PMC-CPU/405

PowerPC 405GPr PrPMC module

User Manual

Product Order No. V.2020.02, Hw-Rev. 2.x

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft.

esd übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

esd hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

The information in this document has been carefully checked and is believed to be entirely reliable.

esd makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document.

esd reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

---

esd electronic system design gmbh

Vahrenwalder Str. 207

D-30165 Hannover

GERMANY

Tel.: +49-511/372 98-0

FAX : +49-511/372 98-68

E-Mail: info@esd.eu

www.esd.eu

Dokumenten Information

document-no.:	V.2020.21
document type:	DOC08xx
document status:	released
document revision:	2.0
date of creation:	2011-09-30
document path:	I:\Prj\esd\PMC\pmc405\Doku\
document filename:	PMC-CPU405_Manual_en_2.0.odt
number of pages / annex:	44 / 0

Responsible for content / author

Name	Company / Department	Phone	Email
M. Fuchs	esd gmbh / SD	+49 511 37298 0	support@esd-electronics.com

Distribution / Review

Name	Firma / Abteilung	Telefon	Email
-	-	-	-

Modification history

Technical modifications in this overview are marked by an additional "**!**".

!	Revision	Chapter	Page	Changelog	Datum / Name
	2.0	-	-	first released manual version of 'PMC-CPU/405 hardware rev. 2.0'	2011-09-30 MF/TB

Contents

1 Overview

1.1 Description Of The PMC-CPU/405 Module

The PMC-CPU/405 is a PMC module in ‘single’ PCI Mezzanine Card form factor. It can act as PMC monarch (PrPMC) or as non-monarch (adapter/target) board. Apart from a powerful CPU core the PowerPC 405GPr embedded processor integrates a SDRAM controller, a PCI bus interface, a controller for serial interfaces and a 10/100 Mbit/s EMAC.

The module comes with SDRAM and flash memory, a double layer capacitor buffered realtime clock (RTC) and a FPGA. Two serial interfaces are designed as RS-232 ports. One can be accessed via a D-sub9 connector in the front panel. The second serial interface is accessible via the PMC-I/O connector. The two high speed CAN interfaces are controlled by SJA1000 CAN controllers. They are suitable for transmission rates up to 1 Mbit/s. The TTL-signals of both controllers are led to the PMC-I/O connector. The Ethernet interface is suitable for 10 Mbit and 100 Mbit networks. It is accessable via an RJ45-socket in the front panel. The LEDs in the front panel show the current status of the PMC-CPU/405 module.

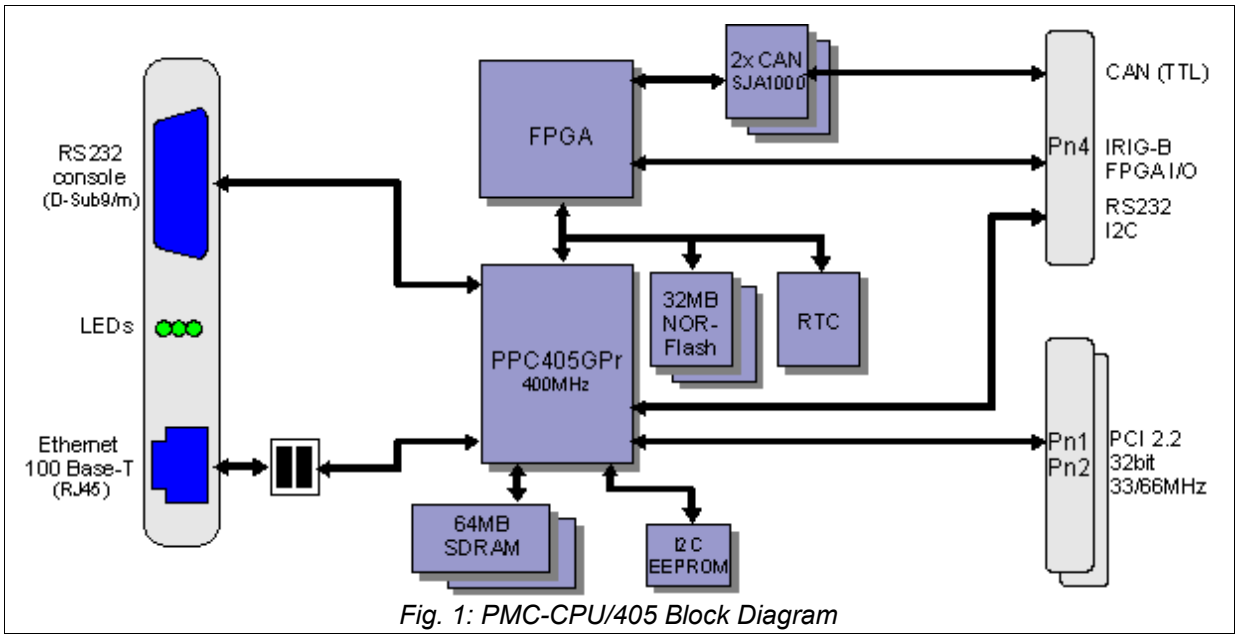


Fig. 1: PMC-CPU/405 Block Diagram

The following overview lists feature enhancements and improments against PMC-CPU/405 version 1.x boards:

- CPU Upgrade: Version 2.x hardware is equipped with an AMCC PowerPC 405GPr running at 400MHz.
- Monarch (PrPMC) / Non-Monarch switching: Version 1.x was available as a Monarch and a Non-Monarch version. Since version 2.0 both configurations will be supported with the same hardware. The MONARCH# signal is used to switch between these modes.
- IRIG-B timecode receiver/transmitter (timecode format: B100)
- The 1.x version's CPLD is replaced by a FPGA to allow sophisticated features like CAN-timestamps and time synchronisation through IRIG-B.
- The frontpanel's RUN- and CAN-LED can also be used as general purpose indicators („IRIG-B synchronized“ etc.)

- On-board EEPROM and Flash can be write protected by hardware.
- The unused pins from the D-sub9 frontpanel serial port can optionally be used as digital differential IRIG-B input and output. The input is electrically isolated. This is an option.
- The CPU can be hard reset (e.g. over the PCI bus) through a GPIO signal. This makes it possible to reset a PMC405 adapter card.
- A PCI interrupt (INTA#) can alternatively be asserted through a GPIO. This fixes some MS Windows™ specific issues with interrupt handling.
- The bootloader (U-Boot) supports firmware loading over the PCI bus. This feature has been backported to the 1.x boards
- The I/O signals on P4 will be supplemented by further signals:
  - I2C bus (master on PMC405)
  - differential digital IRIG-B input + output
  - general purpose TTL-level I/Os with special alternative functions (CLOCK/RESET-based timestamp syncronisation, TTL-level IRIG-B)

1.2 Technical Data

1.2.1 General

Operating temperature	0...50 °C
Humidity	max. 90%, non-condensing
Power supply	5V and 3.3V DC, ±5%
Power consumption	~ 3 W (main powerrail is 3.3V)
PCB formfactor	148.33mm x 74.04mm
weight	~ 100g

1.2.2 CPU Core

CPU	PPC405GPr, AMCC
CPU clock	400MHz
RAM	64MB, SDRAM (max. 128MB)
Flash memory	32MB NOR flash (2 banks of 16 MB, Intel Strata flash)
EEPROM	2kB connected via I2C (used for bootloader configuration)
Watchdog	CPU internal, 4 selectable intervals: 5ms, 83ms, 1.3s, 21s

1.2.3 Realtime Clock (RTC)

Type	DS1685
NVRAM	242 bytes
Backup energy source	Double layer capacitor
Backup time	Up to 1 week

1.2.4 PCI Interface

Specification	PCI 2.2 compatible
Features	- host or target - bus master - 2 target address spaces (BARs)
PCI clock	33/66 MHz
Bus width	32 bit
IO voltage	3.3V, 5V tolerant
Interrupt	non-monarch: INTA# monarch: INTA#..INTD#

1.2.5 Serial Interfaces

Number	2
Controller	CPU internal, 16550 compatible
Physical Layer	RS232
Lines	RxD, TxD, CTS, RTS
Connector	Port0: D-sub9, male on front panel Port1: PMC P4
Baudrate	Max. 115200 baud

1.2.6 CAN Interfaces

Number	2
CAN controller	SJA1000
CAN protocol	CAN 2.0A/2.0B
Physical interface	TTL, no transceiver on-board
Bitrate	10 kbit/s ... 1 Mbit/s
Bus termination	None
Connectors	PMC P4

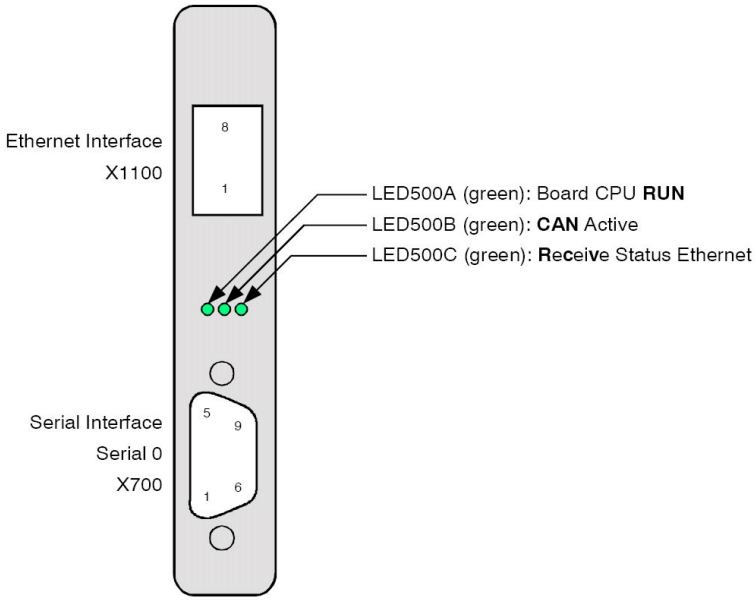
1.2.7 Ethernet Interface

Number	1
Standard	IEEE 802.3, 10/100BaseT
Bitrate	10/100Mbit/s
Controller	CPU internal
Isolation	transformer
Connector	RJ45 socket in frontpanel
MAC-address	00:02:27:83:00:00 + serial#

2 Frontpanel

The PMC-CPU/405 frontpanel provides access to the Ethernet interface and the serial RS232 console via a D-sub9 (male) connector. Three LEDs indicate several status information:

LED designator	Color	Function
LED500A	green	CPU RUN indicator. Is LED is lit or flickering when the CPU is accessing its SDRAM. Typically this means that the CPU is running.
LED500B	green	CAN traffic indicator. This LED is flickering on any traffic on one of the two CAN interfaces. This LED can also be controlled by software when CAN traffic indication is not needed.
LED500C	green	Ethernet receive status. This LED is flickering on received network data.



