CENTIPEDE-PCI CNC Interface Board
Mach3 Version
Firmware V2.0.0

KSI Labs, LLC

2011

COPYRIGHT

Disclaimer

Information in this manual is protected by copyright laws and is the property of **KSI Labs, LLC**. Changes to specification and features in this manual may be made by **KSI Labs, LLC** at any time without prior notice. For product-related information and latest versions check our web site:

http://www.ksilabs.com

Contents

Chapter 1

HARDWARE

CENTIPEDE-PCI is a PCI-2.2 compatible interface card for CNC control applications. It may be used for other purposes as well but it was designed to control a CNC machine when used with optional breakout board, CENTIPEDE-BRK. It is universal PCI Add-In card i.e. it will work in both 3.3V and 5V PCI/PCI-X slot.

CENTIPEDE-PCI (simply PCI from now on) board is designed in the way that allows for maximum flexibility and extensibility. Entire functionality is implemented in infinitely reprogrammable Altera MaxII © CPLD so all board's hardware is actually an HDL code. That code can be changed and programmed into the CPLD thus allowing for easy bug fixes, new functionality addition, and ultimately for making it into something totally differerent. All CPLD source code (in VHDL) is available for free from KSI Labs, LLC web site. New releases, extensions, user-contibuted add-ons etc. will be also available from our web site as well as precompiled CPLD images (*.pof files) that can be directly written into the CPLD.

All external connections are made through a single 100-pin connector. There are 32 galvanically isolated dedicated inputs, 32 Open Drain dedicated outputs, 2 LVDS inputs and 10 LVDS outputs (all galvanically isolated.) LVDS I/O is supposed to implement SPI-like interfaces to external devices with up to 1MHz clock rate. Some of them used for communication with DAC/ADC peripherals on the breakout board (BRK from now on,) some are free for future use with add-on extension boards.

ALL external I/O is designed as galvanically isolated; there is no provision for a direct galvanic connection to the PCI board. Open Drain outputs are supposed to drive optocouplers so no ground connection is provided. There is +5V power from the external connector and external devices' optocouplers are connected between that +5V power and OD outputs. In the similar fashion all input optocouplers' LED anodes are connected together and routed to the external connector for connection to *external* +5V power and their cathodes are supposed to be connected to that *external* 5V ground to pass a signal to the PC.

LVDS I/O is also galvanically isolated with ISO72xx digital isolators. Those have 2 separate power supplies; the PCI board side is powered from the PC and connected to CPLD pins while the external side is powered from external 5V power. There is no galvanic connection between those two sides.

8 of 32 digital inputs use high speed FOD053L optocouplers with 1 μS propagation time for time-critical signals. Remaining 24 use regular MOCD207M devices with 3 μS propagation time.

1.1 Connectors

1.1.1 JTAG

Connector ***J2*** in the center of the board is a standard Altera JTAG connector for programming the CPLD. It is fully compatible with standard Altera programming tools (ByteBlasterII, USB-Blaster etc.) This connector is keyed to prevent from inserting the programming tool plug a wrong way. Other than that there is nothing more to say about it. CPLD programming is fully documented in QuartusII Web Edition software available for free from Altera web site. There is a brief programming procedure description in *"Setting up CENTIPEDE board set for Mach3"* document. Here is the schematics fragment with JTAG connector:

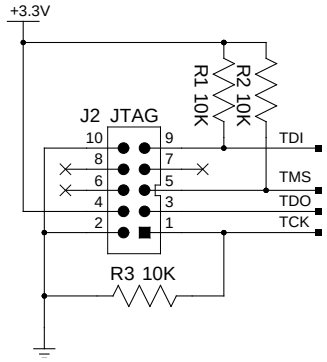

Figure 1.1: JTAG Connector

1.1.2 External Connector

External connector is high quality 100-pin N102A0-52E2PC connector from 3M. All I/O signals come from this connector. Special cable is used for connecting PCI to BRK board. It is 1:1 cable and it is made to order when ordering PCI/BRK boards from **KSI Labs, LLC**. Cable length can be up to 20 ft. according to the customer's specification.

Here is the external connector pinout and signals description:

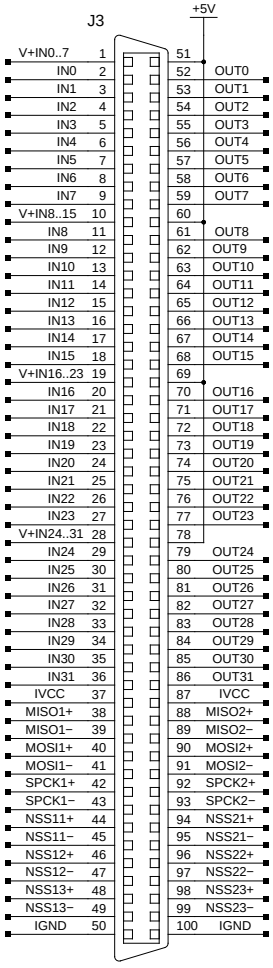

Figure 1.2: External Connector

IN0..31 Input optocouplers cathodes. Connect them to external power negative (or ground) rail to set input to '1' level. IN0 is bit 0 of DATA_IN register.

V+IN0..7-V+IN24..32 Input optocouplers anodes in groups of 8. They should be connected to the positive rail of an external power supply. There are series 330 Ω resistors installed for each optocoupler so there is no need for additional resistors if 5V external power is used. Those are usually powered from BRK board 5V power supply. 4 separate pins are used to spread the load between 4 wires.

OUT0..31 Open Drain outputs for driving external optocouplers. There is ***NO*** common ground connection on the external connector from PCI board

version 1.1 and up. These outputs ***MUST*** drive external optocouplers only to provide galvanical isolation. They should be connected to external optocouplers LED cathode. Anodes of those LEDs should be connected to +5V outputs (pins 51,60,69,78.) Series resistors are required for external optocouplers. They should be designed for 5V operation. OUT0 is bit 0 of DATA_OUT register.

