GECKO4COM

Technical Reference Manual

Revision 1.0.1

Dr. Theo Kluter

GECKO is an active project at HuCE-microLab where various team members have made their contributions. Currently the following researchers are involved:
Andreas Habegger, Benjamin Habegger, Fabian Nenniger, Dr. Theo Kluter,
Prof. Dr. Josef Goette, Prof. Dr. Marcel Jacomet.

HuCE microLab

Bern University of Applied Sciences
Engineering and Information Technology
Institute of Human Centered Engineering

Contents

1 Implemented Commands

The GECKO4COM implements 36 SCPI commands as shown in Table 1.1. The commands starting with a * are compliant with the IEEE488.1 [3] and IEEE488.2 [4] standards and will not be described in this chapter. The exception to the above is the description of the *PUD, *PUD? and *RST commands.

*CLS	*ESE	*ESE?	*ESR?
*IDN?	*IST?	*OPC	*OPC?
*PUD	*PUD?	*RST	*SRE
*SRE?	*STB?	*TST?	*WAI
BITFLASH	BITFLASH?	BOARD?	CONFIG
ERASE	FIFO	FIFO?	FPGA
FPGA?	HEXSWITCH	HEXSWITCH?	IDENTIFY
TRANS	USERRESET	VGA:BGCOL	VGA:CLEAR
VGA:CURSOR	VGA:CURSOR?	VGA:FGCOL	VGA:PUTSTR

Table 1.1: SCPI commands implemented in the GECKO4COM

1.1 Protected User Data

The IEEE488.1 [3] and IEEE488.2 [4] standards describe the *PUD and *PUD? as optional commands. These two commands provide a Protected User Data (PUD) area of at least 63 bytes. The GECKO4COM implements a PUD of 2048 bytes and violates the standard in the protection requirement.

Reading the PUD area:
When executing the *PUD? command the GECKO4COM puts the contents of the complete PUD memory (2048 bytes) in the output queue. The PUD memory can only be read completely; there exist no possibility to read only a part of it.

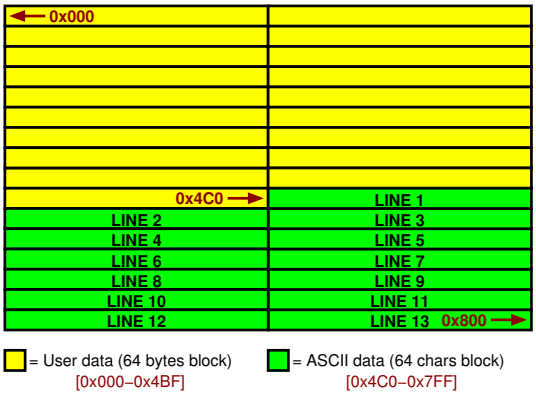

Figure 1.1: Memory organization of the PUD area. The memory is continues; the 64-byte blocks are only shown to indicate the thirteen ASCII lines.

Writing the PUD area:
The PUD area can be written with the `*PUD␣<payload>` command. The number of spaces (␣) between the command and the payload must be exactly one. All the other spaces are interpreted as payload. The size of the payload can be anywhere between 1 and 2048 bytes. Upon receiving this command the GECKO4COM will start writing at address 0 of the PUD memory. If the payload size is more than 2048 bytes the GECKO4COM will wrap around and overwrite earlier written data. After having received the payload, the GECKO4COM will store the complete 2048 bytes of the PUD memory in an external attached flash.

Note: If there is no Flash chip mounted on the GECKO4MAIN*, the PUD memory area will become volatile.*

Important: Each execution of the `*PUD` *command will initiate a sector erase/program cycle of the attached Flash chip; therefore, the number of the* `*PUD` *command invocation is limited by the number of sector erase cycles specified by the Flash chip manufacturer.*

PUD memory organization:
The PUD memory organization is shown in Figure 1.1. The area between

0x000 and 0x4BF is completely user definable; however the area between 0x4C0 and 0x7FF is used to store thirteen lines of sixty four ASCII characters that are displayed in the Static User Window (SUW) of the VGA-controller (see Chapter 3.1).

PUD command error handling:
Both the *PUD and *PUD? are always executed successfully; therefore there is no command error generated for either of these commands.

1.2 Global Reset

The GECKO4COM provides the means to activate the Global Reset. It is important to not confuse the Global Reset with the User Reset as described in Chapter 1.3. The GECKO4COM activates the Global Reset by pulling the signal actively low. In case of a deactivated Global Reset, the GECKO4COM leaves this pin floating. No pull-up on this signal is provided by the GECKO4COM. For more details on the requirements of the Global Reset (GR) signal please refer to the GECKO4MAIN technical reference guide [2].

GR activation:
There are two situations for which the GECKO4COM activates the Global Reset line, namely:

1. **Unconfigured FPGA:** If the user FPGA on the GECKO4MAIN is not configured the Global Reset line is kept activate. In case no user FPGA is mounted, the the Global Reset line is deactivated.

2. **Reset command:** By executing the *RST command, the Global Reset line is activated for a period between 10 ms and 11 ms.

☞ *Important: The* GECKO4COM *is uneffected by the state of the Global Reset line.*

GR command error handling:
The *RST command always executes correctly; therefore there is no command error generated for this command.

1.3 User Reset

Contrary to the Global Reset signal the User Reset signal is only connected to the user FPGA on the GECKO4MAIN. This User Reset signal provides the means of resetting the logic inside the user FPGA without effecting attached boards. The User Reset (UR) signal is an *active low* signal that is actively driven for both a `1` and a `0`. The User Reset signal is connected to pin **Y9** of the user FPGA.

UR activation:
There are two situations for which the User Reset line is activated, namely:

1. **Unconfigured FPGA:** If the user FPGA on the GECKO4MAIN is not configured the User Reset line is kept activate up to a period of 10 ms to 11 ms after the done line of the user FPGA went high. In case no user FPGA is mounted, the User Reset line is deactivated.
2. **Reset command:** By executing the `USERRESET` command, the User Reset line is activated for a period between 10 ms and 11 ms.

☞ *Important: The* GECKO4COM *is uneffected by the state of the User Reset line.*

☞ *Important: The User Reset line is uneffected by the state of the Global Reset line.*

UR command error handling:
The `USERRESET` command always executes correctly; therefore there is no command error generated for this command.

1.4 Bitfile Storage

To be able to operate the GECKO4MAIN in an environment where a USB connection is not wanted, the GECKO4COM provides the means to store the user FPGAs configuration in a non-volatile memory. The user FPGAs configuration, further referred to as *bitfile*, is generated by the Xilinx toolchain and consists of a variable length header and the configuration data in form of a bitstream [1]. The GECKO4COM provides four commands to manipulate the bitfile storage.