2.1 Frontpanel Connector Pinouts

Connector type: D-sub9, male

Pin	Signal Name	Signal Name	Pin
1	IRIG-B_F_INP (DCD)	RxS0	2
3	RxS0	IRIG-B_F_P (DTR)	4
5	GND	IRIG-B_F_INM (DSR)	6
7	RTSS0	CTSS0	8
9	IRIG-B_F_M (RI)		-

Note 1: Signalnames in brackets are standard RS232 signal names for these pins. They are *not* provided by the PMC-CPU/405. The alternative functions pinning (IRIG signals, see below) is chosen based on the most compatible bahavior. When connecting a full blown RS232 device (e.g. modem)

pin 9 (RI) must not be connected because it is an output of the PMC-CPU/405 *and* the modem.

Note 2: The frontpanel IRIG-B signals are optional. The necessary parts on the PCB are not installed by default.

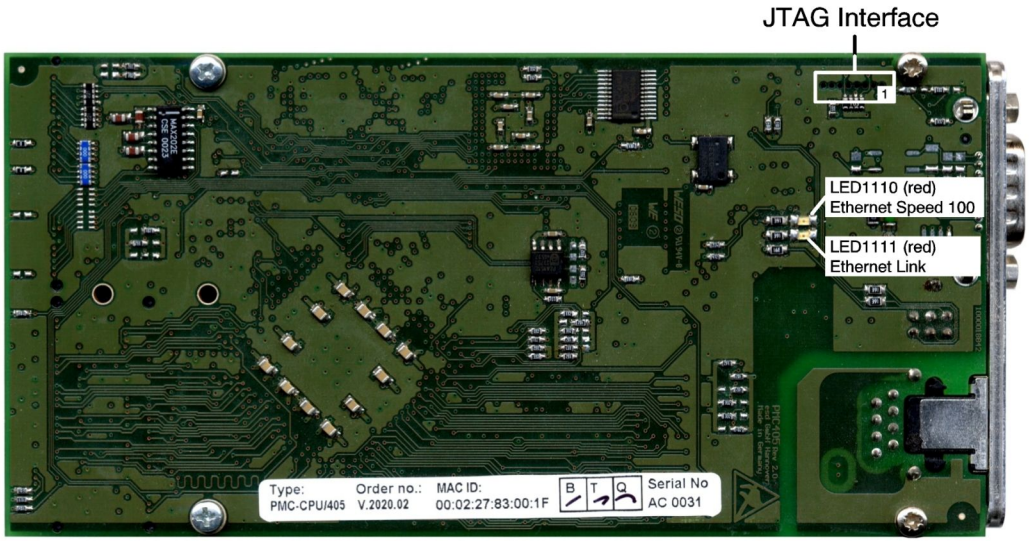
2.2 Frontpanel Connector Signal Description

Signal Name	Direction	Description
RxS0	IN	RS232 receive data, 1st port
TxS0	OUT	RS232 transmit data, 1st port
RTSS0	OUT	RS232 request to send, 1st port
CTSS0	IN	RS232 clear to send, 1st port
IRIG-B_F_INP	IN+	RS422 pseudo differential IRIG-B input, electrical isolated. The output of the onboard differential receiver is connected to an FPGA pin.
IRIG-B_F_INM	IN-	
IRIG-B_F_P	OUT+	RS485 differential IRIG-B in/output. The input of the onboard differential buffer is connected to an FPGA pin. The RS485 driver can also be disabled and IRIG-B_F_P/M signal pair can be used as differential input. This functionality is equal to a physical RS485 half duplex interface.
IRIG-B_F_M	OUT-	
GND	GND	Ground

3 Bottom Side

The PMC-CPU/405 has two additional LEDs on the bottom side of the PCB. These LEDs are visible when the PMC module is installed on a carrier board.

LED designator	Color	Function
LED1110	red	Ethernet speed indication. This LED is lit when the ethernet interface runs at 100Mbit/s.
LED1111	red	Ethernet link indicator. This LED is lit when a link on the ethernet interface is established.



4 PPC405GPr GPIO Functions

The following table shows the usage of the 405GPr's GPIO usage on the PMC-CPU/405.

GPIO	Mode	Name	GPIO	Mode	Name	GPIO	Mode	Name
1	STR	STRAPPING15	9	STR	STRAPPING22	17	IRQ	IRQ-C0#
2	STR/OUT	STRAPPING16/ RESETOUT#	10	CS	LCS1#	18	IRQ	IRQ-C1#
3	STR/OUT	VPEN	11	CS	LCS2#	19	IRQ/(OUT)	INTA#
4	STR	STRAPPING19	12	CS	LCS3#	20	IRQ	INTB#
5	OUT	FPGA-PROG	13	IN	MONARCH#	21	IRQ/(OUT)	INTC#/SELFRST#
6	OUT	FPGA-CLK	14	IN/(OUT)	EREADY	22	IRQ	INTD#
7	IN	FPGA-DIN	15	IN	FPGA-DONE	23	IRQ	IRQ-FPGA#
8	STR/OUT	WPEN	16	IN	FPGA-INIT#	24	STR	STRAPPING21

STR=CPU strapping, *OUT*=output, *IN*=input, *IRQ*=interrupt input, *CS*=peripheral chip select

Name	Function
VPEN	Intel Strata Flash programming enable: 1=programming enable, 0=programming disabled
WPEN	EEPROM write protection: 1=EEPROM is write protected, 0=EEPROM no write protected
FPGA-PROG	Used to boot the on-board Xilinx-FPGA in slave serial mode
FPGA-CLK	
FPGA-DIN	
FPGA-DONE	
FPGA-INIT#	
LCS1# ... LCS3#	GPIO pins that are used as chip selects for flash and memory mapped peripherals (CAN, RTC, FPGA).
MONARCH#	PMC signal. This signal indicates whether the PMC405 is used as monarch or non-monarch board.
EREADY	PMC signal. EREADY is an input in monarch mode and an output in non-monarch mode.
INTA# ... INTD#	PCI/1PMC interrupt lines. These signals are interrupt inputs in monarch mode. I
INTA#	INTA# is used as PCI-INTA# output in non-monarch mode. This provides an alternative method to generate PCI interrupts for non-monarch boards by asserting/deasserting the corresponding bit in the 405's GPIO0_TCR register.
INTC# / SELFRST#	SELFRST# is used as a hard „self-reset“ output in non-monarch mode. Assertion of SELFRST# results in a reset of the PowerPC 405 CPU.
IRQ-FPGA#	This GPIO is setup as interrupt input. It can be used by the FPGA to generate interrupts to the CPU.
RESETOUT#	Driving low this GPIO asserts the PMC RESETOUT# signal.

5 JTAG Debug Interface

The debug port is used during manufacturing tests and to flash an initial firmware.

5.1 JTAG Chain Description

All JTAG capable circuits on the PMC405 module are connected to a common JTAG chain in the order as fllows:

TDI -> **CPU** -> **FPGA** -> **CPLD** -> **Ethernet Phy** -> TDO

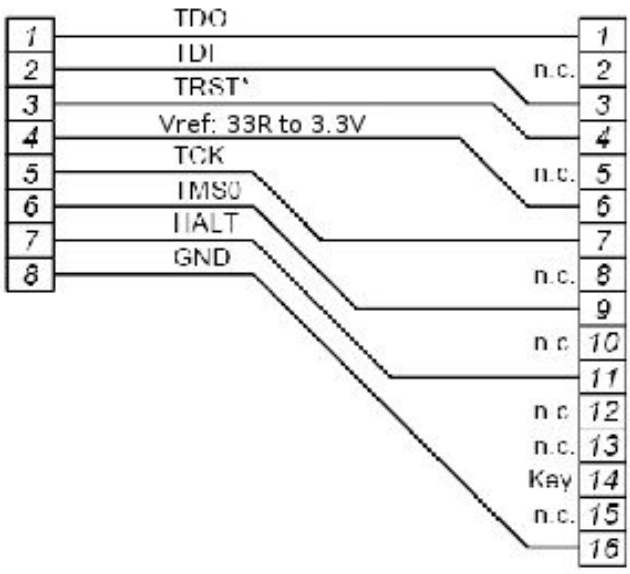
Device position in chain	Function	Device part number	JTAG IR length
1	CPU	PPC405GPr (AMCC)	7
2	FPGA	XC2S200E-FT256 (Xilinx)	5
3	CPLD	XC9536XL-VQ44 (Xilinx)	8
4	Ethernet Phy	LXT971A (Intel, Cortina Systems)	16

5.2 JTAG Connector

The JTAG interface can be accessed through a 8-pin single in-line 1,27mm plug. It is possible to connect to the JTAG interface even when the PMC405 module is assembled on a carrier system. Drillings in the PCB allow interfacing the JTAG port from the solder side of the module. It can be connected via a SMD-pin contact strip connector. It is recommended to build a simple adapter from the SMD-pin contact strip connector to a 16-pin connector to connect to the port. This 16-pin connector is used by a wide range of third party hardware debugger vendors.

Pin Number	Function	Direction
1[1]	TDO	output
2	TDI	input
3	TRST#	input
4	Vref+ (3.3V via 100 Ohm resistor)	output
5	TCK	input
6	TMS	input
7	PPC405 HALT	input
8	GND	reference

1 Pin1 is situated close to the PMC module's frontpanel bezel. The pin 1 drilling is marked by a tiny '1' on the PCB's solder side.