+5V Power for external optocouplers on OUT0..31 lines. This is taken directly from PCI bus and should ***NOT*** be used for anything else (there is no ground connection anyway.) There is 4 pins for this power to spread the load between 4 wires. OUT0..31 OD outputs make ground connection for this power rail when corresponding DATA_OUT register bits set to '1.'

MISO1+/- SPI1 data input differential pair. Not accessible directly from application software.

MOSI1+/- SPI1 data output differential pair. Not accessible directly from application software.

SPCK1+/- SPI1 clock differential pair. Not accessible directly from application software.

NSS1[1..3]+/- SPI1 chip select differential pairs. Not accessible directly from application software.

MISO2+/- SPI2 data input differential pair. Not accessible directly from application software.

MOSI2+/- SPI2 data output differential pair. Not accessible directly from application software.

SPCK2+/- SPI2 clock differential pair. Not accessible directly from application software.

NSS2[1..3]+/- SPI2 chip select differential pairs. Not accessible directly from application software.

IVCC External +5V power for machine side SPI interface. Usually connected to BRK board 5V power supply.

IGND External 5V power supply ground for machine side SPI interface. Usually connected to BRK board ground.

1.1.3 Extension Connector

Extension connector is used for connecting add-on boards. It is ***NOT*** isolated from PC so add-on boards implementing external interface to a machine must provide their own galvanic isolation.

There is no particular add-on interface implemented for this connector as of right now (firmware v.2.0.0) but it is going to change in near future when

add-on boards are out. There is at least one such board in early design stage as of v.2.0.0 firmware time. This interface is *NOT* software accessible right now and should have some VHDL code added to use it.

Here is the connector pinout and signals description:

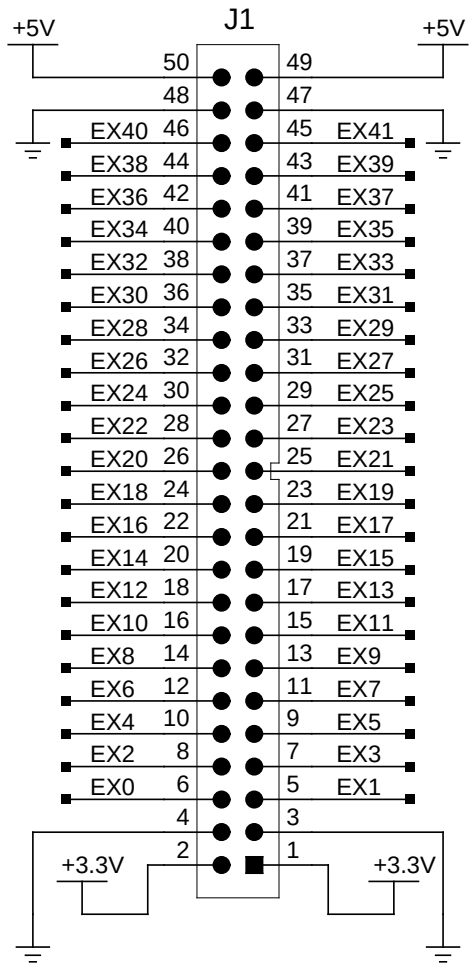

Figure 1.3: Extension Connector

EX0..41 CPLD signals. EX0..41 lines are connected to CPLD pins so they can be used for any purpose with custom VHDL additions.

Power Power rails are self-explanatory. They are supposed to power add-on boards. +5V, +3.3V, and ground all come directly from PCI bus.

Chapter 2

FIRMWARE

CENTIPEDE-PCI board is a set of different buffers/optocouplers/isolators and one big CPLD. The entire functionality is implemented in the CPLD; all other components are just simple interface components not implementing any logical functions. That means that functionality may by radically changed by programming CPLD with different firmware. Such design allows for easy hardware bugfixes, almost unlimited flexibility, new functionality addition by simply reprogramming the CPLD etc.

It is necessary to understand that CPLD code implements ***HARDWARE*** unlike some code for an embedded microcontroller that implements *FIRMWARE*. The principal difference is that CPLD code is actually a table of interconnects between different basic hardware blocks that CPLD is made of. In other words it is a bunch of wires and instructions where each wire is connected. Microcontroller firmware, on the other hand, is a *PROGRAM* i.e a set of instructions that microcomtroller fetches from memory and executes one-by-one. That means that firmware is always slower because every action is usually a sequence of instructions. Another firmware disadvantage is that MCU (MicroController Unit) can only execute a single instruction at a time (actually there are MCUs that are able of executing several instructions at a single step but that is not a regular case and there are other limitations) so it can not act fast enough on several different tasks, it can get into an infinite loop in one execution branch and all other tasks will get suspended indefinitely and there are other issues with such approach. In CPLD, on the other hand we can implement several different units that are working in parallel totally independent of each other.

There is another fundamental difference between CPLD and MCU–there is no program running in CPLD. The interconnection table is loaded from internal FLASH-like memory only once (usually on powerup) and then it is pure hardware operation. It is not bootup like it is in MCU where the initial program is loaded and then that program executes for the entire time the MCU is operating; it is a one-time ***CONFIGURATION*** that only executes once.

That does not mean that CPLD can not execute some action sequences but those sequences are purely hardware ones. Machine gun also performs some

sequence of actions when trigger is pulled but there is no software program in it. MCU counts pulses by incrementing some variable while CPLD implements it as a string of triggers changing their states.

Yet another difference is that unlike MCU CPLD does not have its pin functions preassigned to particular pin (except a few such as power/ground pins or JTAG pins for initial programming.) Almost any signal can be connected to almost any CPLD pin upon initial configuration that makes PCB design much easier because one can reassign signals to different pins if it makes PCB layout easier. There are some limitations of course but they are not all that strict.

