FASTSERIES

FASTCAMERA CCD USER AND REFERENCE MANUAL

FVM-00406

COPYRIGHT NOTICE

Copyright © 2010 by FastVision LLC.

Document Name: FastCamera CCD User and Reference Manual

Document Number: FVM-00406

Revision History: 1.0 August 2010

Trademarks:

FastVision LLC.
131 Daniel Webster Highway, #529
Nashua, NH 03060
USA

Telephone: 603-891-4317
Fax: 603-891-1881

Web Site:
http://www.Fast-Vision.com/

Email:
sales@Fast-Vision.com, or support@Fast-Vision.com

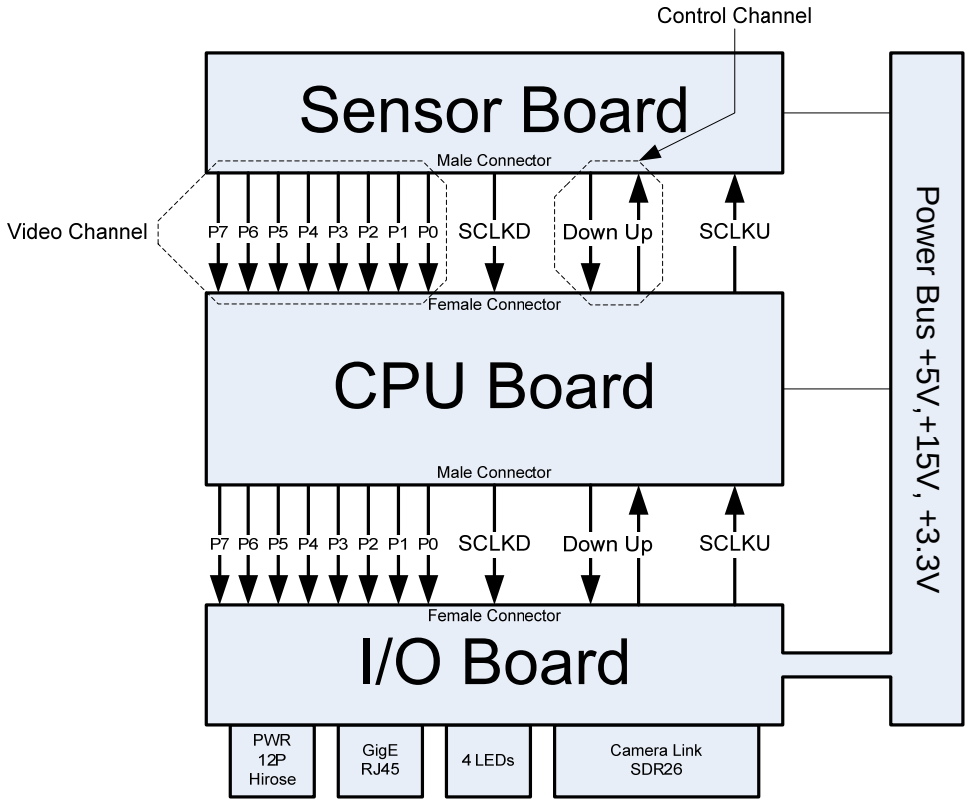
TABLE OF CONTENTS

PURPOSE

This manual describes the architecture and use of the 'Kodak Fast Camera' camera family.

CAMERA BOARDS BLOCK DIAGRAMS

This document applies to the FC105, FC205, FC215, and FC405.



SENSORS

FCxxx camera is designed to be compatible with the 1, 2, 4, and 8 Mpixel Kodak Sensors.

BOARD INTERCONNECTIONS

Power Bus

Running up to stack of boards is the power bus. The power bus is driven by the I/O board at the bottom of the stack. The power bus uses an eleven pin SIP connector with the male pins being soldered into the I/O board, while the other boards have pass through female connectors allowing the male pins to extend up through the whole stack of boards. This connection method limits the number of connector pin contacts between the power supply and the boards to one contact. The 11 pins in the power bus are:

Pin	Function
1	+15 Volts
2	+/-15 Volt Return
3	-15 Volts

4	**Ground**
5	**Ground**
6	**+5 Volts**
7	**+5 Volts**
8	**Ground**
9	**Ground**
10	**+3.3 Volts**
11	**+3.3 Volts**

The I/O board has a power converter which provides the +3.3 volts to the bus from the +5 volt from the power brick. +15 Volts currently comes from the external power brick . .

Serial Channel

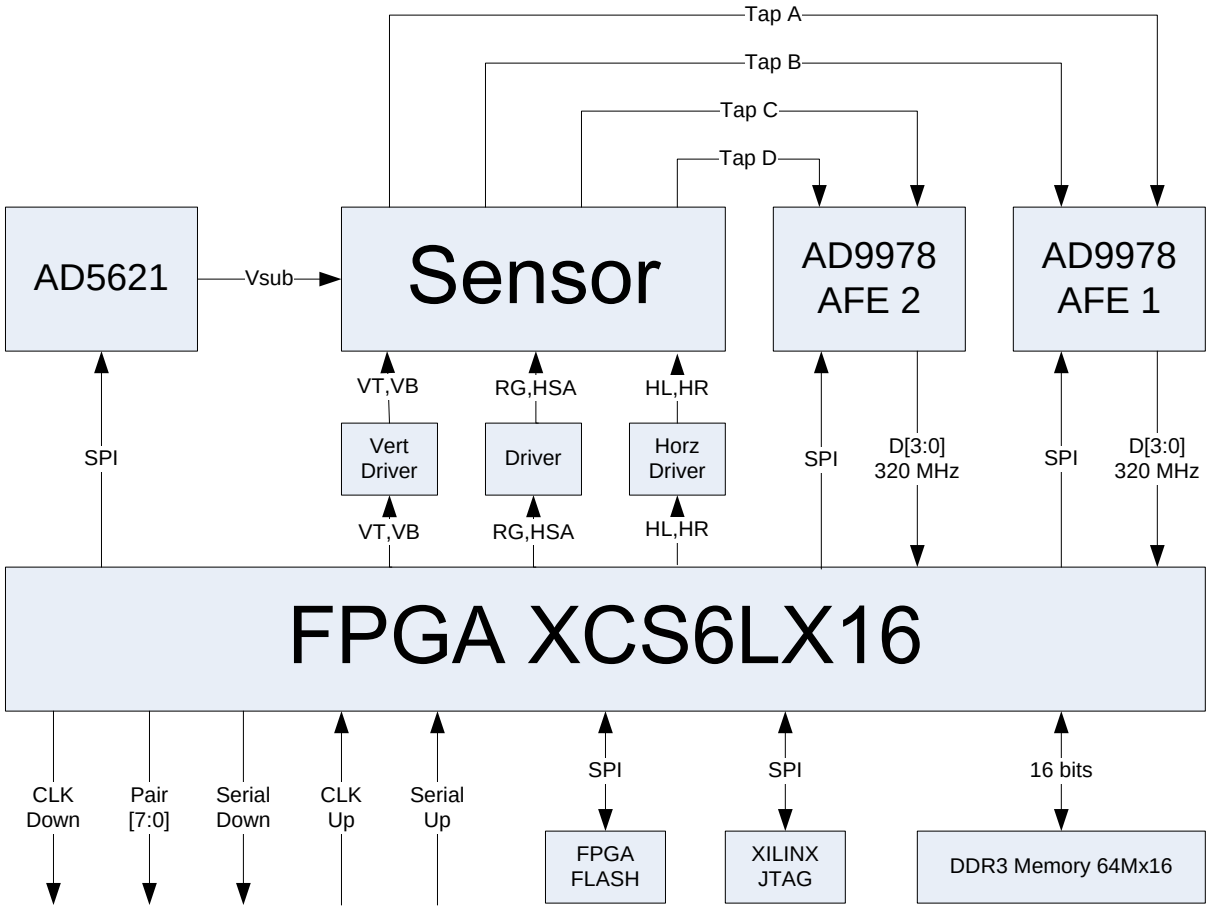
There is a bidirectional serial channel which passes through each board in the stack. The sensor board and the I/O board have receivers and transmitters while the inner CPU boards contain a receiver and transmitter for each direction. The serial channel operates at 320 Mb/sec and is re-timed at each board. The serial channel is implemented with a source synchronous clock, so the transmitters provide the clock to the receivers on the next board, above or below. A total of four signals are used, clock and data, for each direction. The serial cannel implements a 16 bit synchronous protocol using the value 0xA55A as an idle SYNC word. The serial channel sends ‘Cells’ which are from 2 to 17 words (16 bits) long. See the section below for the details of this protocol.

Video Channel

The video channel is a sixteen bit parallel data channel. This channel is implemented as 16 single ended data bits in 8 pairs (P0 to P7 in the figure) with a clock frequency of 160 MHz. The video channel is unidirectional and is sourced at the sensor board. Each board in the stack receives and retimes the video and retransmits it to the next board in the stack. The I/O board at the bottom of the stack receives the video and converts it to the output type implemented for the camera (GigE, Camera Link, or Analog). The video channel operates at 160 MHz single clock per data. This provides bandwidth for the four 16 bit pixel channels (taps) at 40 MHz each.

SENSOR BOARD

Block Diagram



The sensor board contains the sensor drive electronics and is the source of the video channel. The sensor board performs the following steps:

1. The FPGA decides based on the trigger mode and exposure how to operate the sensor.
2. The FPGA generates the driving clocks which operate the sensor in the selected mode for example four tap mode.
3. The sensor detects the light hitting its pixels and provides an analog signal to the Analog front end chips (AFE1 and AFE2).
4. The AFE chips convert the analog signal from the sensor to 14 bit digital values which are representative of the light hitting the sensor.
5. The FPGA re-orders the taps coming from the sensor as it writes the pixels into a frame buffer, to create a raster ordered array of pixels in the buffer memory.
6. When the frame is complete, the FPGA swaps buffers and begins reading out the pixels in raster order from the frame buffer just completed.
7. The FPGA does dark and bright field correction on the pixels using a table stored in the buffer memory.
8. The corrected pixels are sent in ‘Cells’ (small packets) down the video channel to the next board in the stack.