Example of an Adapter (Uncased)

Connectors to build adapter:	SMD-pin contact strip:	Fa. Samtec, 'modified pin contact strip', order-no. MTMS-108-58-T-S-485
	2.54 mm post connector:	i.e. Fa. Harting, 16-pole, straight, order-no. 09185167324

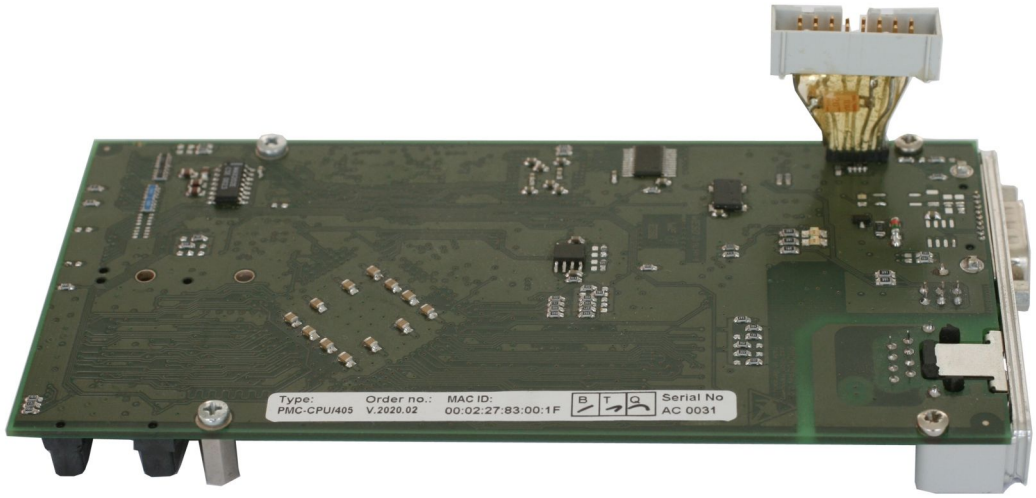


Fig. 2: self-build JTAG adapter inserted into PMC module from botton side of PCB

6 PMC-Connectors

The PMC-CPU/405 module uses the PMC connectors P1, P2 and P4. P1 and P2 provide the PCI interface and power supply connection. P4 has a complete module specific pinout.

6.1 PMC P1 Connector

Pin	Signal	Signal	Pin
1	n.c. (TCK)	-12V	2
3	GND	INTA#	4
5	INTB#	INTC#	6
7	GND (PRESENT#)	+5V	8
9	INTD#	n.c. (reserved)	10
11	GND	n.c. (reserved)	12
13	PCI-CLK	GND	14
15	GND	GNT#	16
17	REQ#	+5V	18
19	VIO	AD[31]	20
21	AD[28]	AD[27]	22
23	AD[25]	GND	24
25	GND	C/BE3#	26
27	AD[22]	AD[21]	28
29	AD[19]	+5V	30
31	VIO	AD[17]	32
33	FRAME#	GND	34
35	GND	IRDY#	36
37	DEVSEL#	+5V	38
39	GND	n.c. (LOCK#)	40
41	n.c. (SDONE#)	n.c. (SBO)	42
43	PAR	GND	44
45	VIO	AD[15]	46
47	AD[12]	AD[11]	48
49	AD[09]	+5V	50
51	GND	C/BE0#	52
53	AD[06]	AD[05]	54
55	AD[04]	GND	56
57	VIO	AD[03]	58
59	AD[02]	AD[01]	60
61	AD[00]	+5V	62
63	GND	n.c. (REQ64#)	64

6.2 PMC P2 Connector

Pin	Signal	Signal	Pin
1	+12V	n.c.	2
3	n.c.	TDO (bridged to TDI)	4
5	TDI (bridged to TDO)	GND	6
7	GND	n.c. (reserved)	8
9	n.c. (reserved)	n.c. (reserved)	10
11	MODE2#	+3.3V	12
13	PCI-RST#	MODE3#	14
15	+3.3V	MODE4#	16
17	n.c. (PME#)	GND	18
19	AD[30]	AD[29]	20
21	GND	AD[26]	22
23	AD[24]	+3.3V	24
25	IDSEL	AD[23]	26
27	+3.3V	AD[20]	28
29	AD[18]	GND	30
31	AD[16]	C/BE2#	32
33	GND	IDSELB	34
35	TRDY#	+3.3V	36
37	GND	STOP#	38
39	PERR#	GND	40
41	+3.3V	SERR#	42
43	C/BE1#	GND	44
45	AD[14]	AD[13]	46
47	M66EN	AD[10]	48
49	AD[08]	+3.3V	50
51	AD[07]	n.c. (REQB#)	52
53	+3.3V	GNTB#	54
55	n.c. (reserved)	GND	56
57	n.c. (reserved)	EREADY	58
59	GND	RESETOUT#	60
61	n.c. (ACK64#)	+3.3V	62
63	GND	MONARCH#	64

6.3 PMC P4 I/O Connector

P4 is used to interface many PMC-CPU/405 specific interfaces like the CAN buses. esd offers a PMC-PIM module with two isolated CAN physical circuits.

6.3.1 Pinout

Pin	Signal Name	Signal Name	Pin
1	n.c. (VCC)	n.c. (TX0-C0#)	2
3	n.c.	n.c. (RX0-C0#)	4
5	n.c.	n.c. (TX0-C1#)	6
7	n.c.	n.c. (RX0-C1#)	8
9	n.c.	n.c. (GND)	10
11	n.c. (GND)	n.c.	12
13	n.c.	n.c. (RxS1)	14
15	n.c. (RTSS1)	n.c. (TxS1)	16
17	n.c. (CTSS1)	n.c.	18
19	n.c.	n.c. (GND)	20
21..31	n.c	n.c	22..32
33	VCC	TX0-C0#	34
35	n.c.	RX0-C0#	36
37	n.c.	TX0-C1#	38
39	n.c.	RX0-C1#	40
41	n.c.	GND	42
43	GND	n.c.	44
45	n.c.	RxS1	46
47	RTSS1	TxS1	48
49	CTSS1	n.c.	50
51	n.c.	GND	52
53	CLOCK_IN	CLOCK_OUT	54
55	RESET_IN	RESET_OUT	56
57	IRIG-B_R_IN	IRIG-B_R_OUT	58
59	IRIG-B_R_P	CLOCK_EN	60
61	IRIG-B_R_M	RESET_EN	62
63	SDA_R	SCL_R	64

Note: Signals in brackets can optionally be connected to these pins by using 0-Ohm-resistor networks. Default connections have no brackets.

6.3.2 Signal Description

Signal Name	Direction	Description
CLOCK_IN	IN	3,3V/5V tolerant general purpose input. This signal is logically connected to a FPGA pin. CLOCK_IN can optionally be configured as a timestamp reference clock source.
CLOCK_OUT	OUT	LVTTL general purpose output. This signal is logically connected to a FPGA pin. CLOCK_OUT can optionally be configured as a timestamp reference clock output.
CLOCK_EN	OUT	LVTTL general purpose output. This signal is logically connected to a FPGA pin. CLOCK_EN can be used to enable an RS485 driver on a PMC-PIM module (application specific).
RESET_IN	IN	3,3V/5V tolerant general purpose input. This signal is logically connected to a FPGA pin. This pin can optionally be enabled to reset the FPGA internal timestamp counter.
RESET_OUT	OUT	LVTTL general purpose output. This signal is logically connected to a FPGA pin. This pin can be used to generate a defined reset pulse to other CLOCK/RESET sinks.
RESET_EN	OUT	LVTTL general purpose output. This signal is logically connected to a FPGA pin. RESET_EN can be used to enable an RS485 driver on a PMC-PIM module (application specific).
IRIG-B_R_IN	IN	3,3V/5V tolerant general purpose input. This signal is logically connected to a FPGA pin. This signal is used as IRIG-B time signal input when a TTL time signal is provided. This signal is logically connected to a FPGA pin.
IRIG-B_R_OUT	OUT	LVTTL general purpose output. This signal is reserved for future implementation of an IRIG-B time code generator.
IRIG-B_R_P	diff. I/O+	Half duplex RS485 differential IRIG-B input/output. This signal pair is used to supply the PMC405 with an external IRIG-B time signal source. This signal pair can also be configured as an output that is controlled by FPGA internal functionality (e.g. IRIG-B timecode generation).
IRIG-B_R_M	diff. I/O-	
TX0-C0#	OUT	TTL, CAN0 transmit
RX0-C0#	IN	TTL, CAN0 receive
TX0-C1#	OUT	TTL, CAN1 transmit
RX0-C1#	IN	TTL, CAN1 receive
RxS1	IN	RS232 receive data, 2nd port
TxS1	OUT	RS232 transmit data, 2nd port
RTSS1	OUT	RS232 request to send, 2nd port
CTSS1	IN	RS232 clear to send, 2nd port
SDA_R	I/O	I2C bus, pulled-up against 3,3V. The PMC405 is always the master on this I2C bus.
SCL_R	I/O	
GND	GND	Ground

Note: All output signals except the CAN signals are using 3.3V signalling. All inputs are 5V tolerant.