There are different ways to make that CPLD (Complex Programmable Logic Device) configuration table. One can use a pure schematic approach by drawing schematics with special CAD software and then it is translated to the particular CPLD device configuration image by special "compiler." This is the most precise way but it is cumbersome and not actually suitable for bigger and more complex designs. Another way is using some kind of HDL (Hardware Description Language) that describes how the hardware is supposed to operate. Then such a description is processed by a set of CAD tools that synthesize a schematic implementation of the described behavior. This way is easier to work with, better suited for big complex designs, more maintainable and more portable between different devices. Here in KSI Labs, LLC we use one of HDL languages, VHDL for CPLD design. The entire VHDL source for CENTIPEDE-PCI board CPLD is available for free from our web site so everybody can customize our board as he sees fit and/or change/extend its functionality.

So strictly speaking "FIRMWARE" is not a right word to call the CPLD configuration but we will be using it for the lack of better one.

This chapter does NOT describe how to configure the CENTIPEDE set of boards for use with particular software (e.g. Mach 3;) it is the description what the board looks like from a programmer's standpoint so he can write his own software for it. Please note that "firmware" may change at every moment so please visit our web site, `http://www.ksilabs.com` for the latest information.

The included information describes Mach3-specic firmware version 2.0.0 that is programmed in CENTIPEDE-PCI boards as they shipped. There is also a Generic GPIO firmware available for this board from our web site that does not have any particular use as of time of writing but can be used for any task by writing an appropriate software for it. There is no particular purpose of writing that firmware but it is released in full binary and source form as a service for the public. It can be used for any type of control applications and much more.

2.1 PCI Device Registers

CENTIPEDE-PCI is a regular PCI board fully conforming to PCI 2.2 standard. It has one 4Kbyte PCI Memory BAR that is initialized for proper system memory mapping by PC BIOS. VendorID is 0xFEFF ***(That might be changed to KSI Labs, LLC VendorID when it is obtained.)*** DeviceID is 0x0002, ClassCode 0x078000. All register offsets are from BAR0 base. Registers are

32-bit and ***MUST*** be accessed with 32-bit instructions. Write operations other that 32-bit will have unpredictable results and probably will lead to faulty operation and computer crash.

Here is the register map:

Register	Offset	Description
DATA_OUT	0x00	Output Data Register, R/W
DATA_IN	0x04	Input Data Register, R/O
DAC	0x08	DAC Data Register, W/O
ADC	0x0c	ADC Data Register, R/W
IRQ_RAW	0x10	Raw IRQ Status Register, R/C
IRQ_MASK	0x14	IRQ Mask Register, R/W
IRQ_STAT	0x18	Masked IRQ Status Register, R/O
MACH_IDX_X	0x1c	Mach3 Axis X Index, R/W
MACH_IDX_Y	0x20	Mach3 Axis Y Index, R/W
MACH_IDX_Z	0x24	Mach3 Axis Z Index, R/W
MACH_IDX_A	0x28	Mach3 Axis A Index, R/W
MACH_IDX_B	0x2c	Mach3 Axis B Index, R/W
MACH_IDX_C	0x30	Mach3 Axis C Index, R/W
MACH_CONFIG	0x34	Mach3 Configuration Register, R/W
MACH_CONFIG2	0x38	Mach3 Second Configuration Register, R/W
MACH_CTL	0x3c	Mach3 Control Register, R/W
MACH_STAT	0x40	Mach3 Status Register, R/C
MACH_FIFO	0x44	Mach3 FIFO Register, W/O
MACH_ENC	0x48	Mach3 Encoders Count Register, R/C
MACH_SPCNT	0x4c	Mach3 Spindle Count, R/O
MACH_SPCTL	0x50	Mach3 Spindle Control, R/W

Table 2.1: CENTIPEDE-PCI/Mach3 Register Map

2.2 Registers Description

2.2.1 DATA_OUT, Offset 0x00

D31	D30	D29	D28
D27	D26	D25	D24
D23	D22	D21	D20
D19	D18	D17	D16
D15	D14	D13/SPSTEP	D12/SPDIR
D11/CSTEP	D10/CDIR	D9/BSTEP	D8/BDIR
D7/ASTEP	D6/ADIR	D5/ZSTEP	D4/ZDIR
D3/YSTEP	D2/YDIR	D1/XSTEP	D0/XDIR

Table 2.2: DATA_OUT Register

D0..31 Output Data, R/W. All data written to this register is immediately reflected on External Connector OUT0..31 pins. '1' will make the output FET to open i.e. the output will go ***LOW***. In other words writing '1' to a particular bit will turn the corresponding external optocoupler LED ON. Read operation will give the current actual output register state. D0..D11 outputs are physically disconnected from output pins when corresponding Axis (XYZABC) is enabled in MACH_CONFIG register. In this case you can still read/write the corresponding bits but the actual output pins are controlled by FIFO outputs so such R/W operations will not have any effect on the actual physical outputs. Defaults to all '0' on powerup.

xDIR Direction outputs for x Axis (XYZABC) when that Axis is enabled in MACH_CONFIG register. In this case the corresponding DATA_OUT register outputs are not connected to anything. You can write to those bits and you will read back what you wrote but they are physically disconnected from the output pins.

xSTEP Step outputs for x Axis (XYZABC) when that Axis is enabled in MACH_CONFIG register. In this case the corresponding DATA_OUT register outputs are not connected to anything. You can write to those bits and you will read back what you wrote but they are physically disconnected from the output pins.

SPDIR/SPSTEP Direction and Step outputs for Spindle drive when Dir/Step Spindle is enabled in MACH_CONFIG2 register. In this case the corresponding DATA_OUT register outputs are not connected to anything. You can write to those bits and you will read back what you wrote but they are physically disconnected from the output pins.