- `BITFLASH␣<bitfile>`. This command stores the `<bitfile>` in the non-volatile memory of the GECKO4COM. This command can take upto two seconds to complete due to the speed of the non-volatile memory. This command is only succesfull in case of an empty non-volatile memory, otherwise an execution error is generated.
  Important: (1) The GECKO4COM *does not control if the provided bitfile is suitable for the mounted user FPGA and (2) only a single space (␣) must be provided between the command and the bitfile.*
- `CONFIG`. This command configures the user FPGA with the bitfile stored in the non-volatile memory. If the non-volatile memory does not contain a bitfile an execution error is generated.
- `ERASE`. This command erases the non-volatile memory and must be provided before the programming of a new bitfile into the non-volatile memory. This command can take several seconds to complete.
- `BITFLASH?`. This command returns `EMPTY` if the non-volatile memory is empty, and the stored bitfile otherwise.

Note: It is highly discouraged to use the `BITFLASH <bitfile>` *and* `ERASE` *commands in a script due to their execution time. Most likely a time-out error is generated on the USBTMC protocol due to these execution times.*

1.5 GECKO4MAIN information

The GECKO4COM provides two commands to report the current status of the GECKO4MAIN system. Both these commands always execute successfully and return an ASCII string.

- `FPGA?`. This command returns the detected user FPGA type and its configuration state. Possible user FPGAs are listed in the GECKO4MAIN Technical Reference Manual [2]. If no user FPGA is mounted or the GECKO4COM was unable to determine the FPGA type this command returns the message *No FPGA mounted or unknown FPGA type* and all user FPGA manipulations are prohibited.

- `BOARD?`. This command returns the powering status, e.g. USB supplied, GENIO1 supplied, powering BUS, of the GECKO4MAIN and the status of the GECKO4COM's non-volatile memory (empty or programmed) [see previous section].

Furthermore the GECKO4COM provides the means of identifying optically an attached GECKO4MAIN by means of the command `IDENTIFY`. After execution of this command the eight LEDs of GECKO4MAIN will start to become red, afterwards they change to green and finally they will go off one after the other. This sequence lasts for approximately one second.

1.6 User FPGA configuration

The GECKO4COM provides the means to configure the user FPGA. This configuration is independent of the current state of the user FPGA. This means that the user FPGA can be configured as many times as required and with as many bitfiles as wanted. By this flexibility of the GECKO4COM progression testing and scripted simulations/emulations. The configuration time for even the largest Spartan 3 5000 FPGA is only a few tens of milliseconds.

The command to configure the user FPGA is `FPGA␣<bitfile>`. This command **must** only contain one space (␣). In case no user FPGA is mounted or the GECKO4COM was unable to detect the mounted user FPGA this command will generate an execution error.
Important: The GECKO4COM *does not control if the provided bitfile is suitable for the mounted user FPGA. It is up to the user to make sure that the bitfile that is uploaded is generated for the correct user FPGA type. In case a wrong bitfile is uploaded the command will execute successfully; however the user FPGA will refuse to load it. To be able to see if the user FPGA has been configured correctly the* `FPGA?` *command can be used to ask the current status if the user FPGA.*

1.7 Mechanical Switch

The GECKO4MAIN provides a hexadecimal encoded switch that is further referred to as *hexswitch*. The GECKO4COM provides two commands to manipulate this switch.

- `HEXSWITCH␣<value>`. This command overrides the mechanical switch and forces the `<value>` to appear on both the user FPGA side (see Chapter 2.2) as well as from the PC side. The parameter `<value>` can be any of the set [0-9,A-F,a-f]. Between the command and the parameter `<value>` there may be as many spaces (␣) as wanted. This command executes successfully as long as the parameter `<value>` is in the set mentioned before. This command is persistent until either the board is powered, or an `INITIATE_CLEAR` USBTMC message [5] is send. This command may be send as often as required and will each time override the previous value send.

- `HEXSWITCH?`. This command reports the current state of the hexswitch or the override value if a `HEXSWITCH` command has been issued before this command. The result of this command is an ASCII character of the set [0-9,A-F] followed by a new-line character (`0x0A`).

1.8 User FPGA communication

The GECKO4COM provides three commands that allows for communication between the PC and the user FPGA. These commands are `FIFO`, `FIFO?`, and `TRANS`. More information on these commands and the protocols involved can be found in Chapter 4.

1.9 VGA controller

The GECKO4COM provides a VGA controller not only for debug purposes but also for demonstration purposes. The VGA controller implemented in the GECKO4COM has one USBTMC window (see Chapter 3.1 for further information). The commands to manipulate this window are listed below.

- `VGA:FGCOL␣<value>`. This command sets the forground color of the VGA window. The parameter `<value>` can be any character between `0` and `7` (see also Chapter 3.1).

- `VGA:BGCOL␣<value>`. This command sets the background color of the VGA window. The parameter `<value>` can be any character between `0` and `7` (see also Chapter 3.1).

- `VGA:CLEAR`. This command clears the VGA window and puts the cursor on the left top position of the screen.

- `VGA:CURSOR?`. This command returns the current cursor position. The format returned is a string in the form `<x>,<y>` where x $\in$ [0..63] and y $\in$ [0..31].

- `VGA:CURSOR␣<x>,<y>`. This command sets the current cursor position to (x,y). The parameters: x $\in$ [0..63] and y $\in$ [0..31].
  Note: The parameters x and y must be specified as ASCII characters and not as bytes, e.g. 31 equals the character "3" (0x33) followed by the character "1" (0x31).

- `VGA:PUTSTR␣<string>`. This command displays all the bytes following the space (␣) at the current cursor position on the VGA window as ASCII characters. A new-line character (`0x0A`) will forse a new line and all bytes with a value smaller than the ASCII character space (`0x20`) will be ignored.
  Important: No commands can be concatinated with this command. Doing so will "print" them on the VGA screen rather than executing them.

1.10 The Status Byte Register

The IEEE488.1 [3] and IEEE488.2 [4] standards define an obligatory status byte register as shown in Figure 11-8 in the IEEE488.2 [4] standard. The GECKO4COM implements this register with the obligatory *MSS*, *ESB*, and *MAV* bits. Besides from these obligatory bits, the GECKO4COM implements 2 extra status bits.