The sensor board is essentially ‘the camera’ part of the FC205 camera. If the camera is built with a CPU board, it is a ‘smart’ camera. The I/O board provides protocol conversion from the parallel format of the video channel, to GigE, Camera Link, and Analog.

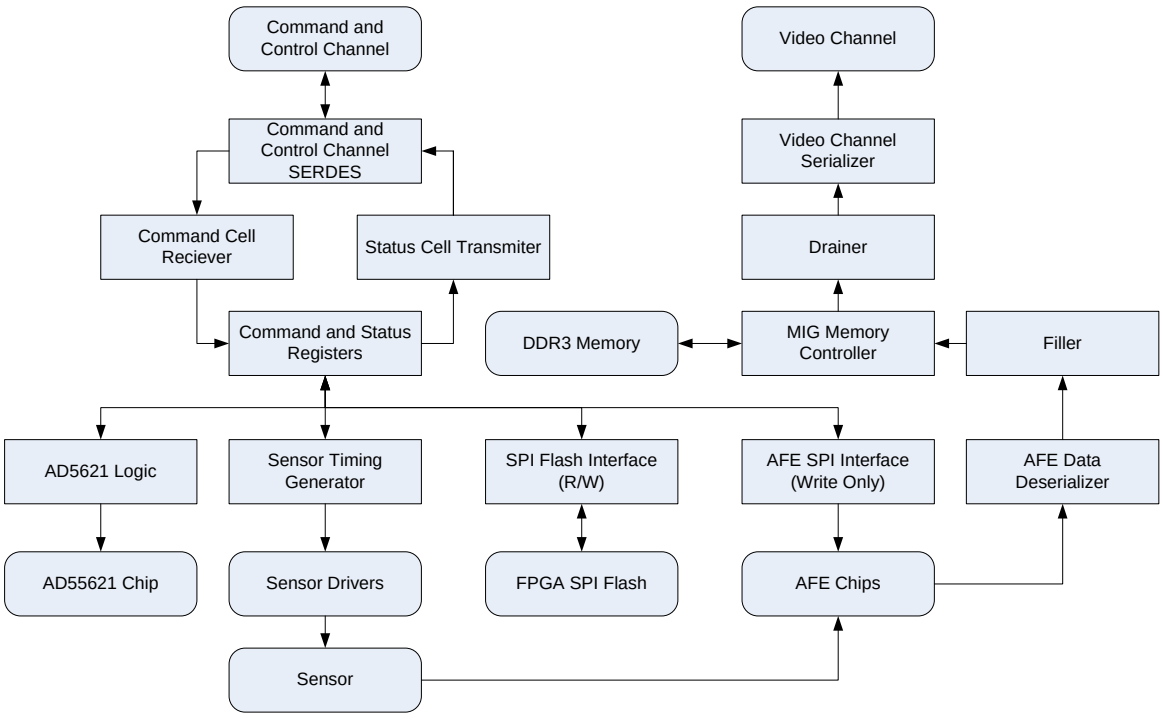
The flash on the FPGA is 128 Mbits. It contains the bit stream for booting the FPGA, a script for initializing the sensor board, and an array of correction coefficients for dark and bright field correction of the sensor image. The sensor can be operated in three modes, one tap, two taps and four tap modes. The taps read out from the corners of the sensor.

In one tap mode the sensor is read out in raster order from bottom to top (correcting for the Lens inversion). In two tap mode the sensor can be split into two pieces. One split is into a left and right half of the sensor; the other split is top and bottom half of the sensor. In the left and right split, the sensor is read out from the bottom on the left and the right corners. The right half is read out reversed left to right, which the FPGA corrects by writing the values to the memory buffer in reverse address order. In the top and bottom readout mode, the top part is read out last line first, so the FPGA writes the lines in reverse order to the buffer, building the image in the buffer memory in raster order.

Finally in four tap mode the sensor is read out from the four corners of the image array. The bottom left reads out in order, the bottom right reads the lines reversed end to end, the top left reads out with the lines in reverse order top to bottom, and the top right tap reads out the lines reversed end to end, and the last line first.

The FPGA corrects the reversals while writing the pixels to the memory buffer. When the buffer swap occurs, the FPGA reads out the in order image and applies the correction factors. The 14 bit pixels have 14 bit values subtracted from them to do the dark field correction. If the resulting pixel value would be negative it is set to zero. Next the dark field corrected pixels are multiplied by a 14 bit gain coefficient, producing a 28 bit result. The 28 bits are shifted right by a programmable amount. If the upper 12 bits of the shifted result are non-zero then the pixel value is set to all ones. The lower 16 bits are the pixel value sent by the sensor board down the stack on the video channel. By selecting the amount of shift performed the binary point of the gain coefficient is positioned. In the machine vision community the dark field correction is sometimes called the PSNU or DSNU which are acronyms for Photo-receptor Static Non-Uniformity or Dark Signal Non-Uniformity. The gain correction is sometimes called the PRNU which stands for Photo-receptor Response Non-Uniformity.

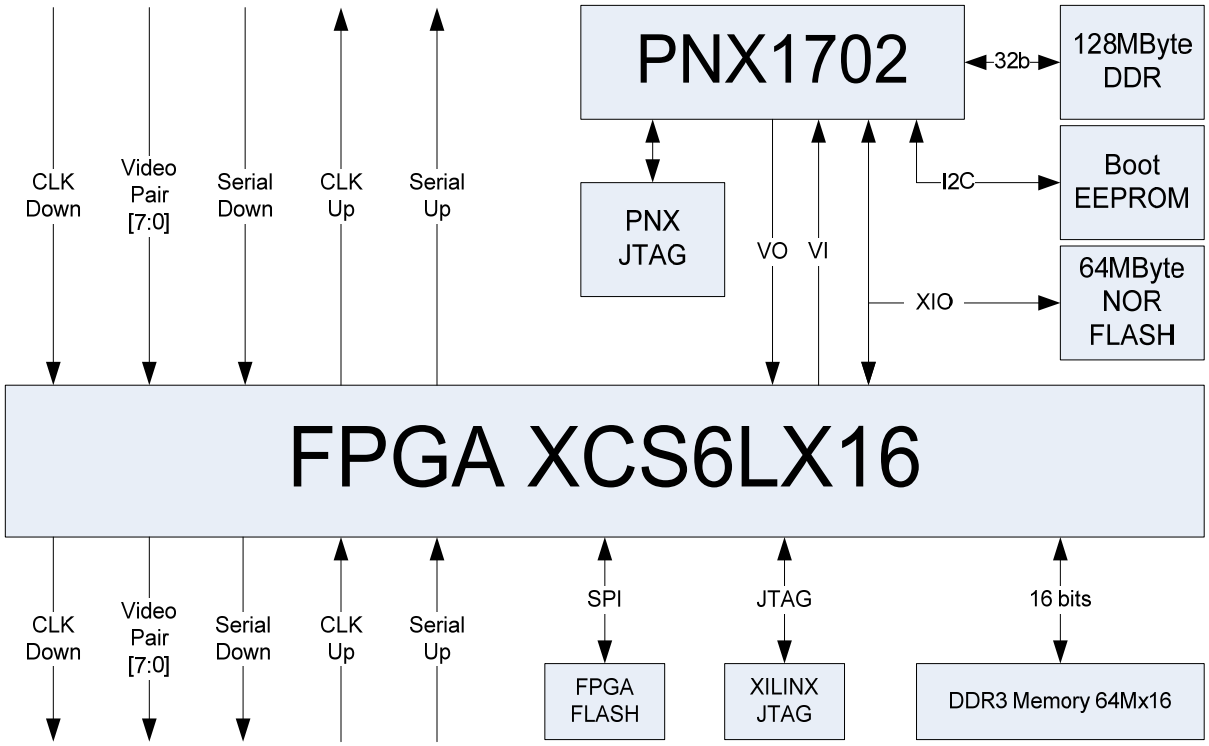
Sensor Board FPGA



The sensor FPGA block diagram is shown above.

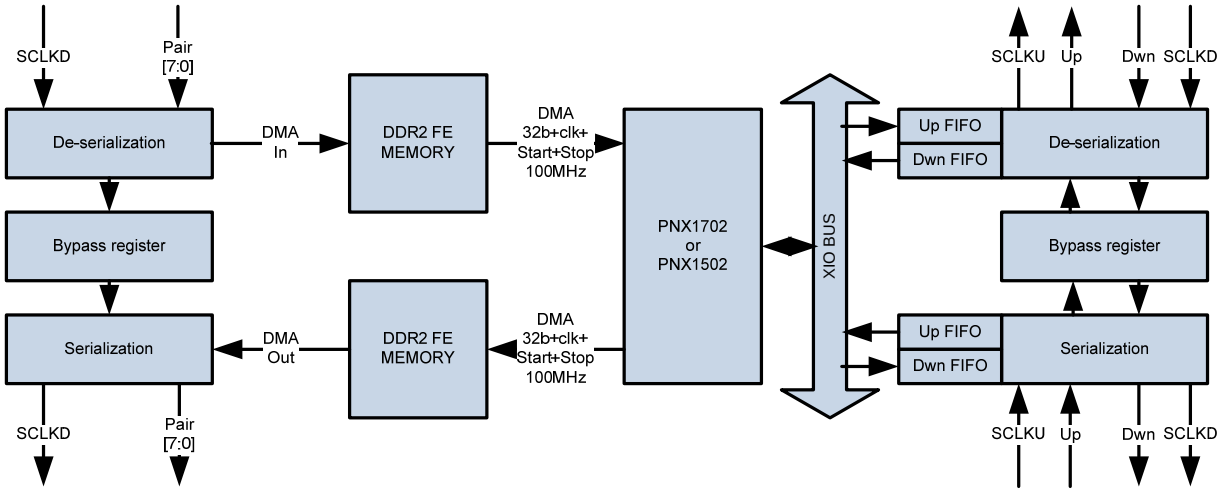
CPU BOARD

Block Diagram



A block diagram of the CPU board is shown above. The CPU board is designed to process video from the sensor board (or any board above it in the stack). The CPU board is used in ‘smart’ camera applications. The CPU can process video, and control all the features supported by the camera, which include sending video to the output, toggle outputs, monitor inputs, sends data to the serial interface or to the GigE interface. The CPU boots from an EEPROM via I2C. The boot loads and executes a program from the 64MB NOR flash. The NOR Flash can be read or written by the processor so that parameter and result data can survive power failures. The CPU can receive video from the stack above it, modify the video and send the modified video down to the next board in the stack.