2.2.2 DATA_IN, Offset 0x04

D31/ENC4_B	D30*/ENC4_A	D29/ENC3_B	D28*/ENC3_A
D27/ENC2_B	D26*/ENC2_A	D25/ENC1_B	D24*/ENC1_A
D23*/SP_IDX	D22/SP_TIMING	D21*	D20
D19*/ESTOP_SW	D18/PROBE_SW	D17*/CHome	D16/CMinus
D15/CPlus	D14/BHome	D13/BMinus	D12/BPlus
D11/AHome	D10/AMinus	D9/APlus	D8/ZHome
D7/ZMinus	D6/ZPlus	D5/YHome	D4/YMinus
D3/YPlus	D2/XHome	D1/XMinus	D0/XPlus

Table 2.3: DATA_IN Register

D0..D31 Input Data, R/O. IN0..31 External Connector data state directly from connector pin latched on every PCI clock (through an optocoupler.) The same rule as for DATA_OUT register applies–'1' means there is current through the input optocoupler LED (switch connected to the corresponding BRK board terminal is CLOSED,) '0' means optocoupler LED is OFF. Bits marked with '*' have fast optocouplers on inputs on rev.1.0 CENTIPEDE-PCI board. Starting from rev.1.1 fast optocouplers are on bits 24..31. Input changes are reflected immediately. Raw state for all optional signals (xHome etc.) can be read from corresponding Dxx bits if needed. Raw means it is *ELECTRICAL*, not *LOGICAL* state i.e. it only tells if there is current through an optocoupler or not. That can be interpreted by the user software (Driver) and hardware as either activated or not depending on "Negated" setting in appropriate configuration registers or in software configuration.

xPlus/xMinus Plus and minus side Limit Switch inputs for x Axis (XYZABC) when the corresponding Axis is enabled in MACH_CONFIG register. Those perform a specific *Hardware* action when the corresponding axis is enabled. They stop all motion and activate EStop hardware state. IRQ is also generated when one of these is activated (if enabled.)

xHome Home Switch inputs for x Axis (XYZABC) when the corresponding Axis is enabled in MACH_CONFIG register. MACH_IDX register for corresponding axis is frozen when this signal is activated so exact hit position can be read. Actual Index Counters are *NOT* affected so the current position information is not lost. MACH_IDX registers are transparent latches that mirror Index Counters all the time but latch the last reading when Probe or Home Switch is hit thus giving the exact hit position. Once read they resume their normal operation if the corresponding switch is not active or not enabled any more. Home Switch freezes only the corresponding axis Index while Probe Switch freezes *ALL* Indexes. IRQ is

generated when one of these switches is activated so the Driver software can take appropriate action immediately.

PROBE_SW Digital Probe Switch input for probing. Its state can be read from a corresponding bit but it also performs some hardware actions when activated. ***ALL*** MACH_IDX registers are frozen when this signal is activated so exact hit position can be read. Actual Index Counters are *NOT* affected so the current position information is not lost. MACH_IDX registers are transparent latches that mirror Index Counters all the time but latch the last reading when Probe or Home Switch is hit thus giving the exact hit position. Once read they resume their normal operation if the corresponding switch is not active or not enabled any more. Home Switch freezes only the corresponding axis Index while Probe Switch freezes *ALL* Indexes. IRQ is generated when this switch is activated so the Driver software can take appropriate action immediately.

ESTOP_SW Emergency Stop Switch input. That can ***NOT*** be disabled or reconfigured to other polarity (not "Negated") and it ***MUST*** be a normally closed switch. If it is *OPEN* i.e. there is no current through its optocoupler hardware will be in EStop hardware mode and there is ***NO*** way to get it out of this state until this switch is closed. In other words this switch is ***ABSOLUTELY*** required and it ***MUST*** be a "Push-To-Break" type.

SP_TIMING Multiple pulse per revolution Spindle rotation sensor input. This is used for reporting Spindle RPM and for Z Axis motion control when lathe threading or rigid tapping. On every transition on this input (rising or faling depending on configuration bit in MACH_CONFIG2 register) the internal counter contents is transferred to MACH_SPCNT register, counter is reset, and then incremented on each and every PCI clock pulse. IRQ is generated on SP_TIMING transition so software can read MACH_SPCNT register and act appropriately by calculating actual Spindle RPM and displaying it or adjusting Z Axis speed thus gearing it to the Spindle. There is also a debouncing counter that is loaded with a value set in MACH_CONFIG2 register and started decrementing every 15 μS on a transition at this input. Input state is ignored until this counter counted down to zero. This cycle repeats on all consecutive transitions, both low-to-high and high-to-low.

SP_INDEX Single pulse per revolution Spindle Index sensor. This is used in lathe threading to start Z Axis motion at exactly the same spot on each consecutive threading pass. This is done in hardware by holding step output until SP_INDEX pulse detected. To do this software should write '1' in MACH_CTL register WAIT4PULSE bit. This will put all step output (except Spindle) on hold and will restart it when SP_INDEX is detected. FIFO will be still taking step data while this hold is in effect until it is full. Then, when Index pulse comes the very first FIFO

entry will be output immediately. This input also performs all the functions of SP_TIMING if that is not enabled. There is a separate debouncing counter for SP_INDEX working in exactly the same fashion as SP_TIMING. IRQ is also generated on each SP_INDEX pulse.

2.2.3 DAC, Offset 0x08

X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
D15	D14	D13	D12	D11	D10	D9	D8
D7	D6	D5	D4	D3	D2	D1	D0

Table 2.4: DAC Register

D0..15 SPI Transmit Holding Register (THR.) Once written data is transferred bit-by-bit to TLV5617A 2-channel 0-10V output DAC on the BRK board. Each write to DAC register clears corresponding bit in IRQ_RAW register and that bit goes back to '1' once that written data is transferred to the DAC. This register is Write-Only. Please read TLV5617A datasheet and Mach3 Driver source to learn how to use this register. Please note that despite only 16 bits are used all writes ***MUST*** be done as 32-bit instructions.

X Don't care.