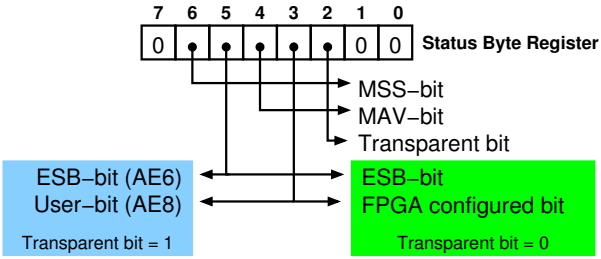

Figure 1.2: The Status Byte Register as defined by the GECKO4COM.

- **Bit 3.** Bit 3 in the Status Byte Register indicates when `1` that the user FPGA is configured and running. When this bit is `0` the user FPGA is either not present or not configured.

- **Bit 2.** Bit 2 in the Status Byte Register indicates when `1` that the GECKO4COM is in *transparent mode* (see Chapter 4.3). When this bit is `0` the GECKO4COM is in normal operation mode.

If the GECKO4COM is in *transparent mode* it does only supply the *MSS* bit, the *MAV* bit, and the transparent bit (bit 2) of the Status Byte Register. It is left to the user FPGA to provide the following bits:

- **Bit 7 (ESB).** In *transparent mode* the user FPGA MUST provide the *ESB* bit. This bit is at pin `AE6` of the user FPGA and of type LVCMOS25.

- **Bit 3.** In *transparent mode* this bit is user definable by the user FPGA. This bit is at pin `AE8` of the user FPGA and of type LVCMOS25.

For the user FPGA to know if the GECKO4COM is in *transparent mode*, the pin `AD6` at the user FPGA is a copy of bit 2 in the Status Byte Register. Figure 1.2 shows the Status Byte Register as provided by the GECKO4COM.

References

[1] Alan Nishioka and Philip Freidin, *FPGA-FAQ 0026—Tell me about the .BIT file format.* `www.fpga-faq.com`, Nov. 2001

[2] Dr. Theo Kluter, GECKO4MAIN *Technical Reference Manual.* Bern University of Applied Sciences, Biel/Bienne, Switzerland, 2011.

[3] IEEE, Inc. *IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.* ISBN 1-55937-238-9, New York, 1992.

[4] IEC and IEEE. *IEC 60488-2, IEEE 488.2: Standard digital interface for programmable instrumentation — Part 2: Codes, formats, protocols and common commands.* First edition, 2005-05.

[5] USB Implementers Forum, Inc. *Universal Serial Bus Test and Measurement Class Specification (USBTMC)–Revision 1.0.* April, 2003

2 Bus Interface

To be able to use the GECKO4COM from the user FPGA it provides a bus interface. The provided IP-cores in the GECKO4COM can be accessed through this bus and are accessed by a memory map.

2.1 Bus Signals

Table 2.1 list all the bus signals as used by the GECKO4COM, their direction, and their pin on the user FPGA. The bi-directional signals (marked by BIDIR) are forced either by the GECKO4COM or the user FPGA depending on the bus protocol (see Appendix A and Chapter 2.3). The signals

Signal	Type	Pin	Signal	Type	Pin
start trans	UF-GC AL	AB11	end trans	BIDIR AL	W11
valid lo	BIDIR AL	AB10	valid hi	BIDIR AL	AC10
error	GC-UF AL	AA10	start send	GC-UF AL	AA11
request irq	GC-UF	AC11	error irq	GC-UF	Y10
available irq	GC-UF	AF8	bus reset	UF-GC AL	Y11
data cntrl 0	BIDIR	AD4	data cntrl 1	BIDIR	AD5
data cntrl 2	BIDIR	AE4	data cntrl 3	BIDIR	AE5
data cntrl 4	BIDIR	AF4	data cntrl 5	BIDIR	AF5
data cntrl 6	BIDIR	W12	data cntrl 7	BIDIR	W13
data cntrl 8	BIDIR	Y12	data cntrl 9	BIDIR	Y13
data cntrl 10	BIDIR	AA13	data cntrl 11	BIDIR	AD12
data cntrl 12	BIDIR	AA6	data cntrl 13	BIDIR	AA7
data cntrl 14	BIDIR	AB6	data cntrl 15	BIDIR	AB7
bus clock	GC-UF	AE13	*All signals:* LVCMOS25		

Table 2.1: The signals of the bus provided by the GECKO4COM. AL represents active low. UF-GC represents a signal driven by the user FPGA and consumed by the GECKO4COM. GC-UF represents a signal driven by the GECKO4COM and consumed by user FPGA. BIDIR represents a bi-directional signal.

marked with AL are active low signals. The signals marked with GC-UF are driven by (outputs of) the GECKO4COM. The signals marked with UF-GC are driven by (outputs of) the user FPGA. The bus is synchrone and is clocked with the 48MHz clock available on pin AE13.

2.2 Memory Map

The IP cores of the GECKO4COM are available by the memory map shown in Table 2.2. Accessing an address not in this list will generate a bus error. Furthermore, writing to a read-only (RO) or reading from a write-only (WO) location will also trigger a bus error. The addresses marked with BURST are capable of burst accesses. Executing a burst on a none burst capable address will trigger a bus error.

The usage of the Tx and Rx fifos (addresses 0x00−0x0F) are described in Chapter 4. The VGA controller (addresses 0x20−0x25) is described in Chapter 3.1. Finally, the buttons, the hexswitch, and the LEDs (addresses 0x26−0x2F) are described in Chapter 3.2.

2.3 Bus Protocol

The implemented bus provides a handshake protocol with a combined data/address/control path. All signals are related to the positive edge of 48 MHz bus clock. The bus clock is generated by the GECKO4COM and cannot be changed. In case of a different system clock in the user FPGA, a clock domain adaptation needs to be implemented.

The bus provides a single transfer operation and burst transfers up to a burst size of 512 shorts (16-bit words). The bus operates pipelined. To keep the theoretic maximum throughput of 48 Mega bytes per second the bus-overhead cycles for the bus protocol should not exceed the burst size. The bus-overhead imposed by the GECKO4COM is maximum 6 clock cycles. If the burst-size is chosen smaller than 6, the theoretic maximum throughput of 48 Mega bytes per second cannot be achieved!