7 Local Memory Map

Start-Address	End-Address	Function
0x0000.0000	0x03ff.ffff	SDRAM (64MB)
0x8000.0000	0xbfff.ffff	PCI address space 0x8000.0000 – 0xbfff.ffff (size: 1GB) Note: This is the default setup by the U-Boot bootloader using the PMM0 register set. It can be changed during runtime by the operating system.
0xef00.0000	0xef00.00ff	FPGA internal registers (32bit access only)
0xef00.8000	0xef00.ffff	FPGA internal registers (reserved for custom extensions)
0xef60.0700	0xef60.071f	GPIO controller
		...
0xf000.0000	0xf000.00ff	CAN0, SJA1000
0xf000.0100	0xf000.01ff	CAN1, SJA1000
0xf000.0500	0xf000.05ff	RTC
		...
0xfe00.0000	0xffff.ffff	32MB NOR flash (last 512kB contain bootloader)

8 External Interrupt Assignment

CPU Interrupt	Ext. Interrupt Pin	Device	Configurations
25	0	CAN0	active low, level sensitive
26	1	CAN1	active low, level sensitive
27	2	PCI-INTA#	active low, level sensitive
28	3	PCI-INTB#	active low, level sensitive
29	4	PCI-INTC#	active low, level sensitive
30	5	PCI-INTD#	active low, level sensitive
31	6	FPGA	active low, level sensitive

9 PCI Configuration

The PMC-CPU/405 uses the following PCI identification:

	Monarch (PrPMC)	Non-Monarch
PMC-CPU/405 (Version 1.x)	Class/Subclasscode: 0x0b20 (processor, PPC) Vendor-ID: 0x1014 Device-ID: 0x0156 Subsystem-Vendor-ID: 0x12fe (esd gmbh) Subsystem-ID: 0x0409	Class/Subclasscode: 0x0b20 (processor, PPC) Vendor-ID: 0x1014 Device-ID: 0x0156 Subsystem-Vendor-ID: 0x12fe (esd gmbh) Subsystem-ID: 0x0408
PMC-CPU/405 (Version 2.x)	Class/Subclasscode: 0x0600 (hostbridge) Vendor-ID: 0x1014 Device-ID: 0x0156 Subsystem-Vendor-ID: 0x12fe (esd gmbh) Subsystem-ID: 0x040d	Class/Subclasscode: 0x0b20 (processor, PPC) Vendor-ID: 0x1014 Device-ID: 0x0156 Subsystem-Vendor-ID: 0x12fe (esd gmbh) Subsystem-ID: 0x040c

PCI base address register mapping:

- PCI-BAR0:
  - default: 64MB local SDRAM
  - mapping can be modified through bootloader environment variables. A minimum size of 4MB is recommended to allow firmware download to non-monarch boards via PCI. The local base can be reconfigured after OS booting during runtime.
  - BAR0 configuration must not be changed when any esd drivers (PciAccess, backplane-network-driver) are used.
- PCI-BAR1:
  - size: 16MB
  - mapping of local address space 0xef00.0000 – 0xefff.ffff
    - This means that the FPGA internal register (e.g. IRIG-B time) start at the beginning of BAR1.
  - offset 0x600700: 405's GPIO controller (used for *remote self-reset#* and *adapter interrupt control* functions)
  - offset 0x000000: FPGA internal registers (IRIG-B time, FIFOs etc.). This can be used to access any FPGA internal function through other PCI targets.
  - BAR1 configuration must not be changed when any esd drivers (PciAccess, backplane-network-driver) are used.

Please read the bootloader chapter of this manual to get detailed information on how to modify the PCI BARx configuration.

10 Bootloader

10.1 License

The PMC-CPU/405 module uses the opensource bootloader *„Das U-Boot“*. The U-Boot sourcecode is published in terms of the GNU public license (GPL) and can be downloaded from http://www.denx.de/wiki/UBoot. You can also contact esd for the full sourcecode of the bootloader for the PMC-CPU/405.

Esd electronics will take every effort to upload all PMC-CPU/405 board specific changes on the U-Boot sourcecode back to the official repository.

10.2 Configuration

U-Boot uses the last 512kB of onboard flash memory. So the binary u-boot image is written into flash at address 0xfff80000.

Note: U-Boot for version 1.x PMC405 boards uses only 256kB of flash memory starting at 0xfffc0000.

The U-Boot console is accessable via the frontpanel serial port. The compiled-in baudrate is 9600 baud. But the boards are shipped with baudrate set to 115200 baud (8N1).

After power-on you should see the bootloader startup messages being output on the serial console:

```
U-Boot 1.1.6-dirty (Sep 26 2007 - 16:35:56)

CPU:   AMCC PowerPC 405GPr Rev. B at 399.96 MHz (PLB=133, OPB=66, EBC=33 MHz)
       Internal PCI arbiter disabled, PCI async ext clock used
       16 kB I-Cache 16 kB D-Cache
Board: PMC405_GB0057 (non-monarch)
I2C:   ready
DRAM:  64 MB
FLASH: 32 MB
FPGA:  pmc405v2.ncd 2s200eft256 2006/12/12 16:54:05
Net:   ppc_4xx_eth0
PARAM: @03c00000
=>
```

10.3 Default Bootloader Environment

U-Boot uses so called environment variables to store any configuration. The full set of variables can be printed to the console by executing the *printenv* command. Variable can be modified through *setenv <name> [<value>]* and finally modification can be stored in a non-volatile memory through *saveenv*.

This is the factory default environment setup:

```
=> printenv
bootdelay=3
pciconfighost=1
serial#=PMC405_GB0057
ethaddr=00:02:27:83:00:39
...
load=tftp 100000 /tftpboot/pmc405/u-boot-V2.bin
preboot=painit
pram=4096
ethact=ppc_4xx_eth0
baudrate=115200
netmask=255.255.0.0
```

```
ipaddr=10.0.111.121
serverip=10.0.0.190
update=protect off fff80000 ffffffff;erase fff80000 ffffffff;cp.b 100000 fff80000
80000;protect on fff80000 ffffffff;reset
...
Environment size: 1696/2044 bytes
=>
```

10.4 Flash Update

The bootloader resides at the top of the PMC405 onboard flash memory. Assuming a binary size of 256kByte (0x40000) for version 1.x board the bootloader must be programmed into the flash memory starting at 0xfffc0000. Boards since version 2.x need a different bootloader image that uses 512kByte (0x80000) of flash memory and therefore needs to be programmed into flash starting at address 0xfff80000. Please check that you are using the correct bootloader image that suits to your board!

The bootloader update process can be very delicate. Any mistake may result in a board that must be shipped back to be reprogrammed by esd using a JTAG debugger.

Here are step-by-step instructions for bootloader update at the serial console. The tftp command requires a correct U-Boot network configuration. This means the the bootloader environment variables *ipaddr, netmask, gatewayip* and *serverip* must be setup according to you network.

Note: Please note the differences between the board versions!

1. Upload the new bootloader binary image into RAM (e.g. at RAM address 0x100000). The easiest way is to load the images from a TFTP-server.

```
=> tftp 100000 /tftpboot/pmc405/u-boot.bin
```

2. Remove flash write protection:

```
=> protect off fffc0000 ffffffff (for board version 1.x)
=> protect off fff80000 ffffffff (for board version 2.x)
```

3. Erase the current bootloader. **Attention: do not switch of the board from now on until a new bootloader has been written into the flash!**

```
=> erase fffc0000 ffffffff (for board version 1.x)
=> erase fff80000 ffffffff (for board version 2.x)
```

4. Copy the new bootloader into flash:

```
=> cp.b 100000 fffc0000 40000 (for board version 1.x)
=> cp.b 100000 fff80000 80000 (for board version 2.x)
```

   This command copies 0x40000 (0x80000) bytes starting at RAM location 0x100000 into flash memory starting at 0xfffc0000 (0xfff80000).

5. Enable flash write protection for bootloader address range:

```
=> protect on fffc0000 ffffffff (for board version 1.x)
=> protect on fff80000 ffffffff (for board version 2.x)
```

6. When all the above commands succeeded reset the board.

```
=> reset
```

To simplify the bootloader update process two environment variables already contain the necessary commands. These variables can be executed using the *run* command:

1. Load binary image from a TFTP-server:

```
=> run load
```

2. Unprotect, erase, flash and reset:

```
=> run update
```

Please doublecheck the content of the *load* and *update* variable before using them. This will prevent you from running into trouble.

10.5 BSP Commands

The following U-Boot BSP commands have been specially added to support the PMC-CPU/405 functions. Type *help <command>* at the bootloader prompt to get a short command reference.

10.5.1 irigb - Get / Set IRIG-B time

The irigb command will display the current IRIG-B time (IRIG-B receiver + local IRIG-B generators time). This command will change in the future, because there will only be a single IRIG-B time.

10.5.2 waitint – Wait for interrupt from PMC405 (version 2.x)

This command is only supported in monarch mode. The command searches for a non-monarch PMC405V2 board on the PCI bus. If multiple boards are available the optional 'index' parameter can be used to select a specific board. When a PMC405V2 board is found the command installs a interrupt service routine for this board and wait for an interrupt.

```
=> ? waitint
waitint [index] wait for interrupt from PA adapter 'index'
=> waitint 1
PMV405V2(1) board found at bus 0, device 7.
Handler installed for interrupt 28
```

Now it is possible to trigger the PCI interrupt from the targeted PMC405 (non-monarch) by using the 'inta' command:

```
=> inta 0
inta# asserted
```

The monarch's U-Boot console will output the following message and reset the PCI interrupt in the peer board:

```
Interrupt! (reg=17021000)
Got interrupt!
=>
```

10.5.3 inta – Assert / Deassert PCI interrupt line on PMC405

This command is only supported in non-monarch configuration. See 'waitint' description for details.

10.5.4 ebctest – Test external bus peripherals

This command tests the external bus peripherals (CAN0, CAN1, RTC, FPGA). The test will stop automatically on the first error or when CTRL-C is hit.

10.5.5 fifo – Control Hardware FIFO Module

The 'fifo' command is used to demonstrate and debug the FPGA internal hardware FIFOs. The full functionality of the FIFO module is described in a separate chapter.

```
=> ? fifo
fifo wait
fifo read
fifo write fifo(0..3) data [cnt=1]
fifo write address(>=4) data [cnt=1]
  - without arguments: print all fifo's status
  - with 'wait' argument: interrupt driven read from all fifos
  - with 'read' argument: read current contents from all fifos
  - with 'write' argument: write 'data' 'cnt' times to 'fifo' or
'address'
```

Without any arguments the 'fifo' command will output a short summary about the four FIFOs state:

```
=> fifo
fifo level status

_____________________________
 0       0  EMPTY
 1       0  EMPTY
 2       0  EMPTY
 3       0  EMPTY
```

The command 'fifo read' will dump all FIFOs' content until the FIFOs are empty. The command 'fifo wait' will do the same but using an interrupt. The 'fifo wait' will dump FIFO data forever until it is interrupted by pressing CTRL-C.

Finally the 'fifo write' command can be used to write data to a FIFO's data port. The FIFO data port can be addressed through giving its physical address or its number. The first way can be used to access a FIFO data port over the PCI bus while the 2nd method only works for a local FIFO access:

```
=> fifo write 84000080 12345678 3
writing 3 x 12345678 to fifo port at address 84000080
=> fifo write 2 aa5555aa 2
writing 2 x aa5555aa to fifo 2
=> fifo
fifo level status

_____________________________
 0       0  EMPTY
 1       0  EMPTY
 2       2  NOT EMPTY
 3       0  EMPTY
=>
```

10.5.6 loadpci – Start PCI firmware loading

This will initialize the PCI firmware download. A rotating cursor will indicate this mechanism. Press CTRL-C at the serial console to abort it. In most setups 'loadpci' will be added to the 'bootcmd' variable:

```
=> printenv bootcmd
bootcmd=run vxworksargs\;loadpci
```

See PCIAccess Driver manual for details about the 'loadpci' command.

