD to FPGA
mux.q[3]	129	?	-	Data bit 3 from CPLD to FPGA

### SDRAM connection

Name	FPGA	CPLD	C64	comments
sd_clk	44	-	-	SDRAM clock
sd_ras_n	43	-	-	Row address select
sd_cas_n	46	-	-	Column address select
sd_we_n	50	-	-	Write enable
sd_ba_0	39	-	-	Bank select bit 0
sd_ba_1	143	-	-	Bank select bit 0
sd_ldqm	51	-	-	Lower byte select (for sd_data[7..0])
sd_udqm	49	-	-	Upper byte select (for sd_data[15..8])

sd_data[0]	83	-	-
sd_data[1]	80	-	-
sd_data[2]	79	-	-
sd_data[3]	71	-	-
sd_data[4]	68	-	-
sd_data[5]	66	-	-
sd_data[6]	64	-	-
sd_data[7]	59	-	-
sd_data[8]	58	-	-
sd_data[9]	60	-	-
sd_data[10]	65	-	-
sd_data[11]	67	-	-
sd_data[12]	69	-	-
sd_data[13]	72	-	-
sd_data[14]	77	-	-
sd_data[15]	76	-	-
sd_addr[0]	4	-	-
sd_addr[1]	6	-	-
sd_addr[2]	32	-	-
sd_addr[3]	30	-	-
sd_addr[4]	7	-	-
sd_addr[5]	8	-	-
sd_addr[6]	10	-	-
sd_addr[7]	11	-	-
sd_addr[8]	28	-	-
sd_addr[9]	31	-	-
sd_addr[10]	144	-	-
sd_addr[11]	33	-	-
sd_addr[12]	42	-	-

## Audio

Name	FPGA	CPLD	C64	comments
sigmaL	86	-	-	Left audio output
sigmaR	85	-	-	Right audio output

## VGA connector

Name	FPGA	CPLD	C64	comments
red[0]	111	-	-	
red[1]	110	-	-	
red[2]	106	-	-	
red[3]	105	-	-	
red[4]	104	-	-	
grn[0]	103	-	-	
grn[1]	101	-	-	
grn[2]	100	-	-	
grn[3]	99	-	-	
grn[4]	98	-	-	
blu[0]	112	-	-	
blu[1]	133	-	-	
blu[2]	135	-	-	
blu[3]	136	-	-	
blu[4]	137	-	-	