Design considerations:
When designing an IP-core that communicates with the GECKO4COM's

Address	Function	Access
0x00	Tx-fifo data write	WO BURST
0x01	Tx-fifo package size write	WO BURST
0x02	Nr. of bytes in TX-fifo	RO BURST
0x03	Max. size in bytes of TX-fifo	RO BURST
0x04	Nr. of shorts in TX-fifo	RO BURST
0x05	Max. size in shorts of TX-fifo	RO BURST
0x06	Nr. of words in TX-fifo	RO BURST
0x07	Max. size in words of TX-fifo	RO BURST
0x08	RX-fifo read data	RO BURST
0x09	RX-fifo read data	RO BURST
0x0A	Nr. of bytes in RX-fifo	RO BURST
0x0B	Max. size in bytes of RX-fifo	RO BURST
0x0C	Nr. of shorts in RX-fifo	RO BURST
0x0D	Max. size in shorts of RX-fifo	RO BURST
0x0E	Nr. of words in RX-fifo	RO BURST
0x0F	Max. size in words of RX-fifo	RO BURST
0x20	VGA foreground color	RW
0x21	VGA background color	RW
0x22	VGA cursor x position	RW
0x23	VGA cursor y position	RW
0x24	VGA write ASCII char	WO
0x25	VGA write dummy to clear screen	WO
0x26	Buttons status	RO
0x27	Hex switch status	RO
0x28	LED 0 mode register	RW
0x29	LED 1 mode register	RW
0x2A	LED 2 mode register	RW
0x2B	LED 3 mode register	RW
0x2C	LED 4 mode register	RW
0x2D	LED 5 mode register	RW
0x2E	LED 6 mode register	RW
0x2F	LED 7 mode register	RW

Table 2.2: Memory map of the GECKO4COM IP cores.

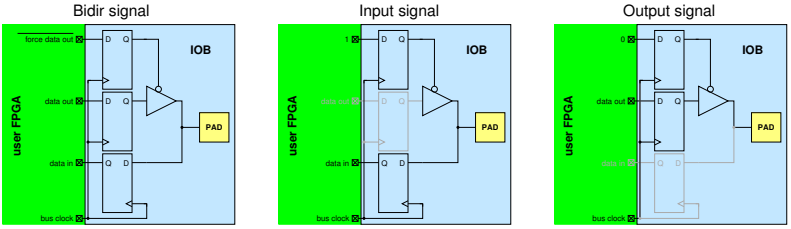

Figure 2.1: Recommended pin configurations for the bus pins in the user FPGA. each of the flipflops marked in black needs to be LOC-ed into the FPGA's pin IOB.

bus it is important that all bus signals (listed in Table 2.1) have positive edge triggered flipflops that are "LOC-ed" in the IOBs of the user FPGA. This provides maximum reliability. Not adhering to this design consideration may cause an error prone communication medium. Figure 2.1 shows an example circuit for different types of IOBs.

Control Word:
The communication over the bus is organized in transactions. Each transaction is initiated by the user FPGA. The initiation is done by sending an active **start trans** signals together with the *Transaction Control Word (TCW)* on the **data cntrl** lines. The format of the TCW is shown in Figure 2.2. The TCW consists of three distinct parts:

- **Address.** The Address is a six bit word identifying the entry in Table 2.2. The least significant bit of the address is contained on the

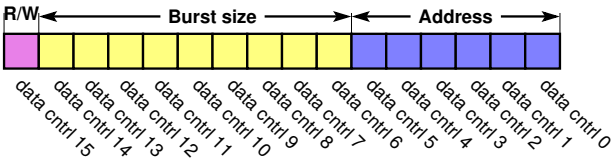

Figure 2.2: The format of the *Transaction Control Word (TCW)*. For the *address* part the signal **data cntrl 0** is the Least Significant Bit (LSB). The signal **data cntrl 6** represents the LSB of the *burst size* part of the TCW.

data cntrl 0 line.

- **Burst size.** The burst size is a nine bit word identifying the number of data to be transferred in this transaction. A burst size value of `0x000` identifies a single datum transfer, whilst a value of `0x1FF` identifies a burst of 512 shorts (16-bit values). The least significant bit of the *Burst size* field is contained on the **data cntrl 6** line.

- **R/W.** The read not write signal identifies the direction of the data to be transferred. If the *R/W* bit is `1` then a read transaction is requested and the data will flow from the GECKO4COM to the user FPGA. If the *R/W* bit is `0` then a write transaction is requested and the data will flow from the user FPGA to the GECKO4COM.

Control Signals:
The bus provides seven transfer control signals. The function of these signals is described below.

- $\overline{\textbf{start trans}}$. This signal is active low and initiates a read or a write transfer over the bus. This signal is always driven by the user FPGA.

- $\overline{\textbf{end trans}}$. This signal is active low and indicates the end of a read or a write transfer over the bus. This signal is driven by the user FPGA in case of a write transaction and by the GECKO4COM in case of a read transaction.

- $\overline{\textbf{start send}}$. This signal is active low and indicates that the user FPGA may start sending the data payload if a write transfer is requested. In case of a read transfer this signal has no function. This signal is driven by the GECKO4COM.

- $\overline{\textbf{error}}$. This signal is active low and indicates a bus error (see Chapter 2.2 for the error conditions). If a write transfer is in progress the user FPGA must respond with the activation of the $\overline{\text{end trans}}$ signal. In case of a read transfer the GECKO4COM activates this signal simultaneously with the $\overline{\text{end trans}}$ signal. This signal is driven by the GECKO4COM.

- $\overline{\textbf{valid lo}}$ and $\overline{\textbf{valid hi}}$. These two signals are active low and indicate during the data transfer that the *data cntrl* lines contain valid data.

In a write transfer the user FPGA must drive these signals. In this case the only valid values are `11` when no valid data is present and `00` when the data lines contain valid data. In a read transfer the GECKO4COM drives these signals, and their meaning is listed below. The combination `01` is special as it indicates the last datum of a USBTMC packet (see also Chapter 4.1).

$\overline{\text{valid hi}}$	**$\overline{\text{valid lo}}$**	**Meaning**
1	**1**	No valid data present.
1	**0**	Only the low 8 bits are valid.
0	**1**	All 16 bits are valid and last datum.
0	**0**	All 16 bits are valid.

- **$\overline{\text{bus reset}}$**. This signal is active low and forces when active (a) the reset of the bus interface of the GECKO4COM and (b) the flush of the Rx and Tx fifos (see Table 2.2 and Chapter 4).

Timing:
The correct order and timing of the (burst) read and (burst) write transactions are shown in Appendix A.

2.4 Interrupts

The GECKO4COM provides three active high interrupts. These interrupts are activated during at least one **bus clock** cycle. Table 2.3 lists the three interrupts and their pin on the user FPGA.

Interrupt	**Pin**
Error Irq	`Y10`
Data Available Irq	`AF8`
Data Request Irq	`AC11`

Table 2.3: Interrupts provided by the GECKO4COM. The interrupts are active high and LVCMOS25.

The function of each of these three interrupts is described below.

- **Error Interrupt.** The error interrupt is activated if (1) the user FPGA is writing to a full Tx FIFO, (2) the user FPGA is reading from an empty Rx FIFO, or (3) the user FPGA failed to write at least one package size short at the beginning of a Tx message (see Chapter 4.1).