2.2.4 ADC, Offset 0x0c

X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	X	X	D9	D8
D7	D6	D5	D4	D3	D2	D1	D0

Table 2.5: ADC Register

D0..9 SPI Transmit Holding Register (THR) on write, Receive Holding Register (RHR) on read. This is a second SPI controller, separate from DAC SPI. Once written data is transferred bit-by-bit to TLV1544 4-channel 0-10V input ADC on the BRK board. At the same time ADC data is received into RHR. Each write to ADC register clears corresponding bit in IRQ_RAW register and that bit goes back to '1' once that written data is transferred to the ADC (and ADC data is received into RHR.) Reading ADC register (RHR) does not have any effect on SPI controller and IRQ state, only write starts a new SPI cycle. Please read TLV1544 datasheet

and Mach3 Driver source to find out how to use this register. Please note that despite only 10 bits are used all reads and writes ***MUST*** be done as 32-bit instructions.

X Don't care on write, '0' on read.

2.2.5 IRQ_RAW, Offset 0x10

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	MACH_SP_TIM	MACH_SP_IDX	MACH_HomeC
MACH_HomeB	MACH_HomeA	MACH_HomeZ	MACH_HomeY
MACH_HomeX	MACH_PROBE	MACH_FOVF	MACH_FAULT
MACH_DONE	MACH_TMR	ADC	DAC

Table 2.6: IRQ_RAW Register

DAC DAC SPI Controller is Idle and ready to take new data to be sent to TLV5617A DAC on the BRK board. This bit is always set when DAC SPI Controller is Idle. Write data to its THR and write '1' to this bit to reset it. It will get set again when transfer is complete.

ADC ADC SPI Controller is Idle and ready to take new data to be sent to TLV1544 ADC on the BRK board. This bit is always set when ADC SPI Controller is Idle. Write data to its THR and write '1' to this bit to reset it. It will get set again when transfer is complete.

MACH_TMR Mach3 timer interrupt. Periodic interrupt every 5th cycle when the board in is RUN condition. Cycle period is determined by CYCLE setting in MACH_CTL register. Write '1' to this bit to clear.

MACH_DONE All data in FIFO were sent out and last cycle has ended so board got idle. Write '1' to this bit to clear.

MACH_FAULT Fault interrupt. Raised when any of Limit Switches hit, or Emergency Stop (EStop from now on) button is activated. Cleared by writing '1' in this bit if the condition that caused it is cleared. Will not reset if condition is still present. FIFO is purged on this event, periodic timer stays ticking.

MACH_FOVF FIFO overflow. Theoretically should not happen because hardware ignores all FIFO writes when it is full but anyways... Write '1' to clear.

MACH_PROBE Probe Switch hit. Write '1' to clear. Will not reset if the switch is still activated. All axes indexes are frozen on this event until they are read, PROBE_HIT bit in MACH_STAT register is written with '1,' and this IRQ is acknowledged by writing '1' to its bit. The freeze is done by stopping transparent latches between actual index counters and IDX registers so they keep the last data while index counters continue counting so the running position is not lost.

MACH_HOMEx Home Switch for x Axis (XYZABC) is hit. Write '1' to clear. Will not reset if the switch is still activated. The corresponding axis index is frozen on this event until it is read, corresponding HOME_HIT bit in MACH_STAT register is written with '1,' and this IRQ is acknowledged by writing '1' to its bit. The freeze is done by stopping transparent latches between actual index counters and IDX registers so they keep the last data while index counters continue counting so the running position is not lost.

MACH_SP_IDX Spindle Index Pulse detected. This is single pulse per revolution Index sensor that is used for starting lathe threading pass. It doubles at Spindle Timing if no Timing sensor installed. Write '1' to this bit to clear.

MACH_SP_TIM Spindle Timing Pulse detected. This is multiple pulse per revolution Timing sensor that is used for measuring Spindle RPM for Mach DRO and gearing Z Axis to the Spindle while threading or rigid tapping. See DATA_IN register for Spindle Index and Timing operation description. Write '1' to clear.

2.2.6 IRQ_MASK, Offset 0x14

X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X
X	MACH_SP_TIM	MACH_SP_IDX	MACH_HomeC
MACH_HomeB	MACH_HomeA	MACH_HomeZ	MACH_HomeY
MACH_HomeX	MACH_PROBE	MACH_FOVF	MACH_FAULT
MACH_DONE	MACH_TMR	ADC	DAC

Table 2.7: IRQ_MASK Register

X Don't care. Those bits can be written to and read back but they don't have any effect for now. They might be used in the future when additional functionality is implemented and add-on boards are out.

Rest Remaining bits are used to mask corresponding IRQ_RAW interrupt bits. Default to all '0' on powerup so no IRQs to PCI bus is generated.

2.2.7 IRQ_STAT, Offset 0x18

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	MACH_SP_TIM	MACH_SP_IDX	MACH_HomeC
MACH_HomeB	MACH_HomeA	MACH_HomeZ	MACH_HomeY
MACH_HomeX	MACH_PROBE	MACH_FOVF	MACH_FAULT
MACH_DONE	MACH_TMR	ADC	DAC

Table 2.8: IRQ_STAT Register

0 Unused, read as '0.'

Rest Remaining bits are a result of bitwise AND on IRQ_RAW and IRQ_MASK registers. PCI IRQ signal is an OR on all bits of this register. That means that it is only raised when the corresponding bit is set in both IRQ_RAW and IRQ_MASK. All bits are read-only. To reset (acknowledge) a particular IRQ '1' should be written to a bit in IRQ_RAW register.

2.2.8 MACH_IDX_[XYZABC], Offsets 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30

D31	D30	D29	D28	D27	D26	D25	D24
D23	D22	D21	D20	D19	D18	D17	D16
D15	D14	D13	D12	D11	D10	D9	D8
D7	D6	D5	D4	D3	D2	D1	D0

Table 2.9: MACH_IDX_[XYZABC] Registers

D0..31 Current Axis position, R/W. Incremented/Decremented on each and every step output to the Axis depending on step direction. Signed (2-complement) 32-bit INT. Actual counters are connected to these registers through transparent latches. Those latches are open most of the time so registers show the current positions. They are frozen when Home or Probe Switch event occurs so the exact hit position can be read. To unfreeze them the Switch Hit event should be cleared. This is done by

writing '1' to a corresponding bit in MACH_STAT register. To prevent this condition from reoccuring if a switch is still activated one can first disable that switch in MACH_CONFIG register and then write '1' to MACH_STAT bit.