CPU Board FPGA



A block diagram of the CPU FPGA is shown above.

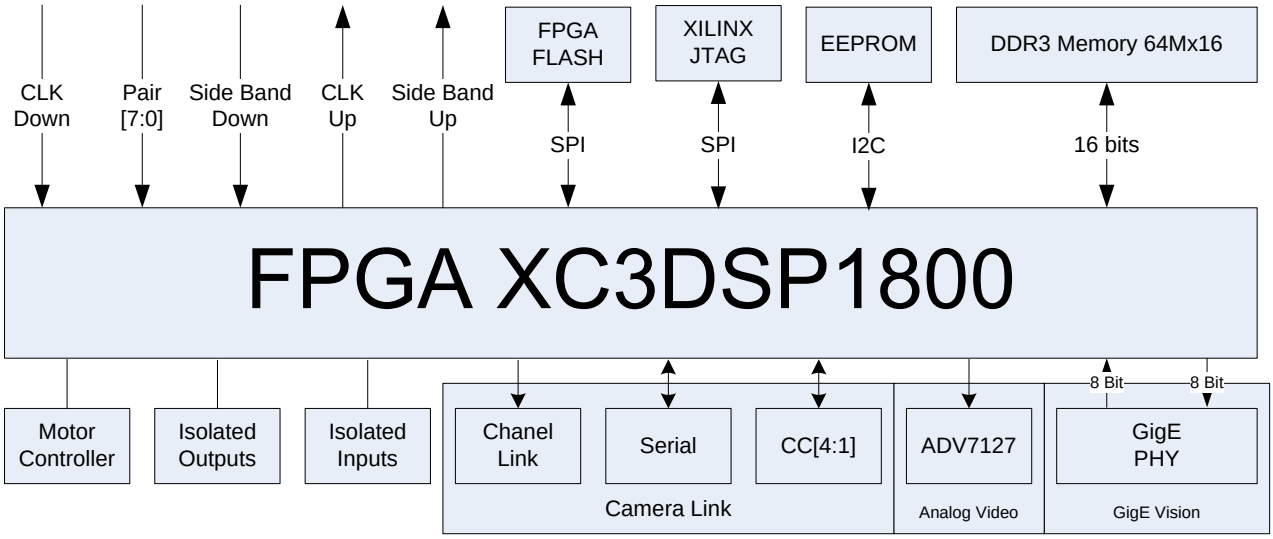
The video interface can operate in three modes.

- The first mode is the video can completely bypass the CPU card without going to the CPU. This is the default mode which is used during booting of the CPU.
- The second mode is the video can be sent to the CPU and at the same time forwarded to the next board in the stack. This mode is used when the CPU is processing the video at the same time it is being output to the board below.
- The third mode the CPU receives the video from the board above, and provides video to the board below. This mode is used when the CPU must modify the video in some way.

The serial channel operates by passing cells up or down the stack which are not targeted for the CPU board FPGA or the CPU. The destination of the cell determines which direction it is routed. Cells that target the FPGA on the CPU board read and write registers in the FPGA. Cells that target the CPU are placed in an input FIFO and can be read by the CPU. In turn the CPU can send cells which can target any of the other boards in the stack. The capability of the CPU to send and receive arbitrary cells gives it control of all the registers in the camera.

I/O BOARD

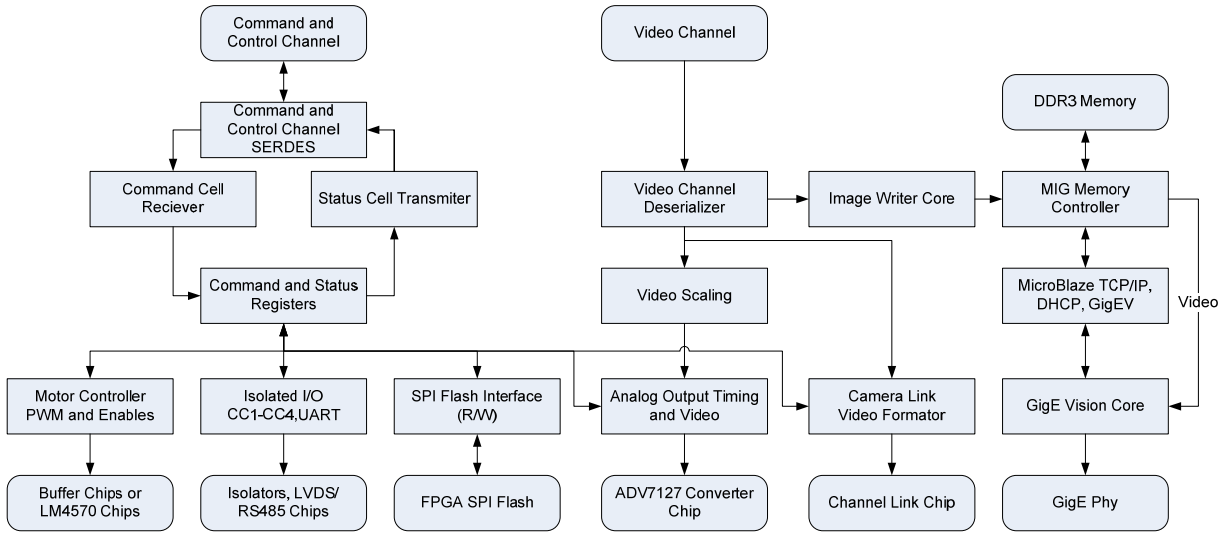
Block Diagram



The I/O board provides interfaces to the world outside the camera. Several interfaces are provided, which can be controlled by the CPU board if it is present, or in dumb camera applications the interfaces are configured to respond as requested by the customer, for example the camera link interface receives the video from the sensor board when a dumb camera link camera is required. The motor controller contains 3 DC motor controllers which can be used to drive small DC motors. The board uses National Semiconductor LM570 motor controllers. There are 6 optically isolated I/O pins can all be configured as outputs. Four of the isolated I/Os can be configured as inputs. The isolated I/O pins use PS2801-1 opto-isolators. The outputs can switch a maximum of 50 ma, and can tolerate up to 80 volts (when off). The outputs provide a 47 ohm resistor and a protection diode in series with the opto-isolators transistor. The optically isolated inputs can tolerate up to 75 volts, and are current limited to 20ma. The input on threshold is 5 volts. The input off threshold is below 1 volt.

The Camera Link interface provides a base camera link configuration. The serial port defaults to 9600 baud and can be programmed to higher baud rates (up to 115,200 baud). The CC1 signal is configured as a trigger input on the dumb camera. On the smart camera they can be configured as inputs or outputs, using LVDS or RS485 style drivers and receivers (order option). The ADV7127 is a 10 bit DAC which can be used to generate a one volt analog video output which is RS343 sync on green compatible. The ADV7127 can be clocked at up to 240 MHz. The GigE interface uses a Marvel 88E1111 phy. The FPGA can be ordered with GigE vision compatible firmware allowing the camera to interoperate with other GigE vision cameras, or it can be used in a smart camera as a gigabit Ethernet interface by the CPU.

I/O Board FPGA



The diagram above shows the I/O board FPGA Firmware.

INTERFACE CONNECTORS

Power Connector

The power connector is a Hirose 12 pin cylindrical connector HR10A-10R-12PB. The pin out of the connector is application dependent and is shown in the table below.

Pin	Option 1	Option 2	Option 3
1	Ground	Ground	Ground
2	+5 Volts In	+5 Volts In	+5 Volts Input Power
3	-15 Volts In	GPIO to Sensor	-15 Volts (May be changing to optional)
4	+15 Volts In	+15 Volts In	+15 Volts (May be changing to optional)
5	Motor 3 negative	+/-15V Ground	Ground
6	Motor 1 positive	CC1 positive	No Connection
7	Motor 1 negative	CC1 negative	No Connection
8	Motor 3 positive	Ground	Ground
9	Motor 2 positive	CC2 Positive	No Connection
10	Ground	Ground	Ground
11	+5 Volts In	+5 Volts In	+5 Volts Input Power
12	Motor 2 negative	CC2 negative	No Connection

Power Requirements

+5 Volts at 3 amps max.
+15 Volts at 100 ma max.

Typical power dissipation for the CPU version is 14 watts.

Camera Link Connector

The camera can be built with a SDR26 connector which can be used to provide a base Camera Link interface, or it can be used to provide general purpose I/O. The optically Isolated I/O is optionally provided on this connector. The pin out of this connector is:

Pin	Signal	Optional	Optional	Signal	Pin
1	Ground	Ground	Ground	Ground	14
2	Tx Out 0 Negative	Optical Out 0 Positive	Optical Out 0 Negative	Tx Out 0 Positive	15
3	Tx Out 1 Negative	Optical Out 1 Positive	Optical Out 1 Negative	Tx Out 1 Positive	16
4	Tx Out 2 Negative	Not used	Not used	Tx Out 2 Positive	17
5	Tx Clock Negative	Not used	Not used	Tx Clock Positive	18
6	Tx Out 3 Negative	Not used	Not used	Tx Out 3 Positive	19
7	Sertc Positive	RS485 Rx Positive	RS485 Rx Negative	Sertc Negative	20
8	Sertfg Negative	RS485 TxNegative	RS485 Tx Negative	Sertfg Positive	21
9	CC1 Negative	Optical In 0 Negative	Optical In 0 Positive	CC1 Positive	22
10	CC2 Positive	Optical In 1 Positive	Optical In 1 Negative	CC2 Negative	23
11	CC3 Negative	Optical In 2 Negative	Optical In 2 Positive	CC3 Positive	24
12	CC4 Positive	Optical In 3 Positive	Optical In 3 Negative	CC4 Negative	25
13	Ground	Ground	Ground	Ground	26

The optical isolated inputs can be configured as additional outputs if required. Be aware that the positive and negative connections are as indicated for historical reasons relating to the history of the Camera Link interface. Be absolutely sure how your camera is configured before connecting to this connector. This connector can be configured in many ways one pin pair at a time.