- **Data Available Interrupt.** The data available interrupt is activated if (1) the GECKO4COM is in transparent mode and an USBTMC package has been received (see Chapter 4.3) or (2) if a `FIFO` commando was issued.

- **Data Request Interrupt.** The data request interrupt is only activated if the GECKO4COM is not in transparent mode and a `FIFO?` commando was issued.

3 Input and Output

The GECKO4COM provides input and output functionality by the implementation of a VGA controller, button and switches, LEDs, and a RS232 pass-through. Each of these IO components will be discussed in this chapter.

3.1 VGA controller

The VGA controller provides a VGA screen at the resolution of 1024x768 by a refresh rate of 60Hz. The screen can only display ASCII characters (so no graphical mode is provided) and is divided into three display areas (see Appendix B). The screen uses a simple 8-color palette as shown in Figure 3.1.

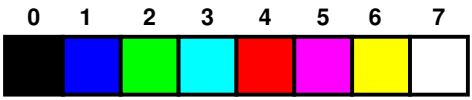

Figure 3.1: The color palette as used by the VGA controller. The numbers above the colors indicate the index for the foreground or background color.

The VGA controller provides a 7-bit ASCII table. As all implemented commands use a byte to put characters on the screen, the 8^{th} bit is used for color inversion. If the 8^{th} bit equals to `0` the character pixels are displayed in the foreground color and the rest is displayed in the background color. If the 8^{th} bit equals to `1` the character pixels are displayed in the background color and the rest is displayed in the foreground color. Each character consists of 16 lines each containing 8 pixels.

The three screens provided by the VGA controller are described below.

- **Static User Window (SUW).** This window displays a static contents that is stored in the non-volatile memory of the GECKO4COM. The colors of this window are fixed to a background color black (`0`) and

a foreground color white (7). The contents of this screen can be changed by the *PUD command described in Chapter 1.1.

- **USBTMC Message Window(UMW).** This window can be controlled by the VGA: SCPI commandos described in Chapter 1.9. At startup the GECKO4COM initializes the foreground color to white (7) and the background color to black (0). The cursor is always displayed and blinks with a frequency of 1 Hz. This window scrolls automatically when the cursor reaches the end of the screen; however, no scroll-back buffer is provided.

- **User FPGA Message Window (FMW).** This window can be controlled by the the memory mapped registers described in Chapter 2.2. At startup the GECKO4COM initializes the foreground color to red (4) and the background color to black (0). The cursor is always displayed and blinks with a frequency of 1 Hz. This window scrolls automatically when the cursor reaches the end of the screen; however, no scroll-back buffer is provided.

3.2 Buttons, Switch and LEDs

The GECKO4COM provides three general purpose buttons and one hexswitch (see Appendix B). The hexswitch can be manipulated by USBTMC commandos (see Chapter 1.7) and read out by the user FPGA through a memory mapped register (see Chapter 2.2). When reading the register at address 0x27 the current value of the hexswitch is returned in the lower 4 bits of the register. The resting bits are set to 0.

The state of the buttons can only be read out by the user FPGA through a memory mapped register (see Chapter 2.2). When reading the register at address 0x26 the current state of the buttons is returned. The current state is 0 when the button is not pressed and 1 otherwise. The GECKO4COM returns the state of BUTTON1 in bit 0, BUTTON2 in bit 1, and BUTTON3 in bit 2. All the other bits of this register are set to 0.
Note: BUTTON3 *has a special functionality; if this button is pressed during power-on or during board reset the FX2 is prevented to load its firmware stored in the* GECKO4COM*'s non-volatile memory.*

Value	LED function
0x0	LED is off
0x1	LED is off
0x2	LED is continues red
0x3	LED is continues green
0x4	LED is slowly blinking red
0x5	LED is slowly blinking green
0x6	LED is slowly toggling red-green
0x7	LED is slowly toggling red-green
0x8	LED is off
0x9	LED is off
0xA	LED is continues red
0xB	LED is continues green
0xC	LED is fast blinking red
0xD	LED is fast blinking green
0xE	LED is fast toggling red-green
0xF	LED is fast toggling red-green

Table 3.1: Values and function of the memory-mapped LED control register.

Besides the buttons and the switch, the GECKO4COM also controls eight bi-color LEDs. During normal operation these LEDs can be controlled by the user FPGA through memory mapped registers (see Chapter 2.2). Each LED has a 4-bit control register. The values that can be written to these control registers and the operation of the LED is listed in Table 3.1. There are two situation in which the LEDs are controlled by the GECKO4COM:

1. In case of an identification command the LEDs light up as described in Chapter 1.5.

2. If the GECKO4COM is busy and cannot accept new command (for example during the erase of the non-volatile memory), the LED7 will light up red and the other LEDs will be off.

3.3 RS232 pass-through

The GECKO4COM provides two signals that are put on the external connector and can be used for example as carriers for a RS232 communication protocol. The signals on the connector are LVTTL whilst the signals on the user FPGA are LVCMOS25. Table 3.2 shows the signals and their pin on the user FPGA.

Name	Direction	Pin
RS232TxD	OUTPUT	AF13
RS232RxD	INPUT	Y16

Table 3.2: RS232 communication lines as provided on the user FPGA. Both these signals are LVCMOS25.

3.4 User clocks

The GENIO1 connector on the GECKO4MAIN provides two user clocks. These clocks are connected to the GECKO4COM for level adaptation. In the GECKO4COM these two clocks are put on a Digital Loop Lock (DLL) for timing compensation. The GECKO4COM provides the user clocks including a lock signal to the user FPGA. If the lock signal is `0` the clocks should not be used, and if the lock signal is `1` the clock is stable and locked by the GECKO4COM. The DLL used in the GECKO4COM poses some restriction on the user clocks. The restriction is the frequency range a user clock is allowed to have. The user clocks must be in the range of [18MHz...167MHz] with a duty cycle in between 40% and 60%. Table 3.3 list the connections to the user FPGA.

Signal	Pin	Signal	Pin
User Clock 1	AF14	User Lock 1	AA9
User Clock 2	AE14	User Lock 2	AB9

Table 3.3: User Clock and User Lock signals on the user FPGA. All signals are INPUTS and of type LVCMOS25.

4 User FPGA communication

The GECKO4COM provides two FIFOs of each four kilo bytes to stream data from the PC to the user FPGA and vice versa. These FIFOs can operate in the IEEE488.x mode or in transparent mode. This chapter will first introduce how to steam data through these FIFOs. After that it will describe the IEEE488.x mode. It will end with the description of the transparent mode.