2.2.9 MACH_CONFIG, Offset 0x34

PROBE_EN	PROBE_NO	C_HOME_EN	C_HOME_NO
C_LIMS_NO	C_DIR_REV	C_EN	B_HOME_EN
B_HOME_NO	B_LIMS_NO	B_DIR_REV	B_EN
A_HOME_EN	A_HOME_NO	A_LIMS_NO	A_DIR_REV
A_EN	Z_HOME_EN	Z_HOME_NO	Z_LIMS_NO
Z_DIR_REV	Z_EN	Y_HOME_EN	Y_HOME_NO
Y_LIMS_NO	Y_DIR_REV	Y_EN	X_HOME_EN
X_HOME_NO	X_LIMS_NO	X_DIR_REV	X_EN

Table 2.10: MACH_CONFIG Register

x_EN Axis x (XYZABC) enable. Enables corresponding axis hardware. Axis x(Plus,Minus,Home) signals are connected to the corresponding DATA_IN bits, x(DIR/STEP) outputs switched from DATA_OUT bits to corresponding FIFO outputs, Limit Switches logic is enabled so MACH_FAULT IRQ will be generated.

x_DIR_REV Axis x (XYZABC) Direction Signal will be inverted. In Mach3 '0' means "move to PLUS side." That is what is stuffed in FIFO for each and every Axis. Actual drives might treat direction signals differently. To accomodate all of them DIR output is inverted if these bits are set to '1.' It is only the actual physical output line that is inverted, all internal signals are still the same and MACH_IDX registers are *Incremented* on every step when corresponding DIR bit is '0.'

x_LIMS_NO Axis x (XYZABC) Limit Switch is Normally Open (NO.) Usually it is recommended that Limit/Home/Probe/EStop switches ***BREAK*** the connection when activated, i.e. they are Normally Closed (NC) until activated. That ensures the machine will be saved from damage even if the switch cable is cut or otherwise disconnected. If the switch is NO its activation will not be detected if it is disconnected and the machine can be damaged. If it is necessary to use NC switches this bit should be set to invert the default logic. It can also be used to make the switch signal always inactive if there is no particular switch on the machine instead of shorting that pin (all axes signals are fixed in hardware and there is no way to disable a Limit Switch or use its pin for something else if an

Axis is enabled.) There is only *ONE* LIMS_NO bit per axis so *BOTH* of switches must be either NO or NC, mix is not allowed.

x_HOME_NO Axis x (XYZABC) Home Switch is Normally Open. Everything from x_LIMS_NO applies here.

x_HOME_EN Axis x (XYZABC) Home Switch Enable. To avoid unnecessary interrupts and to provide for resetting HOME_HIT conditions Home Switches are only enabled briefly during Homing cycles. These are per-axis Enable bits.

PROBE_NO Probe Switch is Normally Open. Everything written about NO/NC switches above applies.

PROBE_EN Probe Switch Enable. It is only enabled when probing, disabled all other time to provide for PROBE_HIT condition reset and eliminame unnecessary interrupts.

2.2.10 MACH_CONFIG2, Offset 0x38

SP_TIM_INV	SP_TIM_EN	TIM_DB[8..7]	
TIM_DB[6..3]			
TIM_DB[2..0]			IDX_DB[9]
IDX_DB[8..5]			
IDX_DB[4..1]			
IDX_DB[0]	SP_DIR_INV	SP_STEP_INV	SP_DIRSTEP
SP_IDX_INV	SP_IDX_EN	C_STEP_INV	B_STEP_INV
A_STEP_INV	Z_STEP_INV	Y_STEP_INV	X_STEP_INV

Table 2.11: MACH_CONFIG2 Register

x_STEP_INV Axis x Step signal Inverted. Usually Step Signal is output as a pulse that turns the drive Step input optocoupler *ON*. Some drives may require inverted signal. These bits invert STEP outputs when set, per axis.

SP_IDX_EN Spindle Index signal Enable.

SP_IDX_INV Spindle Index signal is Inverted i.e. turns CENTIPEDE optocoupler *OFF* to indicate Index pulse.

SP_DIRSTEP Enables Dir/Step Spindle when set.

SP_STEP_INV Spindle Step signal Inverted i.e. drive Step input optocoupler *ON to OFF* transition is used to make a step.

SP_DIR_INV Spindle Dir signal Inverted. Set this if your Spindle rotates in the wrong direction.

IDX_DB[0..9] Spindle Index signal debounce time in 15 µS units. Index signal state is ignored for this number of 15 µS intervals after initial transition. Write '0' to disable debouncing.

TIM_DB[0..8] Spindle Timing signal debounce time in 15 µS units. Index signal state is ignored for this number of 15 µS intervals after initial transition. Write '0' to disable debouncing.

SP_TIM_EN Spindle Timing signal Enable.

SP_TIM_INV Spindle Timing signal is Inverted i.e. turns CENTIPEDE optocoupler *OFF* to indicate Timing pulse.

2.2.11 MACH_CTL, Offset 0x3c

WAIT4PULSE	FIFO_PURGE	X	X
X	X	X	X
X	X	X	X
X	X	X	X
MACH_CYCLE[10..7]			
MACH_CYCLE[6..3]			
MACH_CYCLE[2..0]			RESET
RUN	NO_LIMITS	ESTOP_REQ	SIMULATE

Table 2.12: MACH_CTL Register

SIMULATE When set to '1' motor control outputs are kept at '0' state, no Steps output. Everything else works as usual so this is the most precise Dry Run simulation possible.

ESTOP_REQ Programmatic Emergency Stop (EStop.) Stops the CENTIPEDE FSM, purges FIFO. Board goes in EStop state; only periodic timer IRQs are generated.

NO_LIMITS Makes the board to ignore any of Limit Switches. Normally, when this bit is not set CENTIPEDE immediately goes into EStop state, FIFO purged, FSM stopped, FAULT IRQ generated, only periodic timer is ticking. Board will NOT go out of EStop until the condition is removed i.e. machine is moved out of Limit Switches. Setting this bit overrides this behaviour thus allowing to jog off the switch.