GigE connector

The GigE connector is a standard RJ45 connector for use with UTP CAT5E or CAT6 cable. Its pin out follows the Ethernet standards and is given here for reference.

Pin	Signal
1	A+
2	A-
3	B+
4	C+
5	C-
6	B-
7	D+
8	D-

JTAG Connectors

Inside the camera are JTAG connectors for connection to Xilinx and CPU JTAG controllers / cables. The JTAG connectors use two small connectors.The pin out of the JTAG connects is board dependent but follows the following table generally:

Pin	FPGA Signal	CPU Signal
Ja-1	Ground	Ground
Ja-2	TDO	TDO
Ja-3	TMS	TDI
Ja-4	NC	RESET~
Jb-1	Ground	Ground
Jb-2	TCK	TCK
Jb-3	TDI	TMS
Jb-4	+3.3 Volts	+3.3 Volts

The standard camera does not provide access to these connectors. Please contact the factory should you wish access to these connectors.

Note: Opening the camera case violates the warrantee of the camera.

CAMERA OPTIONS

The Camera can be purchased with several different combinations of options. The standard configurations are Camera Link, GigE Vision, or Analog.

GIGE VISION INTERFACE

The GigE interface provides TCP/IP access and GigE Vision access to the camera. GigE Vision standard 1.0 is supported by the camera.

Boot

On power up the camera gets its IP number via three different schemes. If a static IP has been programmed the camera uses the value programmed. If the camera has DCHP enabled an IP address will be obtained for a responding DCHP server. If the DCHP server does not respond, or if DCHP is disabled the camera will use Auto IP. Once the camera has an IP address it will wait for a discovery broadcast by a GigE Vision server on its subnet.

Discovery

The camera will respond to a discovery request as specified by the GigE Vision Standard. It is possible to bypass discovery and go directly to a command / control connection phase, if the camera is already known to a GigE server.

Command and Control Connections

A GigE server can make a command and control connection to the camera. Access to the required GigE vision registers and to the Cell interface to the camera is provided via the GigE vision interface.

Streaming Connections

The streaming interface supports a single channel of streaming data with packet sizes negotiable during connector, or with fixed packet sizes. In most applications it is not necessary to modify the packet size manually.

Events

The GigE interface supports Events. The camera can provide a triggered event, or an arbitrary event if the camera is a smart camera (has a CPU board).

Triggering

The camera may be triggered by hard wiring to the CC1, or opto input 0 interfaces. In addition the camera may be triggered via the GigE connection. Finally the camera may operate on an internal trigger generated by the CPU, or via the free running trigger option.

CAMERA LINK INTERFACE

The camera provides a standard base camera link interface. The format of the interface is programmable 8, 10, 12, 14 or 16 bits per pixel one tap, or 8, 10 or 12 bits two taps. CC1 is used as the trigger interface, and the serial link defaults to 9600 baud, no parity, 8 bits and one stop bit.

Channel Link

The channel link interface operates at 62.5 MHz typically, though this can be modified should the customer require.

Serial Link

The serial link follows the camera link standard and supports up to 115,200 Baud. The serial link receivers and drivers can be built with RS485 compatible drivers and receivers. Note, these two pairs do not support multi drop operation, that is the drivers are always enabled.

CC1 to CC4 inputs

The CC1 input is typically used for triggering, but all four of the inputs can be enabled as trigger sources should it be needed. These inputs can be used to signal the CPU in a smart camera application. Internally when these input change state a cell is sent to the sensor board to trigger a frame.
The CC1 to CC4 inputs can be built with LVDS or RS485 drivers and receivers. In both cases direction and enable is provided, so these pins can implement bi-directional signaling, and true RS485 serial interfaces. Please contact the factory if you wish to use these pins as outputs.

ANALOG INTERFACE

The analog interface provides a RS343 compatible output which can be programmed to output the captured monochrome (or Bayer Pattern Monochrome) image as a VCBS (1 volt composite sync) analog signal. The characteristics are programmable via register 0x0010 to 0x0017 of the I/O board.

SPECIAL INTERFACES

Several special interfaces are provided.

Serial Interface

The serial interface is present in all versions of the camera. When the camera is powered up the serial interface provides access to the command and control channel in the camera, via a byte serial encoding of command and control Cells.

Motor Controller Output

The motor control outputs can be configured to drive DC motors directly, or provide a 50 Hz TTL PWM signal for use with servo motors. The pulse width is programmable from 1 to 2 msec in 256 steps.

Isolated input and outputs

The isolated inputs and outputs are provided on the SDR26 interface connector and replace camera link signals. The four inputs on the CC1-CC4 signals can be configured as optically isolated outputs. The two dedicated outputs use the channel link signals so if they are ordered the channel link interface is not supported

SOFTWARE AND FIRMWARE

Several software and firmware components are available to support the use of the camera.

FCCM PROGRAM

The FCCM (Fast Camera Control Modular) program provides a direct interface to the Cell interface internal to the camera. The FCCM program uses the serial interface provided with the camera and accesses the serial interface using a camera link style serial interface DLL. The FCCM program uses a special mode of the I/O board FPGA so that it can send and receive cells on the internal serial channel of the camera.

Subsystem	Command	Function
fccm	**B**	**Load Com DLL**
fccm	**C**	**Send a Cell**
fccm	**F**	**Terminal Mode**
fccm	**V**	**Set Verbosity**
fccm	**X**	**Exit Program**
fccm	**?**	**Help**
fccm	**!**	**Dos command**
fccm	**Timeout**	**Set timeout in msec**
sensor	**Sensd**	**Dump Sensor Registers**
sensor	**Sensreg**	**Read or Write to Sensor Register**
sensor	**sspie <reg> [<value>]**	**Erase Sensor SPI Flash**
sensor	**Sspipf**	**Program file to Sensor SPI**

sensor	**sspipm <list of values>**	**Program manual data to Sensor SPI**
sensor	**Sspid**	**Display Sensor SPI data**
sensor	**sspise <start byte address><end byte address>**	**Sensor SPI Sector Erase**
sensor	**Sspiid**	**Read Sensor SPI Id numbers**
sensor	**sspi2f**	**Output SPI contents to file**
sensor	**Ogtr**	**Offset Gain Table Read**
sensor	**Ogtw**	**Offset Gain Table Write**
sensor	**Ogtm**	**Offset Gain Table Manual**
sensor	**Afeinit**	**Initialized AFE Registers**
sensor	**Afew**	**Write to AFE register**
sensor	**Afer**	**Display last written value AFE**
sensor	**Afem**	**Set AFE debug mux**
sensor	**Afed**	**Dump All AFE registers**
sensor	**Afetp**	**Turn On/off AFE Test pattern**
sensor	**Pboot**	**Program register settings in SPI flash**
sensor	**ad5621 <5.0 to 15.0 float>**	**Set Subtrate voltage(Ad5621 DAC)**
sensor	**resolution <N cols><N rows>**	**Set Sensor Resolution**
sensor	**v1t <rising><falling>**	**Set V1T edge delays**
sensor	**v2t <rising><falling>**	**Set V2T edge delays**
sensor	**v3t <rising><falling>**	**Set V3T edge delays**
sensor	**v4t <rising><falling>**	**Set V4T edge delays**
sensor	**v1b <rising><falling>**	**Set V1B edge delays**
sensor	**v2b <rising><falling>**	**Set V2B edge delays**
sensor	**v3b <rising><falling>**	**Set V3B edge delays**
sensor	**v4b <rising><falling>**	**Set V4B edge delays**
sensor	**lp <clocks>**	**Set line period**
sensor	**fp <lines>**	**Set frame period**

sensor	**exp <lines>**	**Set Exposure**
sensor	**mclk1 0=aligned to P6**	**Set AFE1:MCLK delay**
sensor	**mclk2 0=aligned to P6**	**Set AFE2:MCLK delay**
sensor	**trig <0=free run, 1=Single Edge, 2=Exp-level>**	**Set Trigger Mode**
sensor	**tpol <0=Active low/falling 1=active high/rising**	**Set Trigger Polarity**
sensor	**Tstat**	**Show trigger status**
sensor	**Cds**	**Set CDS Points**
sensor	**Avgdir**	**Avg images on disk**
sensor	**Mkgain**	**Make Quadrant Gain Image**
io	**Iod**	**Dump IO Registers**
io	**Ioreg**	**Read/Write IO Registers**
io	**Ispie**	**Erase I/O SPI Flash**
io	**Ispipf**	**Program file to I/O SPI**
io	**Ispipm**	**Program manual data to I/O SPI**
io	**Ispid**	**Display I/O SPI data**
io	**Ispise**	**I/O SPI Sector Erase**
io	**Ispiid**	**Read I/O SPI Id numbers**
io	**Ispiws**	**Write the NOV status bits**
shell	**Help**	**Get Help**
shell	**History**	**Display/use command history**

COMMAND AND CONTROL CELLS

The command and control channel in the camera uses a ‘Cell’ based command and control protocol. A ‘Cell’ is defined to be a header followed by from 1 to 16 words of payload data. The follow ‘rules’ define the format of the cells:

1. All frame cells will contain 16 pixels(16 bit) unless it's the last cell of a frame.
2. When the video or command and control channels are idle, a framing pattern is. The idle pattern for the video channel is 16'hA5A5. The idle pattern for the command and control channel is 16'hA55A. The idle pattern is sent between cells. At least one idle word is sent between every cell.
3. The first 16-bit word of each cell will be a header word and will contain the following fields: Reserved (Rsvd[3:0]), Destination Device Address (DDA[3:0]), Cell Type (CT[3:0]), and a Length (L[3:0]). The format of the header[15:0] is as follows: {Rsvd[3:0], DDA[3:0], CT[3:0], L[3:0]}

4. The reserved field (Rsvd[3:0]) of the header will must contain a value of zero. This value is used to detect the header after a sequence of idle words.
5. The destination device address field (DDA[3:0]) of the header can contain the following values 0x0 == Sensor_FPGA, 0x1 == CPU_FPGA, 0x2 == NXP CPU, 0x3 == IO_FPGA, 0x4 == Serial interface on the I/O board. All other destination device address's are reserved and will be routed to the CPU (if present). The destination address is not used by the video channel.
6. The cell type field (CT[3:0]) of the header can contain the following values: 0x1 == TOF Frame Cell, 0x2 == INT Frame Cell, 0x3 == EOF Frame Cell, 0x4 == SOL Frame Cell, 0x5 == EOL Frame Cell, 0x6 == Write Cell, 0x7 == Read Cell, 0x8 == Read Response Cell, 0x9 == Write Offset-Gain Table Cell, 0xA == Read Offset-Gain Table Cell, 0xB == Read Response Offset-Gain Table Cell. All other values are reserved and will be routed to the NXP CPU.
7. The length field (L[3:0]) denotes the length of the payload, excluding the header word, minus one. A length of zero means one payload word, and so on up to 15 which means 16 payload words.
8. The maximum cell length will be 17 words, which will contain one header word and 16 payload words.
9. Each cell type will contain a unique payload format.
    a. TOF (Top of Frame) Cell Format:
       Frame Cell Format:
       WORD0: header[15:0]
       WORD1: frame sequence number[15:0]
       WORD2: Timer [15:0]
       WORD3: Timer [31:16]
    b. Other Frame Cell Format:
       WORD0: header[15:0]
       …
       WORDN: pixel[15:0]
    c. Write Cell Format:
       WORD0: header[15:0]
       WORD1: local_write_address[15:0]
       WORD2: local_write_data[15:0]
    d. Read Cell Format:
       WORD0: header[15:0]
       WORD1: local_read_address[15:0]
       WORD2: {reserved[11:0], return_device_address[3:0]}
    e. Read Response Cell:
       WORD0: header[15:0]
       WORD1: {reserved[11:0], responder_device_address[3:0]}
       WORD2: local_read_address[15:0]
       WORD3: read_data[15:0]
    f. Write Offset-Gain Table Cell Format:
       WORD0: header[15:0] (6 Pix, MAX)
       WORD1: 2'd0, start_line_num[10:0] //MAX is 1706
       WORD2: 2'd0, start_pix_num[10:0] //MAX is 1264
       WORD3: 2'd0, pix0_offset[13:0]
       WORD4: 2'd0, pix0_gain[13:0]
       WORD5: 2'd0, pix1_offset[13:0]
       WORD6: 2'd0, pix1_gain[13:0]
       WORD7: 2'd0, pix2_offset[13:0]
       WORD8: 2'd0, pix2_gain[13:0]
       WORD9: 2'd0, pix3_offset[13:0]
       WORDA: 2'd0, pix3_gain[13:0]
       WORDB: 2'd0, pix4_offset[13:0]
       WORDC: 2'd0, pix4_gain[13:0]

WORDD: 2'd0, pix5_offset[13:0]
WORDE: 2'd0, pix5_gain[13:0]

g. Read Offset-Gain Table Cell Format:
WORD0: header[15:0] (6 Pix, MAX)
WORD1: 2'd0, start_line_num[10:0] //MAX is 1706
WORD2: 2'd0, start_pix_num[10:0] //MAX is 1264
WORD3: {reserved[8:0], burst_size[2:0], return_device_address[3:0]}

h. Read Response Offset-Gain Table Cell Format:
WORD0: header[15:0] (6 Pix, MAX)
WORD1: {reserved[11:0], responder_device_address[3:0]}
WORD2: 2'd0, start_line_num[10:0]
WORD3: 2'd0, start_pix_num[10:0]
WORD4: 2'd0, pix0_offset[13:0]
WORD5: 2'd0, pix0_gain[13:0]
WORD6: 2'd0, pix1_offset[13:0]
WORD7: 2'd0, pix1_gain[13:0]
WORD8: 2'd0, pix2_offset[13:0]
WORD9: 2'd0, pix2_gain[13:0]
WORDA: 2'd0, pix3_offset[13:0]
WORDB: 2'd0, pix3_gain[13:0]
WORDC: 2'd0, pix4_offset[13:0]
WORDD: 2'd0, pix4_gain[13:0]
WORDE: 2'd0, pix5_offset[13:0]
WORDF: 2'd0, pix5_gain[13:0]

10. In order to synchronize both ends of the command and control channel, all cells are transmitted with at least one idle word between them. The receive FSM (Finite State Machine) should wait in idle for at least one idle to cell transition. Presumably if a loss of synchronization occurs, resynchronization will be accomplished in short order.

Note: On the CPU board Frame Cells are routed via a register setting. The register setting values are: 0x0 == frames are forwarded to the IO, 0x1 == frames are forwarded to the CPU && IO, 0x2 == frames are forwarded to the CPU (Vi_d) and the frames from the CPU (Vo_d) are forwarded to the IO. The CSR word is at address 0x0000 of the CPU FPGA.

SENSOR BOARD

The following registers are supported by the sensor board FPGA. The registers are programmed by Cell messages or by the power up initialization script recorded in the SPI flash.

REGISTER	R/W/RO	BITS	Function
0x0000	**R/W**	**[0]**	**Trigger Enable**
0x0000	**R/W**	**[1]**	**Trigger camera if written with a one**
0x0000	**R/W**	**[15:2]**	**AD5621 data**
0x0001	**R/W**	**[3:0]**	**AFE1 Channel**
0x0001	**R/W**	**[11:4]**	**AFE1 Address**
0x0001	**R/W**	**[14:11]**	
0X0001	**R/W**	**[15]**	**AFE1 Write Enable**

0X0002	**R/W**	**[11:0]**	**AFE1 Data**
0x0002	**R/W**	**[15:12]**	
0x0003	**R/W**	**[3:0]**	**AFE2 Channel**
0x0003	**R/W**	**[11:4]**	**AFE2 Address**
0x0003	**R/W**	**[14:12]**	**Write Enable**
0x0003	**R/W**	**[15]**	**AFE2 Write Enable**
0x0004	**R/W**	**[11:0]**	**Reg[1].[11-0] AFE2 Data**
0x0004	**R/W**	**[15:12]**	**Debug Mux Control**
0x0005	**R/W**	**[1:0]**	**CCD Readout Mode**
0x0005	**R/W**	**[7:4]**	**Horizontal Shift**
0x0005	**R/W**	**[15:8]**	
0x0006	**R/W**	**[15:0]**	**Sensor ROI Horizontal Start**
0x0007	**R/W**	**[15:0]**	**Sensor ROI Horizontal End**
0x0008	**R/W**	**[15:0]**	**Sensor ROI Vertical Start**
0x0009	**R/W**	**[15:0]**	**ROI Vertical End**
0x000A	**R/W**	**[3:0]**	**Correction Gain Shift Factor**
0x000A	**R/W**	**[7:4]**	
0x000A	**R/W**	**[8]**	**Enable Gain Correction**
0x000A	**R/W**	**[11:9]**	
0x000A	**R/W**	**[12]**	**Enable Offset Correction**
0x000B	**R/W**	**[0]**	**Enable CCD Driver Power**
0x000B	**R/W**	**[1]**	**Load AD5621 Enable**
0x000B	**R/W**	**[2]**	**CCD State machine enable**
0x000B	**R/W**	**[3]**	**CCD Test Pattern Enable**
0x000B	**R/W**	**[4]**	**Enable AFE1 test patter**
0x000B	**R/W**	**[9:5]**	
0x000B	**R/W**	**[14:10]**	**Pixel Write Delay**
0x000B	**R/W**	**[15]**	
0x000C	**R/W**	**[15:0]**	**Line Gap Count**

0x000D	**R/W**	**[15:0]**	**SPI Flash Chip Enable**
0x000E	**R/W**	**[15:0]**	**SPI Flash Even bytes**
0x000F	**R/W**	**[15:0]**	**SPI Flash Odd bytes**
0x0010	**RO**	**[3:0]**	**CCD State machine state**
0x0010	**RO**	**[4]**	**AFE2 VD**
0x0010	**RO**	**[5]**	**AFE1 VD**
0x0010	**RO**	**[6]**	**AFE2 HD**
0x0010	**RO**	**[7]**	**AFE1 HD**
0x0010	**RO**	**[15:8]**	
0x0017	**R/W**	**[15:0]**	**Line Period in pixel clocks**
0x0018	**R/W**	**[15:0]**	**Frame Period in lines**
0x0019	**R/W**	**[15:0]**	**Exposure in lines**
0x001A	**R/W**	**[1:0]**	**Reg [26].[1-0] AFE1 MCLKA Phase**
0x001A	**R/W**	**[3:2]**	**Reg[26].[3-2] AFE2 MCLKA Phase**
0x001A	**R/W**	**[5:4]**	**Reg[26].[5-4] Trigger Mode**
0x001A	**R/W**	**[6]**	**Reg[26].[6] Use VSUB Shutter Enable**
0x001A	**R/W**	**[7]**	**Reg[26].[7] Trigger Active High Enable**
0x001A	**R/W**	**[8]**	**Reg[26].[8] CC1 Trigger Enable**
0x001A	**R/W**	**[9]**	**Reg[26].[9] CC2 Trigger Enable**
0x001A	**R/W**	**[10]**	**Reg[26].[10] CC3 Trigger Enable**
0x001A	**R/W**	**[11]**	**Reg[26].[11] CC4 Trigger Enable**
0x001A	**R/W**	**[15:12]**	
0x001B	**R/W**	**[3:0]**	**V1T Rising Edge Delay**
0x001B	**R/W**	**[7:4]**	**V1T Falling Edge Delay**
0x001B	**R/W**	**[11:8]**	**V2T Rising Edge Delay**
0x001B	**R/W**	**[15:12]**	**V2T Falling Edge Delay**
0x001C	**R/W**	**[3:0]**	**V3T Rising Edge Delay**
0x001C	**R/W**	**[7:4]**	**V3T Falling Edge Delay**
0x001C	**R/W**	**[11:8]**	**V4T Rising Edge Delay**