4.1 FIFO communication

The GECKO4COM uses the USBTMC protocol to stream data over an USB connection. This USBTMC protocol has three important aspects:

- **Packet format**. Each USBTMC packet consists of a header and data payload. In all cases the GECKO4COM handles the header generation/interpretation. The user FPGA only sees/provides the data payload.
- **Payload format**. The data payload as provided in an USBTMC packet is byte based, e.g. can be any number of bytes (minimal one).
- **Message size**. Messages that are send/received can be divided into multiple USBTMC packets.

To be able to be as versatile as possible, the GECKO4COM imposes that the messages send to/received from the user FPGA MUST be word aligned. This means that the **message size** MUST always be a multiple of four bytes. Failing to do so may result in unpredictable results, or even hang the GECKO4COM.

As the USBTMC protocol does not provide the means to send the size of a message with the first USBTMC packet, special care has to be taken when reading messages coming from the PC. Messages coming from the PC are stored in the Rx-FIFO. The Rx-FIFO can be accessed by the memory-mapped registers as described in Chapter 2.2. For the user FPGA to be able

to determine the end of a message, the GECKO4COM marks the last short (2 bytes) of the message with a special valid combination (see Chapter 2.3).

Important: An empty Rx FIFO does not mean that a message has ended, only the special valid combination indicates the end of a send message.

Similar to the reception of messages, also the transmission of messages can be split-up in multiple USBTMC packets. For the GECKO4COM to know the size of a transmitted message, the user FPGA must write first the size of the message (one or two short(s) [2 bytes]) prior to putting the messages bytes in the Tx-FIFO. The size register and the Tx-FIFO can be accessed by the memory-mapped registers as described in Chapter 2.2.

Important: The user FPGA MUST *send as many bytes as it indicated in the size register. If the Tx-FIFO is full, the user FPGA* MUST *wait until it can send again data to the Tx-FIFO.*

4.2 IEEE488.x mode

The *IEEE488.x* mode is the default mode of the GECKO4COM. In this mode it provides IEEE488.x compliancy and the SCPI command interpretation of the commands described in Chapter 1. In this mode messages to and from the user FPGA can ONLY be send by using the `FIFO` and `FIFO?` command set; hence the PC forms the master and the user FPGA has the role of slave.

Sending messages from the PC to the user FPGA
Before explaining the protocol involved in sending messages it is emphased that, as described in the previous section, the size of a message should be at least 4 bytes and word aligned (meaning an integer multiple of 4 bytes). Although the GECKO4COM will fill up the message with up to 3 dummy bytes in case of word-mis-alignment, no guarantees are given on proper operation.

The sending of a message is initiated by invoking the `FIFO␣<message>` command at the PC side. Between the command and the message only a single space (␣) is allowed; all other spaces are interpreted as being part of the message. The message (`<message>`) itself may be any stream of bytes, there are no restrictions as all bytes of the message will be send 1-to-1 to the Rx-FIFO of the GECKO4COM. At the moment the GECKO4COM

receives the FIFO command it will activate the **Data Available Interrupt (DAI)** (see Chapter 2.4) for one **bus clock** cycle. Directly after the generation of the DAI the GECKO4COM will start copying the message to the Rx-FIFO.

The user FPGA MUST after the reception of the DAI start reading the message data from Rx-FIFO within one second. Failing to do so may result in a communication time-out in the USBTMC communication channel. The user FPGA must read data from the Rx-FIFO up to the short that is marked as last short of the message (see Chapter 2.3). To avoid time-outs in the USBTMC communication channel the user FPGA has to read at least 512 bytes each second.

Receiving messages from the user FPGA
To receive a message from the user FPGA the PC has to initiate the communication by sending the FIFO? command to the GECKO4COM. After the reception of this command the GECKO4COM will activate the **Data Request Interrupt (DRI)** (see Chapter 2.4) for one **bus clock** cycle. After the generation of the DRI the GECKO4COM will wait for the message from the user FPGA and transfer it to the PC.
The user FPGA must after the reception of the DAI send the message size and some message bytes within one second. Failing to do so may result in a communication time-out in the USBTMC communication channel. The user FPGA starts the sending of the message by first writing its size to the size register at address 0x01 (see Chapter 2.2). The size is written by first writing the low short (2 bytes) to address 0x01 followed by the writing of the high short to address 0x01. Example: The message size is 0x000A1234 bytes. Sending this message would require writing (1) the value 0x1234 to address 0x01, followed by (2) writing the value 0x000A to address 0x01, followed by (3) writing all the messages bytes to address 0x00. The user FPGA must make sure that if the Tx-FIFO is not full it fills it with at least 512 bytes each second; failing in doing so may trigger a time-out in the USBTMC communication channel.

4.3 Transparent mode

The GECKO4COM can be put into transparent mode by invoking the `TRANS` command (see Chapter 1). Once put in transparent mode the GECKO4COM can only be put back into *IEEE488.x* mode by power cycling the board, resetting the board, or invoking an `INITIATE_CLEAR` USBTMC message.

Important: The GECKO4COM *will only switch to transparent mode if the user FPGA is configured. If the user FPGA is not configured an execution error is generated and the* GECKO4COM *stays in IEEE488.x mode.*

When in transparent mode the GECKO4COM will disable all command interpretations; all commands, as listed in Chapter 1, are interpreted as part of a message and are not executed. In the transparent mode all messages send from the PC are directly copied to the Rx-FIFO. Each new message will trigger a **Data Available Interrupt (DAI)**. The sending of messages from the user FPGA to the PC can be done by writing the message size followed by the message bytes themselves in the Tx-FIFO.

Note: In the transparent mode the Data Request Interrupt (DRI) has no function.

This mode is useful when the user wants to implements a new SCPI command interpreter (see also Chapter 1.10 concerning the Status Byte Register) or an own protocol on top of the USBTMC protocol.

Appendices

A Bus transactions

This appendix describes the timing diagram of valid bus transactions and transactions containing errors. Although the timing diagrams are shown for a three datum burst transaction, it also reflects the single datum transaction.

A.1 Write transactions

In the write transaction the data flows from the user FPGA towards the GECKO4COM. The timing diagram of a correct write transaction is shown in Figure A.1. A write transaction is initiated by activating the **$\overline{\text{start trans}}$** signal and sending the *Transmission Control Word (TCW)* (see Chapter 2.3 and Figure 2.2) over the **data cntrl** lines.