RUN Setting this bits allows the Finite State Machine (FSM) to run so the actual Mach3 work can be done.

RESET Setting this bit generates board RESET signal – most registers are set to all zero, FSM stopped, FIFO purged, periodic timer is also stopped so no IRQs. This is self-clearing bit because reset sets all the registers to their initial state (almost always all zeroes) thus resetting this very bit among others.

MACH_CYCLE[0..10] Mach3 Cycle Period, 11 bit. This field determines kernel frequency or Engine frequency as it is called in Mach3. Every Mach3 cycle will be this number of PCI clocks long. PCI Clock is roughly 30 nS that corresponds to standard 33.333 MHz PCI Frequency (the exact value is measured by the Mach Driver.) For 100 KHz, e.g. the value is 333 (33.3 MHz / 333 = 100 KHz.)

FIFO_PURGE Writing '1' to this bit will purge the FIFO – FIFO pointer (internal) will be set to the first element, i.e. FIFO will become empty. This bit is not sticky i.e. action is only taken once on the write operation and its value is not saved. That means there is no need to write '0' there after writing '1.'

WAIT4PULSE Writing '1' to this bit will make CENTIPEDE put all step data on hold until Spindle Index Pulse. This is used for lathe threading to start each consecutive thread pass at exactly the same spot. This bit is not sticky i.e. action is only taken once on the write operation and its value is not saved. That means there is no need to write '0' there after writing '1.'

X Don't care.

Please note that there is no Step Pulse settings any more. It did not make sence to have them at all because *ALL* stepper/servo drives we are aware of only state *MINIMUM* pulse width. They all make step on Step input *TRANSITION*, either low-to-high or high-to-low. That is why CENTIPEDE step generation has changed.

Now we generate step pulses as transitions from inactive to active state exactly in the middle of each Mach Cycle period and resetting it back to inactive at the very end of that cycle. The exact meaning of "active" depends on x_STEP_INV bits in MACH_CONFIG2 register. All Step signals are exactly one Mach Cycle wide with actual Step transition happening at the middle of it. In Mach terms (Motor Tuning dialog) that means that both Dir and Step Pulse are ***ALWAYS*** half of Mach Cycle period i.e. half of Kernel Frequency period. For 100KHz Kernel Frequency they are both 5 μS. Mach3 settings are ignored by the driver.

That allowed to simplify our hardware implementation a little bit thus freeing CPLD resources for other purposes and gave us some additional benefits. One of those is that step is *ALWAYS* taken at the middle of Mach Cycle that reduces step jitter when lathe threading or rigid tapping.

2.2.12 MACH_STAT, Offset 0x40

RUNNING	ESTOP	BUSY	SP_STOPPED
X	X	FIFO_FULL	FIFO_EMPTY
FIFO_ROOM[4..1]			
FIFO_ROOM[0]	ProbeHit	CHomeHit	CMinusHit
CPlusHit	BHomeHit	BMinusHit	BPlusHit
AHomeHit	AMinusHit	APlusHit	ZHomeHit
ZMinusHit	ZPlusHit	YHomeHit	YMinusHit
YPlusHit	XHomeHit	XMinusHit	XPlusHit

Table 2.13: MACH_STAT Register

xPlusHit Axis x (XYZABC) Plus side Limit Switch hit. This is *NOT* a switch state but the indicator it's been hit. This event generates FAULT interrupt and forces board into EStop. Writing '1' to these bits will clear the condition if the Limit Switch is no longer activated. If the switch is still active it will have no effect unless NO_LIMITS bit is set in MACH_CTL register.

xMinusHit Same as above for Minus side Limit switch.

xHomeHit Axis x (XYZABC) Home Switch hit. Freezes the IDX register for the affected axis, stays set until cleared with writing '1' to a particular bit. Such a write will have no effect if condition still exists. Recommended action is read the frozen value (hit point,) disable the switch in MACH_CONFIG, reset with writing '1.' Usually Home Switches are only enabled briefly on Homing forward pass and disabled right after the switch is hit. Resetting a particular bit unfreezes the corresponding axis IDX register so it starts giving the actual position.

FIFO_ROOM[0..4] Current free space left in FIFO. Used by the driver to make a decision on how many steps it can push in FIFO.

FIFO_EMPTY Set when FIFO is empty. Indicator bit, R/O. Resets automatically if there is at least one entry in FIFO and its cycle is not finished.

FIFO_FULL No more room in FIFO. R/O, self-resets when the first entry is popped.

SP_STOPPED Spindle stopped. Either Dir/Step Spindle is stopped (if enabled) or no Spindle Index/Timing pulses detected for more than 8 seconds.

BUSY CENTIPEDE is busy i.e. FIFO is not empty and there are still cycles to do. R/O, self-clearing.

ESTOP Board is in EStop state – Limit Switch hit, EStop button activated, connection to BRK board lost, or ESTOP_REQ bit set in MACH_CTL register. Writing '1' to this bit will clear EStop if the condition that forced CENTIPEDE in that state is no longer active. Write will have no effect if condition persists. Will not go away by itself when condition removed; writing '1' to reset is required.

RUNNING Indicator bit, R/O meaning the board is up and running. Running means board was configured properly and RUN bit set in MACH_CTL. EStop does NOT affect this bit – board can be running but in EStop condition. '0' in this bit means FSM is stopped and board is not active, even the periodic timer is stopped.

X Don't care.