0x001C	**R/W**	**[15:12]**	**V4T Falling Edge Delay**
0x001D	**R/W**	**[3:0]**	**V1B Rising Edge Delay**
0x001D	**R/W**	**[7:4]**	**V1B Falling Edge Delay**
0x001D	**R/W**	**[11:8]**	**V2B Rising Edge Delay**
0x001D	**R/W**	**[15:12]**	**V2B Falling Edge Delay**
0x001E	**R/W**	**[3:0]**	**V3B Rising Edge Delay**
0x001E	**R/W**	**[7:4]**	**V3B Falling Edge Delay**
0x001E	**R/W**	**[11:8]**	**V4B Rising Edge Delay**
0x001E	**R/W**	**[15:12]**	**V4B Falling Edge Delay**
0x001F	**R/W**	**[0]**	**CC1 State**
0x001F	**R/W**	**[1]**	**CC2 State**
0x001F	**R/W**	**[2]**	**CC3 State**
0x001F	**R/W**	**[3]**	**CC4 State**
0x001F	**R/W**	**[4]**	**CC1 Changed**
0x001F	**R/W**	**[5]**	**CC2 Changed**
0x001F	**R/W**	**[6]**	**CC3 Changed**
0x001F	**R/W**	**[7]**	**CC4 Changed**
0x001F	**R/W**	**[15:8]**	
0x0020	**RO**	**[15:0]**	**AFE1 DCLK Frequency**
0x0021	**RO**	**[15:0]**	**AFE2 DCLK Frequency**
0x0022	**RO**	**[15:0]**	**AFE1 CCLK Frequency**
0x0023	**RO**	**[15:0]**	**AFE2 CCLK Frequency**
0x0024	**RO**	**[15:0]**	**Frame Counter**
0x0025	**RO**	**[15:0]**	**usec Timer Low**
0x0026	**RO**	**[15:0]**	**usec Timer high**
0x0027	**RO**	**[15:0]**	**usec at last frame low**
0x0028	**RO**	**[15:0]**	**usec at last frame high**

CPU BOARD

XIO REGISTERS

The following registers are in the XIO space of the CPU on the CPU board. The can be read or written by dereferencing points to the XIO base address plus the register offset. The interface is 16 bits.

Register	R/W/RO	Bits	Function
0x0000	R/W	[1:0]	Frame Cell routing Frame Cells are routed via a CSR setting. 0x0 == frames are forwarded to the IO 0x1 == frames are forwarded to the CPU && IO 0x2 == frames are forwarded to the CPU (Vi_d) and the frames from the CPU (Vo_d) are forwarded to the IO.
0x0000	R/W	[15:2]	not used
0x0001	R/W	[15:0]	Bits[15:0] vdi_gap_csr Sets the speed that the FPGA can send FGPI messages larger numbers mean slower.
0x0002	R/W	[0]	Bits[0] Send single frame
0x0002	R/W	[1]	Bits[1] Continous frame
0x0002	R/W	[15:2]	Not used
0x0003	R/W	[10:0]	Start pixel number
0x0003	R/W	[15:11]	not used
0x0004	R/W	[10:0]	End pixel number
0x0004	R/W	[15:11]	not used
0x0005	R/W	[10:0]	Start line number
0x0005	R/W	[15:11]	not used
0x0006	R/W	[10:0]	End line number
0x0006	R/W	[15:11]	not used
0x0007	R/W	[15]	c3_p0_cmd_en Command field contains a memory command
0x0007	R/W	[14:12]	c3_p0_cmd_instr Memory Command
0x0007	R/W	[11:6]	c3_p0_cmd_bl burst length
0x0007	R/W	[5]	c3_p0_wr_en Command is a write
0x0007	R/W	[4:1]	c3_p0_wr_mask Byte line write enables
0x0007	R/W	[0]	c3_p0_rd_en Command is a read
0x0008	R/W	[15:0]	MemoryWriteData[31:16]
0x0009	R/W	[15:0]	MemoryWriteData[15:0]
0x000A	R/W	[15:14]	temp_addr
0x000A	R/W	[13:0]	Memory Address for the command c3_p0_cmd_byte_addr[29:16]
0x000B	R/W	[15:0]	Memory Address for the command c3_p0_cmd_byte_addr[15:0]
0x000C	RO	[15:0]	Reads 0xAAAA;
0x000D	RO	[15:0]	Reads 0x55555;
0x000E	RO	[15:10]	Zero
0x000E	RO	[9]	c3_p0_cmd_empty Memory Controller Command FIFO empty
0x000E	RO	[8]	c3_p0_cmd_full Memory Controller Command FIFO full

0x000E	RO	[7]	c3_p0_wr_full Memory Controller Write Data FIFO full
0x000E	RO	[6]	c3_p0_wr_empty Memory Controller Write Data FIFO empty
0x000E	RO	[5]	c3_p0_wr_underrun Memory Controller Write Data was not available when the write command ran
0x000E	RO	[4]	c3_p0_wr_error Memory Controller Write Error Flag
0x000E	RO	[3]	c3_p0_rd_full Memory Controller Read Data FIFO Full
0x000E	RO	[2]	c3_p0_rd_empty Memory Controller Read Data FIFO Empty
0x000E	RO	[1]	c3_p0_rd_overflow Memory Controller Read Data overflowed
0x000E	RO	[0]	c3_p0_rd_error Memory Controller Read Error Flag
0x000F	RO	[15:14]	Zero
0x000F	RO	[13:7]	c3_p0_wr_count Write FIFO count
0x000F	RO	[6:0]	c3_p0_rd_count Read FIFO count
0x0010	RO	[15:0]	MemoryReadData[31:16]
0x0011	RO	[15:0]	MemoryReadData[15:0]
0x0012	RO	[15:0]	line_in_frame[15:0]
0x0013	RO	[15:0]	pixels_in_frame[11:0]
0x0014	R/W	[15:3]	not used
0x0014	R/W	[2]	Interrupt Mask Default 1 at power-up. When 0 FIFO not empty will generate IRQ(level) on PCI_REQ_B_B
0x0014	R/W	[1]	ClearUntilNextHeader Writing a ‘1’ to this bit will move the next message to start of FIFO. Will auto reset to 0
0x0014	R/W	[0]	Clear FIFO Writing a ‘1’ to this bit will empty/reset FIFO. Will auto reset to 0
0x0015	R/W	[15:1]	not used
0x0015	R/W	[0]	FIFO read enable Writing a ‘1’ to this bit will move the next word to start of FIFO. Will auto reset to 0. FIFO must be popped first before read
0x0016	RO	[15]	DataReady Indicates more 1 or more messages available in FIFO
0x0016	RO	[14]	Overrun Indicates buffer overrun
0x0016	RO	[13]	MessageHeader Indicates if next word in Fifo is a MessageHeader
0x0016	RO	[12:8]	Number of msgs in Fifo Actual number of msgs in FIFO, max is 31 if word count > 31, count is set to 31
0x0016	RO	[7:0]	Number of 16-bit words in Fifo Actual number of 16-bit words in FIFO, max is 255 if word count > 255, count is set to 255
0x0017	RO	[15:0]	sensor serial message fifo read data
0x0018	R/W	[15:4]	not used
0x0018	R/W	[3]	Interrupt Mask Default 1 at power-up. When 0 FIFO not empty will generate IRQ(level) on PCI_REQ_B_B
0x0018	R/W	[2]	ClearUntilNextHeader Writing a ‘1’ to this bit will move the next message to start of FIFO. Will auto reset to 0
0x0018	R/W	[1]	FIFO read enable Writing a ‘1’ to this bit will move the next word to start of FIFO. Will auto reset to 0. FIFO must be popped first before read
0x0018	R/W	[0]	Clear FIFO Writing a ‘1’ to this bit will empty/reset FIFO.
0x0019	R/W	[15:1]	not used
0x0019	R/W	[0]	FIFO read enable Writing a ‘1’ to this bit will move the next word to start of FIFO. Will auto reset to 0. FIFO must be popped first before read
0x001A	RO	[15]	DataReady Indicates more 1 or more messages available in FIFO
0x001A	RO	[14]	Overrun Indicates buffer overrun

0x001A	**RO**	**[13]**	**MessageHeader Indicates if next word in Fifo is a MessageHeader**
0x001A	**RO**	**[12:8]**	**Number of msgs in Fifo Actual number of msgs in FIFO, max is 31 if word count > 31, count is set to 31**
0x001A	**RO**	**[7:0]**	**Number of 16-bit words in Fifo Actual number of 16-bit words in FIFO, max is 255 if word count > 255, count is set to 255**
0x001B	**RO**	**[15:0]**	**IO serial message fifo read data**
0x001C	**RO**	**[15:0]**	**FPGA revision**
0x001D	**RO**	**[15:4]**	**not used**
0x001D	**RO**	**[3]**	**Cell error status error cell, not sent not flushed.**
0x001D	**RO**	**[2]**	**Cell OK status Cell in right format and sent**
0x001D	**R/W**	**[1]**	**Reset FIFO Writing a '1' to this bit will empty/reset FIFO. Will auto reset to 0**
0x001D	**R/W**	**[0]**	**Transmit Cell Writing a '1' to this bit will send the message in the FIFO. The FIFO should a complete message in it. Will auto reset to 0**
0x001E	**R/W**	**[15:0]**	**Cell message word Writing the register will push cell word into FIFO**
0x001F	**RO**	**[15:4]**	**not used**
0x001F	**RO**	**[3]**	**Cell error status error cell, not sent not flushed.**
0x001F	**RO**	**[2]**	**Cell OK status Cell in right format and sent**
0x001F	**R/W**	**[1]**	**Reset FIFO Writing a '1' to this bit will empty/reset FIFO. Will auto reset to 0**
0x001F	**R/W**	**[0]**	**Transmit Cell Writing a '1' to this bit will send the message in the FIFO. The FIFO should a complete message in it. Will auto reset to 0**
0x0020	**R/W**	**[15:0]**	**Cell message word Writing the register will push cell word into FIFO**

FC XXX CPU FIRMWARE

CPU Board Process Cell API

The FCXXX camera is a very modular camera, which consists of a number of building blocks. At the minimum, the camera has an I/O.board, that connects to the host via Camera-Link or GigE and a Sensor-board. The camera can be extended with a cpu-board, that gets inserted between the i/o-board and the sensor-board. Currently a PNX1702 is used on the CPU-board. The main-purpose of the cpu-board is to add special functionality, such as image-handling and statistics. The pnx-cpu can be programmed by the user, to add special functions to the application, that the camera is used in. The programming environment is PNX SDK + JTAG for debugging.