10.5.7 fpga command / fpgadata Variable

Since version 2.x the PMC-CPU/405 uses an onboard Xilinx Spartan IIE FPGA (XC2S200E-FT256) to provide some special functionality. The U-Boot bootloader provides a statically linked FPGA design with some default functionality. This image is programmed into the FPGA just after reset.

It is possible to disable or replace the default image. Two steps are necessary to load an alternative

image into the FPGA after reset:

1. Programm the alternative FPGA bitstream into flash memory (e.g. at address 0xfff00000). The bootloader supports the Xilinx native binary bitstream format (.bit file) or gzip compressed .bit files. The latter are much smaller.
2. Set the bootloader environment variable „fpgadata“ to the address where the FPGA bitstream can be found in flash. When „fpgadata“ is not set, the default bitstream is used.

The alternative FPGA bitstream is loaded into the FPGA on the next reset of the board.

Step-by-step instructions to write an alternative bitstream into flash memory:

1. Upload the bitstream (uncompressed or gzip compressed) into RAM (e.g. at RAM address 0x100000). The easiest way is to load the images from a TFTP-server:

```
=> tftp 100000 /tftpboot/pmc405/pmc405v2_fpga.bit
```

2. Remove flash write protection:

```
=> protect off fff00000 fff7ffff
```

3. Erase the current flash content:

```
=> erase fff00000 fff7ffff
```

4. Copy the new bootloader into flash:

```
=> cp.b 100000 fff00000 80000
```

   This command copies 0x80000 bytes starting at RAM location 0x100000 into flash memory starting at 0xfff00000

5. Enable protection of the flash area:

```
=> protect on fff00000 fff7ffff
```

6. Set fpgadata variable:

```
=> setenv fpgadata fff00000
```

7. Save environment variables:

```
=> saveenv
```

8. Reset the board:

```
=> reset
```

By using the 'fpga' command it is possible to clear, reset and booting the FPGA from the command line:

```
=> ? fpga
fpga boot [addr]
fpga reset
fpga clear
```

The 'clear' subcommand asserts the FPGAs PROGRAM# signal for a couple of milliseconds. This will clear the FPGA content. The 'reset' subcommand asserts the FPGA's reset signals (the PMC405 FPGA uses the 'DIN' pin as a low-active reset signal).

The 'boot' subcommand boots the FPGA with the default functionality when no further parameter is passed. The optional 'addr' parameter is used to pass the address of an alternative FPGA .bit file image (optionally compressed by gzip).

10.5.8 painit Command

The 'painit' command does all the setup in order to use the PCIAccess driver with a PMC405V2. 'painit' requires a correctly setup 'pram' variable (see later chapter). 'painit' is typically called by the 'preboot' command:

```
=> setenv preboot painit
=> saveenv
```

The 'painit' command presets the reserved memory (see 'pram' variable) in the following way:

1. The reserved memory is zero'd.
2. The bootloader's representation of the bootloader environment is copied at the beginning of the reserved memory.
3. The environment size (e.g. 0x00000800) is written to the last 32bit memory location (e.g. 0x3ffffc).
4. The reserved memory size (see 'pram') in units of bytes is written before the previous value (e.g. 0x3ffff8).
5. A CRC32 checksum is calculated over the two 32bit value and written at the third last memory location (e.g. 0x3ffff4).
6. A 32bit zero (0x00000000) is written to the fourth last memory location (e.g. 0x3ffff0).

Version 1.x boards need to initialize the reserved memory manually:

```
=> setenv preboot 'eeprom read 3c00000 0 800;mw.l 3fffffc 800;mw.l 3fffff8 400000;crc32
3fffff8 8 3fffff4;mw.l 3fffff0 0 2'
=> saveenv
```

10.5.9 Asserting the PMC RESETOUT# signal

The PMC405 never asserts PMC RESETOUT# automatically because this would lead to race conditions during power-up or reset cycles. But it is possible to assert RESETOUT# using a GPIO pin of the CPU.

The current PMC405 U-Boot implementation does not provide a BSP command to assert the PMC RESETOUT# signal in a comfortable way. In order to assert RESETOUT# manually the following command can be used:

```
=> mw ef600704 20000000
```

This command will make the 405's GPIO2 pin drive a low level by enabling it in the GPIO_TCR register.

Attention: Never add this command to any bootloader variable that is automatically executed after power-on (e.g. preboot or bootcmd). This would lead to a reset loop.

10.6 Special Environment Variables

10.6.1 pcidelay Variable

The booloader variable 'pcidelay' can be used to delay PCI enumeration through the bootloader. When 'pcidelay' is not set, the bootloader on a PMC405 in monarch mode starts PCI enumeration as soon as possible (EREADY is still checked). When 'pcidelay' is set to an amount of time in 1000th of seconds, the bootloader waits for this period, checks EREADY and finally starts PCI enumeration.

Example:

```
=> setenv pcidelay 2000
=> saveenv
```

10.6.2 ptm1la, ptm1ms, ptm2la and ptm2ms Variables

This set of bootloader environment variables can be used to overwrite the default PCI memory resources that are requested by the PMC module. Each PCI address space of the PPC405GPr's PCI bridge is configured by a set of variable ptmXla and pciXms. X stands for the two address spaces 1 and 2.

- The ptmXla determine the 405GPr local base address for the corresponding PCI address translation.
- PtmXms setup the address window size through a size mask (~(size – 1)). The LSB enables the address space.

When the variables are not set these are the defaults:

- ptm1la = 0, ptm1ms= ~(installed_mem_size – 1) | 1
  - This setup enables BAR1 to map the complete SDRAM
- ptm2la = 0xef000000, ptm2ms = 0xff000001
  - This setup enabled BAR2 to map the 405GPr's internal peripherals (e.g. GPIO registers) and the FPGA internal registers.

See the 405GPr usermanual for more details.

10.6.3 pram Variable

The pram variable is used to reserve RAM that is not used by the bootloader. When 'pram' is set, the variable *mem* is automatically set to the amount of available RAM (total RAM – pram). The reserved RAM is used by the esd PCIAccess driver.

Pram must be set to the amount of reserved RAM in KiB.

```
=> setenv pram 4096
=> saveenv
```

11 FPGA

The PMC-CPU/405 comes with a Xilinx Spartan IIE FPGA (part#: XC2S200E-FT256) installed. The FPGA provides several functions that are describes in this chapter.

11.1 Functional Blocks

- IRIG-B
  - decoding of B100 timecode format
  - input source selection
    - frontpanel differential input (option)
    - frontpanel elec. isolated, pseudo differential input (optocoupler) (option)
    - P4 rear differential input (default)
    - P4 rear TTL input
  - local time code generation
- (CAN-) timestamping unit
  - local time (64 bit counter) with 10us resolution
  - timestamp latching for CAN
  - free running counter, sync'ed to IRIG-B signal (digital PLL) or external clock
    - default: external clock from Pn4
  - resetable via register access or IRIG-B control function flag
  - 100kHz CLOCK_OUT + RESET_OUT generation on Pn4 for simple syncronisation of external peripherals
- Pn4 + frontpanel GPIO configuration
  - usage as GPIO
  - alternative functions (IRIG-B, timestamp syncing signals)
- Hardware FIFOs
- Custom module for customized extensions of the PMC-CPU/405

11.2 FPGA Registers

All registers are 32bit wide. Bit31 is MSB, Bit0 is LSB.

<table>
<tr><th>Register</th><th>Offset</th><th>Mode</th><th>Function</th></tr>
<tr><td>CTRL</td><td>0x0000</td><td>R/W</td><td>Mode / Control register
-Pn4 IO configuration (GPIO/alternative functions)
-front panel LED control
-LED behavior
(see description below)</td></tr>
<tr><td>STATUS</td><td>0x0004</td><td>R</td><td>Status register
(see description below)</td></tr>
<tr><td></td><td>...</td><td></td><td>reserved</td></tr>
<tr><td>TSH</td><td>0x0010</td><td>R</td><td rowspan="2">64 bit local timestamp counter
The clock source and frequency can be setup by the TSCTRL register.</td></tr>
<tr><td>TSL</td><td>0x0014</td><td>R</td></tr>
<tr><td>TSCTRL</td><td>0x0018</td><td>R/W</td><td>Timestamp unit control register</td></tr>
<tr><td></td><td>...</td><td></td><td>reserved</td></tr>
<tr><td>TSL0H</td><td>0x0020</td><td>R</td><td rowspan="2">64 bit local timestamp latch for CAN0.
TSH+TSL is latched into these registers when a hardware interrupt from the CAN controller 0 occurs. The register stays unchanged until the next falling edge of the interrupt line.</td></tr>
<tr><td>TSL0L</td><td>0x0024</td><td>R</td></tr>
<tr><td>TSL1H</td><td>0x0028</td><td>R</td><td rowspan="2">64 bit local timestamp latch for CAN1.
TSH+TSL is latched into these register when a hardware interrupt from the CAN controller 1 occurs. The register stays unchanged until the next falling edge of the interrupt line.</td></tr>
<tr><td>TSL1L</td><td>0x002c</td><td>R</td></tr>
<tr><td></td><td>...</td><td></td><td>reserved</td></tr>
<tr><td>IRIG_TIME</td><td>0x0040</td><td>R/W</td><td>IRIG-B time that is used by transmitter. The time can be set by a write to this register.
<table>
<tr><th>Bits</th><th>N</th><th>Function</th></tr>
<tr><td>(MSB) 31...30</td><td>2</td><td>unused</td></tr>
<tr><td>29...20</td><td>10</td><td>Day (2.5 bcd digits: 0..366)</td></tr>
<tr><td>19...14</td><td>6</td><td>Hour (1.5 bcd digits: 0..23)</td></tr>
<tr><td>13...7</td><td>7</td><td>Minute (2.5 bcd digits: 0..59)</td></tr>
<tr><td>6...0 (LSB)</td><td>7</td><td>Second (2.5 bcd digits: 0..59)</td></tr>
</table></td></tr>
<tr><td>IRIG_TOD</td><td>0x0044</td><td>R</td><td>IRIG-B time that is used by transmitter. The SBS field is calculated from the content of the IRIG_TIME register. This register is read-only.
<table>
<tr><th>Bits</th><th>N</th><th>Function</th></tr>
<tr><td>(MSB) 31..17</td><td>15</td><td>unused</td></tr>
<tr><td>16..0</td><td>17</td><td>SBS (straight binary seconds) (0..86399)</td></tr>
</table></td></tr>
<tr><td>IRIG_CF</td><td>0x0048</td><td>R</td><td>IRIG-B control function flags that are inserted into the transmitted frame.</td></tr>
</table>