☞ *Important: All signals are active for one clock period of the **bus clock**.*

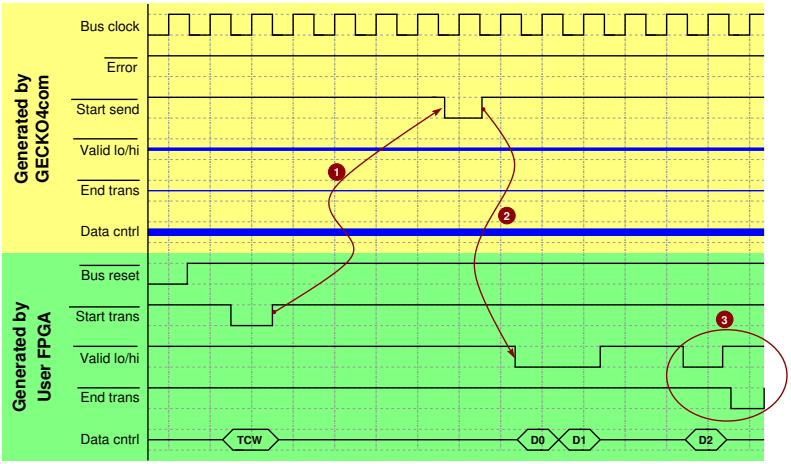

Figure A.1: A correct write transaction that causes no bus error. Here a 3-short burst transaction is shown. The blue lines represent tri-stated FPGA pins.

After the initiation of the write transaction the user FPGA has to wait for the GECKO4COM to activate the **start send** signal (dependency ❶). Sending data before this dependency will result in unpredictable results. After the reception of the **start send** signal the user FPGA may start transmitting the data payload(shown by ❷). The data payload may be a continues stream or chunk-ed into parts. For each datum the user FPGA has to put the datum on the **data cntrl** lines and activate both the **valid lo** and **valid hi** lines to indicate valid data. After the sending of the last datum, the user FPGA has to end the transaction by activating the **end trans** signal as shown at ❸. The **end trans** signal may be activated after or in parallel with the sending of the last datum of the payload.

Important: the GECKO4COM *does not check whether or not the user FPGA sends the correct number of data. It assumes that the user FPGA does send the number of data as announced in the Transmission Control Word (TCW).*

A.2 Write transaction aborts

Write transactions can be aborted under the conditions described in Chapter 2.2. An aborted transaction is indicated by the GECKO4COM by the activation of the **error** line. This condition can occur anywhere during the transaction. Figure A.2 and Figure A.3 show two examples of aborted write transactions. After the reception of the activated **error** signal the user FPGA is required to activate the **end trans** signal to end the transaction (as shown in dependency ❹).

Important: Failing to adhere to dependency ❹ may leave the bus in an undefined state.

A.3 Read transactions

Simular to the write transaction a read transaction is initiated by activating the **start trans** signal and sending the *Transmission Control Word (TCW)* (see Chapter 2.3 and Figure 2.2) over the **data cntrl** lines. As during the read transaction the data has to flow from the GECKO4COM to the user FPGA the controlling of the bi-directional signals, shown in Table 2.1,

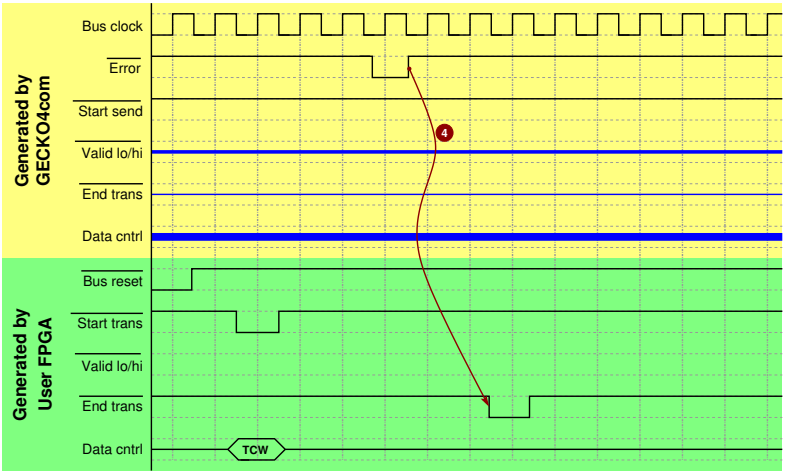

Figure A.2: An aborted write transaction before the reception of the **start trans** signal.

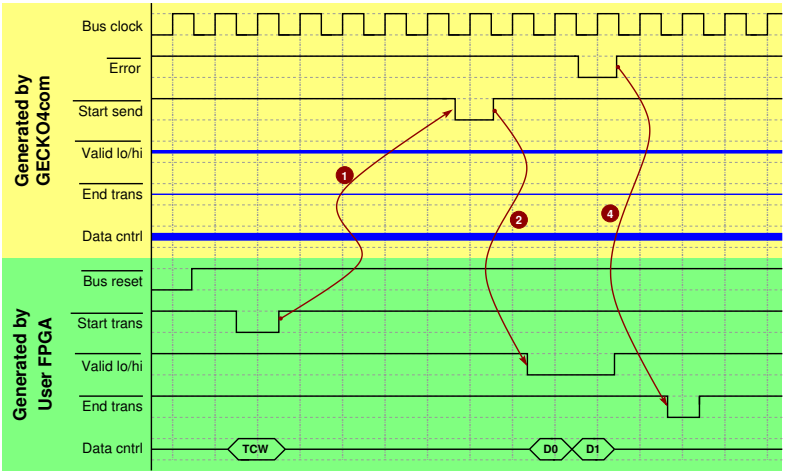

Figure A.3: An aborted write transaction after the reception of the **start trans** signal.

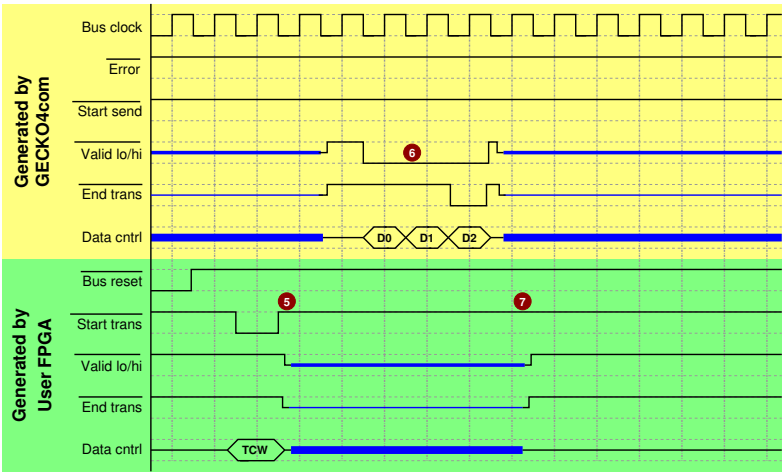

Figure A.4: A correct read transaction that causes no bus error. Here a 3-short burst transaction is shown. The blue lines represent tri-stated FPGA pins.

need special attention. Figure A.4 depicts a none-aborted read transaction for a burst of three. One cycle after the initialization of a read transaction (❺) the user FPGA has to tristate all bi-directional signals. The user FPGA has to keep the bi-directional signals in tristate up to one cycle after receiving an activated **end trans** signal (❼).