2.2.13 MACH_FIFO, Offset 0x44

X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X
X	X	X	X	CStep	CDir	BStep	BDir
AStep	ADir	ZStep	ZDir	YStep	YDir	XStep	XDir

Table 2.14: MACH_FIFO Register

xDir Axis x (XYZABC) Direction for current step. '0' means move to the Plus direction. FIFO is 20 entries deep. That number is chosen because Mach3 outputs Planned Motion points in sets of 5 so FIFO is made a multiple of 5. The first entry written to FIFO will be automatically output to the motors by CENTIPEDE hardware on the next cycle. Then it will remove it automatically and pop the next one from FIFO until no more entries left. It pops one entry each CENTIPEDE cycle. Periodic timer interrupt is generated every 5th cycle so the driver ISR can load another set of 5 steps to FIFO.

xStep Axis x (XYZABC) Step signal. Step pulse will be generated if it is '1'. Everything from xDir also applies here. As usual only 32-bit writes should be used. This is a W/O register, reads are not guaranteed to return any particular value.

2.2.14 MACH_ENC, Offset 0x48

ENC4_CNTR[7..0]
ENC3_CNTR[7..0]
ENC2_CNTR[7..0]
ENC1_CNTR[7..0]

Table 2.15: MACH_ENC Register

ENCx_CNTR[7..0] 8-bit counter for Encoder x (1..4.) Signed (2-complement) CHAR. Increments/Decrements on Quadrature Encoder rotation depending on direction. Registers are R/O and they are reset to zero after each read. That is done to save on CPLD resources – the main 32-bit counter is in the software driver and these 8-bit counters are used as "Change from last read." Every periodic timer tick driver reads all these counters and adds them to the main 32-bit counters. Decoder logic for quadrature encoders is all-hardware, x4 multiplying i.e. every transition is detected so the actual resolution is 4 times of encoder stated one. That means that e.g. 2500 CPR encoder will give 10000 steps per revolution.

2.2.15 MACH_SPCNT, Offset 0x4c

X	X	X	X	SP_CNT[27..24]
SP_CNT[23..16]				
SP_CNT[15..8]				
SP_CNT[7..0]				

Table 2.16: MACH_SPCNT Register

SP_CNT[0..27] Spindle Count. Number of PCI Clock pulses between last 2 Spindle Timing (or Spindle Index if Timing is not enabled) pulses. Read-Only.

X Don't care.

2.2.16 MACH_SPCTL, Offset 0x50

X	X	X	X	FAKE_PULSE	SP_RUN	SP_DIR	SP_DIV[24]
SP_DIV[23..16]							
SP_DIV[15..8]							
SP_DIV[7..0]							

Table 2.17: MACH_SPCTL Register

SP_DIV[0..24] Dir/Step Pulse Frequency Divisor. Steps are generated by dividing PCI clock. This is a value that determines resulting step frequency e.g. Dir/Step Spindle RPM. Every time internal counter decrements to zero Step output is inverted and counter is reloaded with SP_DIV value (this is all done by the hardware; SP_DIV is stored in a register when written so it only needs to be written once for particular RPM.) For a full step pulse 2 such transitions needed so this value must be calculated to get a frequency twice the desired RPM multiplied by steps per rotation. E.g. if we want to get 300 RPM from a motor with 1000 steps per revolution we should feed it with 300(RPM) / 60(sec/min) $*$ 1000(steps/rev) = 5 KHz pulse rate. We need 2 transitions per step so we should program our divisor for 5 KHz $*$ 2 = 10 KHz frequency. PCI clock is usually 33.333MHz so to get 10 KHz we should use 33.333(MHz) / 10(KHz) = 3333 divisor value.

SP_DIR Dir signal for Dir/Step Spindle.

SP_RUN Go signal. Set this to '1' to actually start the Spindle, set to '0' to stop it.

FAKE_PULSE This bit imitates Spindle Timing pulse. It is only used on Driver startup to measure actual PCI clock frequency. This is Write-Only bit.

X Don't care. No effect on write, read back as '0.'

Please note that there is no Step Pulse settings. It did not make sence to have them at all because *ALL* stepper/servo drives we are aware of only state *MINIMUM* pulse width. They all make step on Step input *TRANSITION*, either low-to-high or high-to-low. Our Step Pulses are actually a constant 50% Duty Cycle square wave. This is generated in hardware without any need for sotware intervention once SP_DIV is programmed and SP_RUN is set. Software only has to intervene when Spindle RPM or some other state (e.g. Direction or Run/Stop) needs to be changed.

Chapter 3

Appendix A - Hardware differences between rev.1.0 and rev.1.1

There are some differences between rev.1.0 and rev.1.1 CENTIPEDE-PCI boards. They are not all that significant but there some caveats.

- First of all, there were 4 unpopulated footprints for optional oscillator on rev.1.0 board (elements R4, C15, U2, and R5) close to the boards center. Those were removed completely from rev.1.1 PCB. There is absolutely no impact because those elements were never used.
- Rev.1.0 board has 4 out of 8 fast optocouplers on different input lines (see section 2.2.2.) It is not very likely somebody would use that feature but if he would one should take care to assign data lines properly with application software (e.g. configuration editor or whatever it's called) for his particular board revision.
- As a result of the above CPLD project file differs between those 2 revisions. VHDL code is absolutely identical between the two but there are some changes in pin assignments in centipede_pci.qsf file and resulting centipede_pci.pof file is different. Changes are minimal but please be careful anyway. All future CPLD firmware will be released in versions for both rev.1.0 and rev.1.1 PCI boards. There is no differences between those revisions from software point of view.
- ***IMPORTANT!*** Rev.1.0. PCI board had PCI bus ***GROUND*** connected to pins 69 and 78 of the External Connector, NOT +5V power. It had 2 +5V power pins (51 and 60) and 2 GROUND pins (69 and 78.) On rev.1.0 CENTIPEDE-BRK board pins 69 and 78 of the mating connectors are ***NOT CONNECTED*** to anything so it works OK with rev.1.0

CENTIPEDE-PCI board. Rev.1.1 PCI board will also work just fine with rev.1.1 BRK board though 2 +5V conductors in the cable will not be used. ***But rev.1.0 PCI board can NOT be used with rev.1.1 BRK board because the latter has pins 51, 60, 69, and 78 all connected together that will make a short circuit between PC +5V and ground.***

- Other than that those boards absolutely identical and fully compatible with each other.