Register	Offset	Mode	Function
			<table><tr><th>Bits</th><th>N</th><th>Function</th></tr><tr><td>(MSB) 31..27</td><td>5</td><td>unused</td></tr><tr><td>26..0 (LSB)</td><td>27</td><td>CF (control functions) LSB (bit0) corresponds to CF0. CF0 ist the first bit of the control functions filed in the IRIG-B frame.</td></tr></table>
IRIG_CTRL	0x004c	R/W	IRIG-B modul configuration: Bit0: 1=receive framing error
IRIG_RX_TIME	0x0050	R	Received IRIG-B time. Bitfields in this register are identical to IRIG_TIME register.
IRIG_RX_TOD	0x0054	R	Received IRIG-B time. Bitfields in this register are identical to IRIG_TOD register.
IRIG_RX_CF	0x0058	R	Received IRIG-B time. Bitfields in this register are identical to IRIG_CF register.
	...		reserved
HOSTCTRL	0x0060	R/W	Host control register. This register can be used to request an interrupt service by the PPC405. This is an alternative way than a write access to the PCI_COMMAND register for an other PCI device (e.g. host CPU) to trigger an interrupt. The implementation of PCI configuration cycles, as needed for access to the PCI_COMMAND register, is very slow and time consuming on some operating systems (e.g. MS Windows with RTX extension). The HOSTCTRL register also provides interrupt masking capabilities for the FIFO module and the optional FPGA custom module.
	...		reserved
DDFSCTRL	0x0070	R/W	DDFS control register / ADPLL phase error (see detailed description below)
DDFSINC	0x0074	R	DDFS increment value. (see detailed description below)
	...		reserved
FIFO0_DATA	0x0080	R/W	FIFO 0 data port
FIFO0_CTRL	0x0084	R/W	FIFO 0 control/status register
FIFO1_DATA	0x0088	R/W	FIFO 1 data port
FIFO1_CTRL	0x008c	R/W	FIFO 1 control/status register
FIFO2_DATA	0x0090	R/W	FIFO 2 data port
FIFO2_CTRL	0x0094	R/W	FIFO 2 control/status register
FIFO3_DATA	0x0098	R/W	FIFO 3 data port
FIFO3_CTRL	0x009c	R/W	FIFO 3 control/status register

11.3 Register Description

11.3.1 CTRL Register (0x0000)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET_EN	CLOCK_EN	CLKRST_EN_CSTM	IRIGB_R_EN	IRIGB_R_EN_CSTM	IRIGB_F_EN	IRIGB_F_EN_CSTM	CLOCK_MODE[2..0]			RESET_MODE[2..0]			IRIGB_SRC[2..0]			IRIGB_R_OUT_MODE[1..0]		IRIGB_R_OUTPM_MODE[1..0]		IRIGB_F_OUTPM_MODE[2..0]			HOST_IE	Reserved				RUNLED_MODE[1..0]		RST_C1N	RST_C0N

Bits	Name	Function
31	RESET_EN	When set to '1' the RS485 RESET signal driver is enabled. The RS485 driver circuit is located on the PMC405-PIM module. When enabled the PMC405 is the source of the RESET signal. When the PMC405 is not used with the PMC405-PIM this flag can be used to control the state of the TTL signal RESET_R_EN on Pn4 (pin#62). Default after reset: '0'
30	CLOCK_EN	When set to '1' the RS485 CLOCK signal driver is enabled. The RS485 driver circuit is located on the PMC405-PIM module. When enabled the PMC405 is the source of the CLOCK signal. When the PMC405 is not used with the PMC405-PIM this flag can be used to control the state of the TTL signal CLOCK_R_EN on Pn4 (pin#60). Default after reset: '0'
29	CLKRST_EN_CSTM	When set to '1' the RESET_EN and CLOCK_EN signals on Pn4 (see above) are connected to the FPGA's custom module. Default after reset: '0'
28	IRIGB_R_EN	When set to '1' the RS485 output driver for IRIG-B transmission on Pn4 on the PMC405 is enabled and IRIG_B_R_OUT_P/_M become a differential output. Default after reset: '0'
27	IRIGB_R_EN_CSTM	When set to '1' the IRIGB_R_EN signal (driver enable for RS485 transceiver) is connected to the FPGA's custom module. Default after reset: '0'

Bits	Name	Function
26	IRIGB_F_EN	When set to '1' the RS485 output driver for IRIG-B transmission on the frontpanel D-Sub9 on the PMC405 is enabled and IRIG_B_F_OUT_P/_M become a differential output. (Frontpanel interface is an option) Default after reset: '0'
25	IRIGB_F_EN_CSTM	When set to '1' the IRIGB_F_EN signal (driver enable for RS485 transceiver) is connected to the FPGA's custom module. Default after reset: '0'
24..22	CLOCK_MODE[2..0]	'000': CLOCK_OUT signal is driven low '001': CLOCK_OUT signal is driven high '010': 100kHz, 50% duty cycle clock with IRIG-B sync option '111': CLOCK_OUT is connected to the FPGA's custom module. Default after reset: '000'
21..19	RESET_MODE[2..0]	'000': RESET_OUT signal is driven low '001': RESET_OUT signal is driven high '010': RESET_OUT pulse is triggered by manual timestampcounter reset (see timestamp unit) '011': RESET_OUT pulse is triggered by IRIG-B receiver unit (CF). '100': esd internal test mode – do not use '111': RESET_OUT is connected to the FPGA's custom module. Default after reset: '000' The automatically timed reset pulse has about 6 us pulse width.
18..16	IRIGB_SRC[2..0]	IRIG-B input select for IRIG-B module: '000': IRIG_B_R_P/M (Pn4 diff. input) '001': IRIG_B_R_IN (Pn4 TTL input) '010': IRIG_B_F_P/M (frontpanel RS485 input) (option) '011': IRIG_B_F_IN_P/M (frontpanel opto isolated input) (option) '1xx': reserved for future use Default after reset: '000'
15..14	IRIGB_R_OUT_MODE[1..0]	configuration of IRIG-B TTL output on Pn4 '00': signal is driven low '01': signal is driven high '10': IRIG-B output '11': signal is connected to the FPGA's custom module. Default after reset: '00'

Bits	Name	Function
13..12	IRIGB_R_OUTPM_MODE[1..0]	configuration of IRIG-B differential output on Pn4 '00': signal is driven low '01': signal is driven high '10': IRIG-B output '11': signal is connected to the FPGA's custom module. Default after reset:1 '00'
11..9	IRIGB_F_OUTPM_MODE[2..0]	configuration of IRIG-B differential output on frontpanel: '000': signal is driven low '001': signal is driven high '010': IRIG-B output '011': IRIG-B input passthrough (this can be used to pass an IRIG-B signal from Pn4 to the frontpanel D-Sub connector) '111': signal is connected to the FPGA's custom module. Default after reset: '000'
8	HOST_IE	Host control interrupt enable. See HOSTCTRL register.
7..4	-	reserved
3..2	RUNLED_MODE[1..0]	'00': indicate „CPU running state“ (DRAM access) '01': IRIG-B sync state (seconds clock=sync'd, fast blinking=error) '10': LED off '11': LED on
1	RST_C1N	Reset# line of 2nd SJA1000 CAN controller Default after (FPGA-) reset: '0'. Bit is set by bootloader.
0	RST_C0N	Reset# line of 1st SJA1000 CAN controller Default after (FPGA-) reset: '0'. Bit is set by bootloader.

11.3.2 STATUS Register (0x0004)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VERSION								Reserved													CSTM_ISF	FIFO_ISF	HOST_ISF	Reserved		IRIGB_R_IN	IRIGB_R_IN_PM	IRIGB_F_IN	IRIGB_F_IN_PM	CLOCK_IN	RESET_IN

The logic state readback functionality is independant from any alternative function.

Bits	Name	Function
(MSB) 31..24	VERSION	FPGA version (current version is 0x05)
23 .. 11	reserved	unused (read '0')
10	CSTM_ISF	Custom module interrupt status flag. This bit indicates a pending interrupt from the FPGA's custom module.

Bits	Name	Function
9	FIFO_ISF	FIFO module interrupt status flag. This bit indicates a pending interrupt from the FIFO module.
8	HOST_ISF	Host interrupt status flag. This bit indicates a pending host interrupt.
7 .. 6	reserved	unused (read '0')
5	IRIGB_R_IN	logic state of Pn4 TTL input signal
4	IRIGB_R_IN_PM	logic state of Pn4 RS485 input signal
3	IRIGB_F_IN	logic state of frontpanel opto input
2	IRIGB_F_IN_PM	logic state of frontpanel RS485 input
1	CLOCK_IN	logic state of Pn4 CLOCK input signal
0	RESET_IN	logic state of Pn4 RESET input signal

11.3.3 TSCTRL - Timestamp unit control register (0x0018)

Bits	Name	Function
(MSB) 31..5	reserved	unused -reads always 0
5	TS_CFRSTEN	Enable timestamp counter reset via IRIG-B control flags
4	TS_MRST	Timestamp manual reset -write '1' to reset timestamp counter -reads always '0'
3	TS_XRSTEN	Timestamp reset enable: '1': enable reset of timestamp counter via external RESET signal on Pn4
2..0	TS_SRC[2..0]	Timestampcounter clock source select: '000': FPGA internal 100kHz clock '010': external CLOCK signal on Pn4 (other combinations of the TS_SRC bits are reserved for future use.)