VIDEO-BUS AND MESSAGE-BUS

The FCXXXhas a number of busses, that moves video from sensor to i/o-board and/or cpu-board and messages between i/o-board, cpu-board and sensor-board.

CPU video in/out and message handling

The cpu receive video from CPU-FPGA to FGPI 32-bits wide, if FPGA video-routing is set to 2.It sends video to i/o-board via FGPO 32-bits wide -> CPU-FPGA. All cell-messages, that are not video, are handled via registers in CPU_FPGA.

FGPISRVHANDLER

A demo-server for receiving video-frames is provided in **fc205_cpu.c** in

```
void FgpiSrvHandler(void* args)
```

The following functions in same file have been provided to start and stop this server from the debugger shell, for testing:

void FgpiStartServer(TCHAR *cmd_line)
void FgpiStopServer(TCHAR *cmd_line)

FGPOSRVHANDLER

A demo-server for sending frames from cpu to i/o-board has been provided in **fc205_cpu.c** in:

void FgpoSrvHandler(void* args)

The following functions in same file have been provided to start and stop this server from the debugger shell, for testing:

void FgpoStartServer(TCHAR *cmd_line)
void FgpoStopServer(TCHAR *cmd_line)

RTCELL ROUTING OF VIDEO

Following function has been provided to set the routing of video-cells from the sensor-board:

void SetRoute(TCHAR *cmd_line)

If no parameters , current setting is displayed.
Following options are available:

rtcell –r0<cr> Set routing to Sensor-board to i/o-board
rtcell –r1<cr> Set routing to Sensor-board to i/o-board + cpu-board
rtcell –r2<cr> Set routing to Sensor-board to cpu-board and cpu-board to i/o-board

EXAMPLE OF VI + VO + ROUTING OF VIDEO-CELLS

For testing from debugger-shell, the following sequence will set routing to sensor->cpu/cpu->i/o-board, start video-out-srv and video-in-srv. The video-in-srv will send every 2nd frame to video-out-srv, which will show up on the output on the i/o-board. The communication between the video-in-srv and video-out-srv is made via a queue. It is the intention that the user can make changes, that suit their application. This is only shown as an example on how to use VI and VO and the setup VI-handling and VI to VO communication.

MSGSRVHANDLER

For handling non-video cell-messages, a message-server-handler has been provided. It is located in **interrupt.c**:

void MsgSrvHandler(void* args)

This server provides handling of the following non-video cell-messages:

Read, write and read-response.

Write-requests are handled differently, depending on the write-address in the cell-messages.
The MsgSrvHandler() is by default started in tmMain() function.

I/O-board opto-electronics change of state

Any changes of state in the opto-electronics on the i/o-board, result in a write cell-message to cpu at addr 0x0, with at copy of the data. The user should expand the following function in **interrupt.c**:

void IoMsgWriteRequest(msg_t *pMsg)

to handle this request.

Write-requests

Same function also handles uart-receive-data write-request and write-data to cpu-storage if write-addr is within 0x1000 – 0x10FF. All other write-requests are ignored.Please see interrupt.h:

typedef struct fc205_cpu_storage

Read-requests

If a read-request is received and read-addr is within 0x1000 – 0x10FF, the data in ReadOut cpu-storage is returned, using read-addr as index. By default index 0(0x1000) contains FPGA-rev-id and index 1(0x1001) contains ROI last used on VideoIn to pnx-cpu. Other can be user-defined as needed.
Please modify following function in **interrupt.c** to your needs:

void IoMsgReadRequest(msg_t *pMsg)

Read-response-request

If pnx-cpu sends a read-request to sensor-board or i/o-board (or host, if uart is not in uart-mode), then pnx-cpu will receive a read-response-package. Please modify following function in **interrupt.c** to your needs:

void IoMsgReadResponseRequest(msg_t *pMsg)

Uart receive chars

If uart in i/o-board is programmed to **uart-mode**, then all data sent from host to uart on i/o-board are re-packaged into cell-messages and forwarded to addr 0x100(if number of chars in cell are even) or addr 0x101(if number of chars in cell are odd).
These cells are extracted into a receive-buffer, for readout by user.
Following 2 functions are provided for checking for chars ready and reading chars:

```
/**
*
* CheckUartTxFifo will read a register on IO board, that shows:
*  Fifo Full, Fifo HalfFull or Fifo Empty
*
* \fn : int CheckUartTxFifo(UInt16 *pIoUartFifoStatus)
*
```

```
* \param : UInt16 *pIoUartFifoStatus        : Placeholder for returning IoUart 0 =
more than half full, 1 = half full and 2 = empty, fifo status: 3 = full
*
*
* \return : TM_OK if no errors else appropriate error code
*
**/
int CheckUartTxFifo(UInt16 *pIoUartFifoStatus)


/**
*
* RxUartCharFC205 returns the number of char requested is available.
* pCount have have actual number of chars returned
* Timeout defines number of MilliSecs to wait
*
* \fn : int RxUartCharFC205(char *pCharBuffer, UInt16 *pCount, int Timeout)
*
* \param : char *pCharBuffer        : Pointer to buffer, where char(s) will be
stored
* \param : UInt16 *pCount           : Number of chars requested. If less, pCount
will reflect return-count
* \param : int Timeout              : Number of MilliSecs to wait for request
*
*
* \return : TM_OK if no errors, else appropriate error code

**/
int RxUartCharFC205(char *pCharBuffer, UInt16 *pCount, int Timeout)
```

Uart transmit chars

Following function is provided for transmitting chars to host:

```
/**
*
* TxUartCharFC205 will send requested number of chars to Host
*
* \fn : int TxUartCharFC205(char *pCharBuffer, UInt16 Count)
*
* \param : char *pCharBuffer
* \param : UInt16 Count
*
*
* \return : TM_OK if no errors else appropriate error code
*
**/
int TxUartCharFC205(char *pCharBuffer, UInt16 Count)
```

SENDING CELL – MESSAGES

Cell – messages can be sent to sensor-board, i/o-board and host(if uart on i/o-board is not in **uart-mode**). The following structs have been created for this purpose:

```
typedef struct MsgWritePacket
{
    msg_header_t    MsgHdr ;                // msg-header
    UInt16          LocalWriteAddress ;     // Local write address at target
    UInt16          LocalWriteData[15] ;    // Data to write at target
} MsgWritePacket_t ;

typedef struct MsgReadPacket
{
    msg_header_t    MsgHdr ;            // msg-header
    UInt16          LocalReadAddress ;  // Local read address at target
    UInt16          ReturnAddress:4 ;   // Return addr of requester
    UInt16          Reserved:12 ;       // Reserved bits
} MsgReadPacket_t ;
```

Following struct has been created for ReadResponse:

```
typedef struct MsgReadResponsePacket
{
    msg_header_t    MsgHdr ;             // msg-header
    UInt16          ResponderAddress:4 ; // Address of responder
    UInt16          Reserved:12 ;        // Reserved bits
    UInt16          LocalReadAddress ;   // Local read address at target
    UInt16          ReadData[14] ;         // Read Data
} MsgReadResponsePacket_t ;
```

Fill in the appropriate fields and call following function in **fc205_cpu.c**:

```
void fc205_send_cell (cell_t *cell)
```

with each of the structs type-casted to 'cell_t'.

command_sensreg_api

A number of functions are provided as examples for doing this:
Fc205_cpu.c:

```
void command_sensreg_api (short addr, short data)
```

The above function takes an **addr** and **data** as parameters and formats a CELL_WRITE package and sends it to the sensor-board.

command_ioreg_api

```
void command_ioreg_api (short addr, short data)
```

The above function takes an **addr** and **data** as parameters and formats a CELL_WRITE package and sends it to the i/o-board.