☞ *Important: Failing to adhere to dependency ❺ and dependency ❼ may destroy the IOB buffers of either or both the user FPGA and the* GECKO4COM *FPGA!*

Two cycles after the reception of a read transaction (❺) the GECKO4COM will start driving the bi-directional signals. The GECKO4COM will send the requested number of shorts in one continues burst (❻) over the **data cntrl** lines and activates the **valid low** and **valid low** as described in Chapter 2.3. During the transmission of the last datum of the read transaction the GECKO4COM ends the transmission by activation of the **end trans** signal. The **bus clock** cycle after the activation of the **end trans** signal the GECKO4COM puts all the bi-directional signals in three-state.

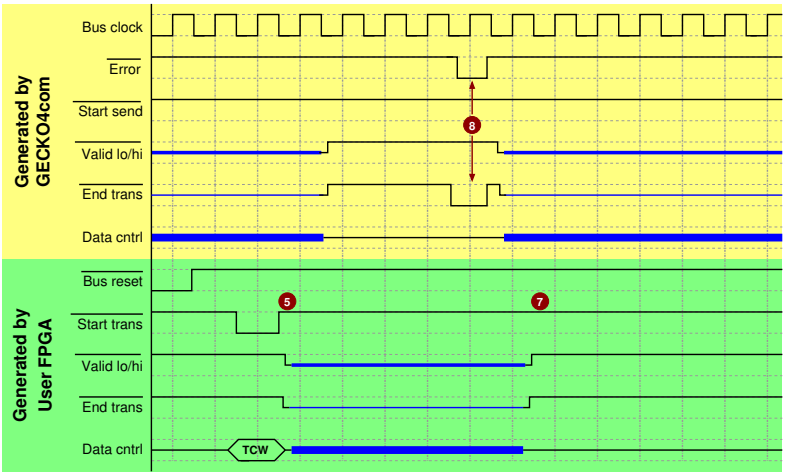

Figure A.5: An aborted read transaction. The blue lines represent tri-stated FPGA pins.

☞ *Important: The user FPGA has to make sure it has enough buffer capacity to store the requested amount of data, as there is no way to interrupt the data flow coming from the* GECKO4COM.

A.4 Read transactions abort

Read transactions can be aborted under the conditions described in Chapter 2.2. An aborted transaction is indicated by the GECKO4COM by the activation of the **error** line together with the activation of the **end trans** signal (❽). Figure A.5 shows an aborted read transaction.

B GECKO4COM IO

This appendix describes the different IO components and the connection diagrams.

B.1 VGA screen and message windows

Figure B.1 show the VGA screen produced by the GECKO4COM. In Figure B.1 the three different text screens are marked in brown.

Figure B.1: The VGA screen as produced by the GECKO4COM. The different message windows are marked in brown.

B.2 Buttons, switch and LEDs

Figure B.2 shows the GECKO4COM's part of the GECKO4MAIN. The different buttons, the switch and the bicolor LEDs are marked.

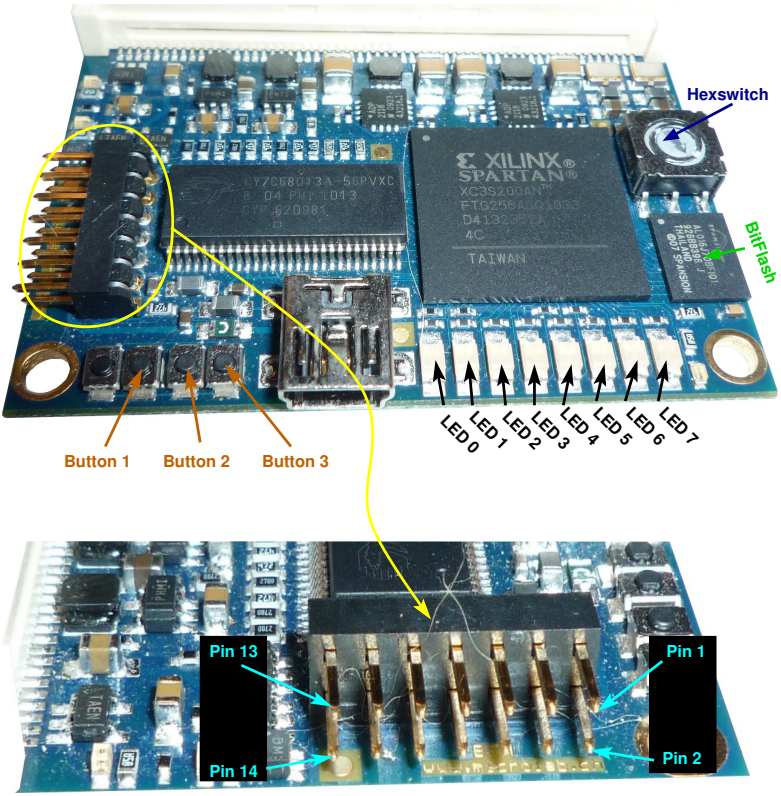

Figure B.2: The buttons, the switch and the LEDs of the GECKO4COM.

B.3 VGA connection and RS232

The bottom of Figure B.2 shows the GECKO4MAIN's dual function connector and its pin numbering. This connector can be used to connect a VGA-screen and a RS232 connector. Table B.1 lists the pin-numbers of this connector, the corresponding signal names and their connection on a female VGA connector, respectively their connections on a female RS232 connector. All signals not listed in Table B.1 should be left unconnected.

Note: The 3.3V power supply can be used to supply level translators for the RS232 communication (for example the MAX232).

Dual function pin	Signal name	VGA pin
1	Ground	5,10
3	Vertical Sync	14
4	Horizontal Sync	13
10	Blue	3
12	Green	2
13	Analog Ground	6,7,8
14	Red	1
Dual function pin	**Signal name**	**RS232 pin**
2	3.3V power supply	
7	RS232 RxD	3
9	RS232 TxD	2
11	Ground	5

Table B.1: Connection diagram of the dual function connector to a VGA connector and a RS232 connector (9-pol. Zero Modem connection)