Note: Write 0x0000.0000 to TSCTRL for testing and read the current 64 bit counter value from offset 0x0010 and 0x0014. This uses the FPGA internal 100kHz timestamp clock source.

11.3.4 HOSTCTRL – Host control register (0x0060)

Bits	Name	Function
31 .. 6	RESERVED	Always read '0'
5	CSTMIE_GATE	see CSTMIE_FLAG
4	CSTMIE_FLAG	'1': unmask interrupts from the FPGA custom module. '0': mask interrupts from the FIFO module. The CSTMIE_FLAG can only be modified when the CSTMIE_GATE bit is set during the same write access to the HOSTCTRL register. This makes modification of the CSTMIE_FLAG possible without read-modify-write operations.
3	FIFOIE_GATE	see FIFOIE_FLAG

Bits	Name	Function
2	FIFOIE_FLAG	'1': unmask interrupts from the FIFO module. FIFO interrupts must also be enabled for each FIFO in the corresponding FIFOx_CTRL register. '0': mask interrupts from the FIFO module. The FIFOIE_FLAG can only be modified when the FIFOIE_GATE bit is set during the same write access to the HOSTCTRL register. This makes modification of the FIFOIE_FLAG possible without read-modify-write operations.
1	HCINT_GATE	see HCINT_FLAG
0	HCINT_FLAG	'1': assert host interrupt (host interrupts must be enabled via HOST_IE bit in the CTRL register) '0': deassert host interrupt The HCINT_FLAG can only be modified when the HCINT_GATE bit is set during the same write access to the HOSTCTRL register. This makes modification of the HCINT_FLAG possible without read-modify-write operations.

11.3.5 DDFSCTRL/DDFSINC – Clock generator registers (0x0070, 0x0074)

The FPGA uses a DDFS (direct frequency synthesis clock generator to synthesize a 100kHz reference clock. This clock can be output on Pn4 and/or directly passed to the timestamp unit. The reference clock can optionally be synchronized through an IRIG-B timecode signal and the frequency is controlled by an FPGA internal ADPLL.

The current reference clock frequency is calculated as

$f_{100kHz} = 33.33\ MHz * DDFSINC / 2^{32}$.

Base value with IRIG-B sync regulator disabled is 0x00c4a100. When the IRIG-B sync regulator is active the DDFSINC register is updated every IRIG-B bit time.

The DDFS unit is can be controlled through the DDFSCTRL register:

Bits	Name	Function
31 .. 3	RESERVED	Always read '0'
2	RESTART	'1' : reference clock is stopped an regulator parameter are reset. '0': reference clock is enabled. When REGULATE flag is set, the oscillator starts with the next IRIG-B bit. Default after (FPGA-) reset is: '0'
1	REGULATE	'1': reference clock is synchronized to IRIG-B input source. '0': reference clock is free running Default after (FPGA-) reset is: '0'
0	STOP	'1': reference clock is stopped and regulator parameter a left untouched. '0': reference clock is enabled. When REGULATE flag is set, the oscillator starts with the next IRIG-B bit.

Bits	Name	Function
		Default after (FPGA-) reset is: '0'

11.3.6 FIFO<0...3>_DATA

32 bit wide FIFO data port. Data producers write to this register. Consumers read from this register. Only 32 bit access to this register is supported.

11.3.7 FIFO<0...3>_CTRL

FIFO control register. Each register controls the behavior of a single FIFO.

Bits	Name	Access	Function
15	IE	R/W	Enable interrupts from this FIFO. An interrupt is asserted when the FIFO contains at least one entry (EMPTY=0). The interrupt is reset when all data is read from the FIFO. Clearing the IE bit only masks the interrupt.
11 ... 14	RESERVED	R	Always read '0'
10	OVERFLOW	R/W	This flag is set when a write access to a full FIFO occured, which means that data got lost. Writing a '0' resets the OVERFLOW flag.
9	EMPTY	R	FIFO empty. The FIFO does not contain any valid data.
8	FULL	R	FIFO full. The FIFO contains exactly 256 valid entries that are ready to be read.
7...0	LEVEL	R	Filling level of FIFO (0=empty). Each write access to the data port increases LEVEL while a read access decreases LEVEL. LEVEL=0 means the FIFO is empty (EMPTY=1). A full fifo is indicated by LEVEL=0x00 and the fifo full flag set to '1'. This means bits 8..0 can be interpreted as the total fifo level from 0x000 to 0x100.

11.4 Using the FIFO module

The PMC405 provides four hardware FIFOs that are implemented in the on-board FPGA. Each FIFO provides a 32 bit data port and has a capacity of 256 entries. The FIFOs share a common external interrupt to the PPC405 CPU. Each FIFO is represented by a set of two 32 bit registers. The registers are accessible from the PPC405 and from the PCI bus. This allows other PCI devices to write data into the FIFOs data ports. The following sample code demonstrates the FIFO module usage. The extract is taken from the PMC405V2 U-Boot sourcecode:

```
/* FPGA interface */
#define FPGA_OUT32(p,v) out32((int)(p), (v))
#define FPGA_IN32(p) in32((int)(p))
#define FPGA_SETBITS(p,v) out32((int)(p), in32((int)(p)) | (v))
#define FPGA_CLRBITS(p,v) out32((int)(p), in32((int)(p)) & ~(v))

struct pmc405v2_fifo_s {
      u32 data;
      u32 ctrl;
```

```
};

/* bits in fifo ctrl register */
#define FIFO_IE              (1 << 15)
#define FIFO_OVERFLOW        (1 << 10)
#define FIFO_EMPTY           (1 <<  9)
#define FIFO_FULL            (1 <<  8)
#define FIFO_LEVEL_MASK 0x000000ff

#define FIFO_COUNT      4

struct pmc405v2_fpga_s {
      u32 ctrl;
      u32 status;
      u32 pad1[0x40 / sizeof(u32) - 2];
      u32 irig_time;                  /* offset: 0x0040 */
      u32 irig_tod;
      u32 irig_cf;
      u32 pad2;
      u32 irig_rx_time;               /* offset: 0x0050 */
      u32 pad3[3];
      u32 hostctrl;                   /* offset: 0x0060 */
      u32 pad4[0x20 / sizeof(u32) - 1];
      struct pmc405v2_fifo_s fifo[FIFO_COUNT]; /* 0x0080..0x009f */
};

typedef struct pmc405v2_fpga_s pmc405v2_fpga_t;

/* status register */
#define STATUS_FIFO_ISF      (1 <<  9)

/* hostctrl register */
#define HOSTCTRL_FIFOIE_GATE (1 <<  3)
#define HOSTCTRL_FIFOIE_FLAG (1 <<  2)

/* FPGA to PPC interrupt */
#define FPGA_IRQ 31

int got_fifoirq;

/* FIFO module interrupt handler */
int fpga_interrupt(pmc405v2_fpga_t *fpga)
{
      int rc = -1; /* not for us */
      u32 status = FPGA_IN32(&fpga->status);

      /* check for interrupt from fifo module */
      if (status & STATUS_FIFO_ISF) {
            /* disable/mask this int source */
            FPGA_OUT32(&fpga->hostctrl, HOSTCTRL_FIFOIE_GATE);
            rc = 0;
            got_fifoirq = 1; /* trigger backend */
      }
      return rc;
}

/* pretty print content of fifo 'f' */
void dump_fifo(pmc405v2_fpga_t *fpga, int f, int *n)
{
      u32 ctrl;

      while(!((ctrl = FPGA_IN32(&fpga->fifo[f].ctrl)) & FIFO_EMPTY)) {
            printf("%5d  %d    %3d  %08x",
                  (*n)++, f, ctrl & (FIFO_LEVEL_MASK | FIFO_FULL),
                  FPGA_IN32(&fpga->fifo[f].data));
            if (ctrl & FIFO_OVERFLOW) {
                  printf(" OVERFLOW\n");
                  FPGA_CLRBITS(&fpga->fifo[f].ctrl, FIFO_OVERFLOW);
```

```
            } else {
                  printf("\n");
            }
      }
}

/* 'fifo' command from PMC405V2 U-Boot sourcecode */
int do_fifo(cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
      pmc405v2_fpga_t *fpga = (pmc405v2_fpga_t *)FPGA_BA;
      int i;
      int n = 0;
      u32 ctrl, data, f;
      char str[] = "\\|/-";
      int abort = 0;
      int count = 0;
      int count2 = 0;

      switch(argc) {
      case 1:
            /* print all fifos status information */
            printf("fifo level status\n");
            printf("_____________________________\n");
            for (i=0; i<FIFO_COUNT; i++) {
                  ctrl = FPGA_IN32(&fpga->fifo[i].ctrl);
                  printf(" %d    %3d  %s%s%s %s\n",
                        i, ctrl & (FIFO_LEVEL_MASK | FIFO_FULL),
                        ctrl & FIFO_FULL ? "FULL     " : "",
                        ctrl & FIFO_EMPTY ? "EMPTY    " : "",
                        ctrl & (FIFO_FULL|FIFO_EMPTY) ? "" : "NOT EMPTY",
                        ctrl & FIFO_OVERFLOW ? "OVERFLOW" : "");
            }
            break;

      case 2:
            /* completely read out fifo 'n' */
            if (!strcmp(argv[1],"read")) {
                  printf("  #   fifo level data\n");
                  printf("_____________________________\n");

                  for (i=0; i<FIFO_COUNT; i++) {
                        dump_fifo(fpga, i, &n);
                  }
            } else if (!strcmp(argv[1],"wait")) {
                  got_fifoirq = 0;

                  irq_install_handler (FPGA_IRQ,
                                       (interrupt_handler_t *)fpga_interrupt, fpga);

                  printf("  #   fifo level data\n");
                  printf("_____________________________\n");

                  /* enable all fifo interrupts */
                  FPGA_OUT32(&fpga->hostctrl,
                             HOSTCTRL_FIFOIE_GATE | HOSTCTRL_FIFOIE_FLAG);
                  for (i=0; i<FIFO_COUNT; i++) {
                        /* enable interrupts from all fifos */
                        FPGA_SETBITS(&fpga->fifo[i].ctrl, FIFO_IE);
                  }

                  while (1) {
                        /* wait loop */
                        while(!got_fifoirq) {
                              count++;
                              if (!(count % 100)) {
                                    count2++;
                                    putc(0x08); /* backspace */
                                    putc(str[count2 % 4]);
                              }
```

```
                                /* Abort if ctrl-c was pressed */
                                if ((abort = ctrlc())) {
                                        puts("\nAbort\n");
                                        break;
                                }
                                udelay(1000);
                        }
                        if (abort) {
                                break;
                        }

                        /* simple fifo backend */
                        if (got_fifoirq) {
                                for (i=0; i<FIFO_COUNT; i++) {
                                        dump_fifo(fpga, i, &n);
                                }
                                got_fifoirq = 0;
                                /* unmask global fifo irq */
                                FPGA_OUT32(&fpga->hostctrl,
                                        HOSTCTRL_FIFOIE_GATE | HOSTCTRL_FIFOIE_FLAG);
                        }
                }

                /* disable all fifo interrupts */
                FPGA_OUT32(&fpga->hostctrl, HOSTCTRL_FIFOIE_GATE);
                for (i=0; i<FIFO_COUNT; i++) {
                        FPGA_CLRBITS(&fpga->fifo[i].ctrl, FIFO_IE);
                }
                irq_free_handler(FPGA_IRQ);

        } else {
                printf ("Usage:\nfifo %s\n", cmdtp->help);
                return 1;
        }
        break;

case 4:
case 5:
        if (!strcmp(argv[1],"write")) {
                /* get fifo number or fifo address */
                f = simple_strtoul(argv[2], NULL, 16);

                /* data paramter */
                data = simple_strtoul(argv[3], NULL, 16);

                /* get optional count parameter */
                n = 1;
                if (argc >= 5) {
                        n = (int)simple_strtoul(argv[4], NULL, 10);
                }

                if (f < FIFO_COUNT) {
                        printf("writing %d x %08x to fifo %d\n",
                                n, data, f);
                        for (i=0; i<n; i++) {
                                FPGA_OUT32(&fpga->fifo[f].data, data);
                        }
                } else {
                        printf("writing %d x %08x to fifo port at address %08x\n",
                                n, data, f);
                        for (i=0; i<n; i++) {
                                out32(f, data);
                        }
                }
        } else {
                printf ("Usage:\nfifo %s\n", cmdtp->help);
                return 1;
        }
```

```
            break;

      default:
            printf ("Usage:\nfifo %s\n", cmdtp->help);
            return 1;
      }
      return 0;
}
U_BOOT_CMD(
      fifo,  5,     1,     do_fifo,
      "fifo    - Fifo module operations\n",
      "wait\nfifo read\n"
      "fifo write fifo(0..3) data [cnt=1]\n"
      "fifo write address(>=4) data [cnt=1]\n"
      "  - without arguments: print all fifo's status\n"
      "  - with 'wait' argument: interrupt driven read from all fifos\n"
      "  - with 'read' argument: read current contents from all fifos\n"
      "  - with 'write' argument: write 'data' 'cnt' times to 'fifo' or 'address'\n"
      );
```

11.5 FPGA Custom Module

The FPGA custom module allows users to add customized functionality to the PMC405 onboard FPGA. For extending the PMC405 FPGA esd provides the following files (shipped in a zip archive) that have to be added to an Xilinx ISE design project:

- pmc405v2_cstm_fpga_r6/pmc405v2.vhd
  (top level VHDL source for PMC405 FPGA design)
- pmc405v2_cstm_fpga_r6/pmc405v2_custom_module.vhd
  (example VHDL sourcecode to custom extension)
- pmc405v2_cstm_fpga_r6/pmc4053.ucf
  (constraints file)
- pmc405v2_cstm_fpga_r6/pmc405v2_common.ngc
  (esd IP of fixed design parts as described in previous chapter of this document)
- pmc405v2_cstm_fpga-r5/README_CustomDesign.txt
  (information text file)

After implementing the FPGA design the genererated .bit file can directly be loaded into the PMC405 FPGA by using the bootloader's 'fpga' command (see above).

Note: 'r6' in the above filenames stands for esd's design revision.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity custom_module_pmc405v2 is
      port (
             -- bus interrupt to CPU / local bus
             CLK : in std_logic;                          -- 33.33 MHz
             RESET_N : in std_logic;                      -- async active low reset
             D_OUT : out std_logic_vector(31 downto 0);   -- data bus output
             D_IN : in std_logic_vector(31 downto 0);     -- data bus input
             ADDR : in std_logic_vector(15 downto 0);     -- address bus
                           -- addr(1 downto 0) are always low
                           -- because only 32bit access is supported.
                           -- address range 0x0000-0x7fff is reserved
                           -- for esd modules

             nOE : in std_logic;                           -- low active output enable / read sign
             READY : out std_logic;                        -- can be used to insert wait states
                                                           -- keep '1' when not needed
             nCS : in std_logic;                           -- low active chip select
                                                           -- active from ADDR = 0x8000...0xffff
                           -- nCS is active for at least three cycles
                           -- or longer when READY is held low.

             nIRQ : out std_logic;                         -- active low level sensitive interrupt
                                                           -- interrupt must globally enabled in
                                                           -- CTRL-register


      -- IOs available on PMC-CPU405/440 (take care of signal direction)
             -- inputs are always usable
             -- outputs must be connected to FPGA pins through top level module
             -- via CTRL-register
      RESET_R_IN : in std_logic;                   -- general purpose input on PMC-Pn4
      RESET_R_OUT : out std_logic;                 -- general purpose output on PMC-Pn4
      RESET_R_EN : out std_logic;                  -- general purpose output on PMC-Pn4
      CLOCK_R_IN : in std_logic;                   -- general purpose input on PMC-Pn4
      CLOCK_R_OUT : out std_logic;                 -- general purpose output on PMC-Pn4
      CLOCK_R_EN : out std_logic;                  -- general purpose output on PMC-Pn4
```

```
      IRIG_B_R_IN : in std_logic;                  -- general purpose input on PMC-Pn4
      IRIG_B_R_OUT : out std_logic;                -- general purpose output on PMC-Pn4

      IRIG_B_R_IN_T : in std_logic;                -- | These three signals control a
      IRIG_B_R_OUT_T : out std_logic;              -- | MAX3485 RS485 diff. transceiver.
      IRIG_B_R_OUT_EN : out std_logic;              -- | The signal is available on Pn4.

              -- PMC405 only
      IRIG_B_F_IN : in std_logic;                  -- | These three signals control a
      IRIG_B_F_OUT : out std_logic;                -- | MAX3485 RS485 diff. transceiver.
      IRIG_B_F_OUT_EN : out std_logic;             -- | The signal is available on the
                                                   -- | D-Sub9 frontpanel connector


      IRIG_B_F_IN_G : in std_logic                 -- opto-isolated pseudo diff. input
                                                   -- on D-Sub9 frontpanel connector
      );
end custom_module_pmc405v2;

architecture Behavioral of custom_module_pmc405v2 is

constant ADDR_REG_CSTM_COUNT : std_logic_vector(15 downto 0) := x"8000";
constant ADDR_REG_CSTM_TEST1 : std_logic_vector(15 downto 0) := x"8010";
constant ADDR_REG_CSTM_TEST2 : std_logic_vector(15 downto 0) := x"8020";

signal counter : std_logic_vector(31 downto 0);
signal reg_test1 : std_logic_vector(31 downto 0);
signal reg_test2 : std_logic_vector(31 downto 0);

begin
      -- assign defaults to unused bus signals
      -- to avoid disturbing other modules
      READY <= '1';
      nIRQ <= '1';

      -- implement a loadable 32bit counter and some registers
      process(clk, reset_n)
      begin
              if reset_n='0' then
                      counter <= (others=>'0');
                      reg_test1 <= (others=>'0');
                      reg_test2 <= (others=>'0');
              elsif rising_edge(clk) then
                      -- counter is loadable via write access to address 0x8000
                      if nCS='0' and nOE='1' and addr=ADDR_REG_CSTM_COUNT then
                              counter <= d_in;
                      else
                              counter <= counter + 1;
                      end if;

                      if nCS='0' and nOE='1' then
                              case addr is
                                      when ADDR_REG_CSTM_TEST1 =>
                                              reg_test1 <= d_in;
                                      when ADDR_REG_CSTM_TEST2 =>
                                              reg_test2 <= d_in;
                                      when others =>
                                              null;
                              end case;
                      end if;
              end if;
      end process;

      -- read counter via read access to address 0x8000
      D_OUT <= counter   when nCS='0' and nOE='0' and addr=ADDR_REG_CSTM_COUNT
              else (others=>'Z');
      D_OUT <= reg_test1 when nCS='0' and nOE='0' and addr=ADDR_REG_CSTM_TEST1
              else (others=>'Z');
      D_OUT <= reg_test2 when nCS='0' and nOE='0' and addr=ADDR_REG_CSTM_TEST2
              else (others=>'Z');

      -- 5 x general purpose output
   RESET_R_OUT <= '0';
   RESET_R_EN <= '0';
   CLOCK_R_OUT <= '0';
   CLOCK_R_EN <= '0';
   IRIG_B_R_OUT_T <= '0';
```

```
      -- 2 x bidirectional RS485 signals
   IRIG_B_R_OUT <= '0';
   IRIG_B_R_OUT_EN <= '0';

   IRIG_B_F_OUT <= '0';
   IRIG_B_F_OUT_EN <= '0';

   -- inputs are ignored in this sample code
      --  RESET_R_IN, CLOCK_R_IN, IRIG_B_R_IN_T (TTL inputs)
      --  IRIG_B_F_IN, IRIG_B_F_IN_G, IRIG_B_R_IN (diff. inputs)
end Behavioral;
```

12 Ordering information

Type	Description	Order-no.
PMC-CPU/405	PowerPC PrPMC module	V.2020.02
PMC-CPU/405-ME	Usermanual in english	V.2020.21
PIM-CPU/405 2xCAN	PIM module with 2 CAN physical layers	V.2025.02
PIM-CPU/405-PBOX 2xCAN	PIM module with 2 CAN physical layers and PMC-Box clock/reset distribution logic	V.2025.04
PMC-CPU/405-VxW	VxWorks BSP	V.2020.30
PMC-CPU/405-Linux	Linux BSP	V.2020.32