API – FUNCTIONS

A number of functions have been provided as an api to functionality on the fc205 cpu-board:

GetRoi_api

```
/**
*
* GetRoi gets the Roi in the FPGA, by reading the Roi registers
*
* \fn : int GetRoi(short *StartPix, short *EndPix, short *StartLine, short
*EndLine)
*
* \param1 : short *StartingPixel         : Placeholder for starting pixel in the
line
* \param2 : short *EndingPixel           : Placeholder for ending pixel in the line
* \param3 : short *StartingLine          : Placeholder for starting  line
* \param4 : short *EndingLine            : Placeholder for ending line
*
*
* \return : tm_OK if no errors, else appropriate error
*
**/
int GetRoi_api(UInt16 *StartPix, UInt16 *EndPix, UInt16 *StartLine, UInt16
*EndLine)
```

SetRoi_api

```
/**
```

```
*
* SetRoi_api sets the requested Roi in the FPGA, by defining
* a starting pixel, ending pixel, stating line and ending line
*
* \fn : int SetRoi_api(short StartPix, short EndPix, short StartLine, short
EndLine)
*
* \param1 : short StartingPixel         : Defines starting pixel in the line
* \param2 : short EndingPixel           : Defines ending pixel in the line
* \param3 : short StartingLine          : Defines starting  line
* \param4 : short EndingLine            : Defines ending line
*
*
* \return : tm_OK if no errors, else appropriate error
*
**/
int SetRoi_api(UInt16 StartPix, UInt16 EndPix, UInt16 StartLine, UInt16 EndLine)
```

GetFrameControl_api

```
/**
*
* GetFrameControl_api gets FrameControl addr 2(+0x04 on FPGA_XIO_BASE_ADDR)
*
* \fn : int GetFrameControl_api(UInt16 *FrameControl)
*
* \param : UInt16 *FrameControl             : Address to store value
*
*
* \return : TM_OK if no error else XIO-error
*
**/
int GetFrameControl_api(UInt16 *pFrameControl)
```

CaptureOneFrame_api

```
/**
*
* CaptureOneFrame_api sets bit0 of FrameControl addr 2(+0x04 on FPGA_XIO_BASE_ADDR)
*
* \fn : int CaptureOneFrame_api()
*
* \param : none
*
*
* \return : TM_OK if no error else XIO-error
*
**/
int CaptureOneFrame_api()
```

CaptureContinousFrame_api

```
/**
*
* CaptureContinousFrame_api sets/clears bit1 of FrameControl addr 2(+0x04 on
FPGA_XIO_BASE_ADDR)
*
* \fn : int CaptureContinousFrameOff_api(bool OnOff)
*
* \param1 : Bool OnOff              : 0 = off, 1 = on
*
*
* \return : TM_OK if no error else XIO-error
*
**/
int CaptureContinousFrame_api(Bool OnOff)
```

GetFpgaRevID_api

```
/**
*
* GetFpgaRevID_api return Fpga Rev ID
*
* \fn : int GetFpgaRevID_api(UInt16 *FpgaRevID)
*
* \param : none
*
*
* \return : TM_OK if no errors, else appropritate errorcode
*
**/
int GetFpgaRevID_api(UInt16 *FpgaRevID)
```

SetShowUartMode_api

```
/**
*
* SetShowUartMode_api sets/shows the mode of the Uart to cell-mode or Uart-mode
*
* \fn : int SetShowUartMode_api(UInt16 UartMode)
*
* \param : UInt16 UartMode      : Uart´mode: 0 = cell, 1 = uart, 2 = display only
*
*
* \return : TM_OK if no error, else appropriate error code
*
**/
int SetShowUartMode_api(UInt16 UartMode, UInt16 *pRetUartMode)
```

setupCpuStorageReadOut_api

```
/**
*
* SetupCpuStorageReadOut initializes the CpuStorageReadOut - memory
*
* \fn : int SetupCpuStorageReadOut()
*
* \param : none
*
*
* \return : 0 if o.k. else last errorode
*
**/
int SetupCpuStorageReadOut_api()
```

MsgSrvStart_api

```
/**
*
* MsgSrvStart_api starts message receive - server
*
* \fn : int MsgSrvStart_api()
*
* \param : none
*
*
* \return : TM_OK if no errors, else appropriate error
*
**/
int MsgSrvStart_api()
```

I/O BOARD FIRMWARE

The follow registers are accessible on the I/O board using cells.

Register	R/W/R	Bits	Function

	O		
0x0000	R/W	[0]	CL Clk 33/66 MHz
0x0000	R/W	[1]	CL Clk 42.5/85 MHz
0x0000	R/W	[2]	CL Clk (33/66)/(42/85) MHz
0x0001	R/W	[0]	Optical Output 1
0x0001	R/W	[1]	Optical Output 2
0x0001	R/W	[15:2]	Not used
0x0002	R/W	[0]	Optical Input 1
0x0002	R/W	[1]	Optical Input 2
0x0002	R/W	[2]	Optical Input 3
0x0002	R/W	[3]	Optical Input 4
0x0002	R/W	[4]	Optical Input 1 Changed
0x0002	R/W	[5]	Optical Input 2 Changed
0x0002	R/W	[6]	Optical Input 3 Changed
0x0002	R/W	[7]	Optical Input 4 changed
0x0002	R/W	[15:8]	Not Used
0x0003	R/W	[0]	LED 1
0x0003	R/W	[1]	LED 2
0x0003	R/W	[15:2]	Not Used
0x0004	R/W	[8:0]	Motor 1 PWM
0x0004	R/W	[9]	Motor 1 On
0x0004	R/W	[15:10]	Not Used
0x0005	R/W	[8:0]	Motor 2 PWM
0x0005	R/W	[9]	Motor 2 On
0x0005	R/W	[15:10]	Not Used
0x0006	R/W	[8:0]	Motor 3 PWM
0x0006	R/W	[9]	Motor 3 On
0x0007	R/W	[15:0]	Baud Rate Divisor
0x0008	R/W	[15:0]	Not Used
0x0009	R/W	[15:0]	Not Used
0x000A	R/W	[15:0]	Not Used
0x000B	R/W	[15:0]	Not Used
0x000C	R/W	[3:0]	CL LSB Select
0x000C	R/W	[5:4]	Bits per pixel
0x000C	R/W	[6]	Two Tap Enable
0x000C	R/W	[7]	Enable Header
0x000C	R/W	[15:8]	Not Used
0x000D	R/W	[3:0]	GigE LSB Select
0x000D	R/W	[15:4]	Not Used
0x000E	R/W	[0]	CC1 input
0x000E	R/W	[1]	CC2 input
0x000E	R/W	[2]	CC3 input
0x000E	R/W	[3]	CC4 input
0x000E	R/W	[4]	CC1 Changed Bit
0x000E	R/W	[5]	CC2 Changed Bit

0x000E	**R/W**	**[6]**	**CC3 Changed Bit**
0x000E	**R/W**	**[7]**	**CC4 Changed Bit**
0x000E	**R/W**	**[15:8]**	**Not Used**
0x000F	**R/W**	**[0]**	**Enable Camera Link Output**
0x0010	**R/W**	**[15:0]**	**Analog Output Clocks Per Line**
0x0011	**R/W**	**[15:0]**	**Analog Output Horz. Active**
0x0012	**R/W**	**[7:0]**	**Hsync width**
0x0012	**R/W**	**[15:8]**	**Hsync position**
0x0013	**R/W**	**[15:0]**	**Analog Output Lines Per Frame**
0x0014	**R/W**	**[15:0]**	**Analog Output Vertical Acrive**
0x0015	**R/W**	**[7:0]**	**VSync width**
0x0015	**R/W**	**[15:8]**	**Vsync position**
0x0016	**R/W**	**[0]**	**Analog Out DAC Enable**
0x0017	**R/W**	**[1]**	**Analog Out 150MHz Enable**

SIZE AND COOLING

The smart camerais 63mm x 63mm x 57mm not including the protrusion of the connector bodies. The smart camera dissipates 14 Watts maximum.

The dumb camera (camera without a CPU board) is 63mm x 63mm x 49mm not including the connector bodies. The dumb camera dissipates 8 Watts maximum.

The dumb and smart cameras may be cooled by conduction by mounting it to a cold plate at a maximum of 50 degrees C. Forced are cooling can also be used, in which case the camera requires 50 LFM air flow at a maximum of 50 degrees C.

TROUBLESHOOTING

There are several things you can try before you call FastVision Technical Support for help.

_____ Make sure the computer is plugged in. Make sure the power source is on.

_____ Go back over the hardware installation to make sure that the system is properly installed.

_____ Go back over the software installation to make sure you have installed all necessary software.

_____ Run the Installation User Test to verify correct installation of both hardware and software.

_____ Run the user-diagnostics test for your main board to make sure it's working properly.

_____ Insert the FastVision CD-ROM and check the various Release Notes to see if there is any information relevant to the problem you are experiencing.

FASTVISION TECHNICAL SUPPORT

FastVision offers technical support to any licensed user during the normal business hours of 9 a.m. to 5 p.m. EST. We offer assistance on all aspects of processor board and PMC installation and operation.

CONTACTING TECHNICAL SUPPORT

To speak with a Technical Support Representative on the telephone, call the number below and ask for Technical Support:

Telephone: **603-891-4317**

If you would rather FAX a written description of the problem, make sure you address the FAX to Technical Support and send it to:

Fax: **603-891-1881**

You can email a description of the problem to ***support@FastVision.com***

Before you contact technical support have the following information ready:

_____ Serial numbers and hardware revision numbers of all of your boards. This information is written on the invoice that was shipped with your products.

_____ Also, each board has its serial number and revision number written on either in ink or in bar-code form.

_____ The version of the FASTVIEWER, or FAST CAMERA-DVRP software that you are using.

_____ The type and version of the host operating system, i.e., Windows 98.

_____ Note the types and numbers of all your software revisions, daughter card libraries, the application library and the compiler

_____ Returning Products for Repair or Replacements

Our first concern is that you be pleased with your FastVision products.

If, after trying everything you can do yourself, and after contacting FastVision Technical Support, you feel your hardware or software is not functioning properly, you can return the product to FastVision for service or replacement. Service or replacement may be covered by your warranty, depending upon your warranty. The first step is to call FastVision and request a “Return Materials Authorization” (RMA) number. This is the number assigned both to your returning product and to all records of your communications with Technical Support. When a FastVision technician receives your returned hardware or software he will match its RMA number to the on-file information you have given us, so he can solve the problem you’ve cited.

When calling for an RMA number, please have the following information ready:

_____ Serial numbers and descriptions of product(s) being shipped back

_____ A listing including revision numbers for all software, libraries, applications, daughter cards, etc.

_____ A clear and detailed description of the problem and when it occurs

_____ Exact code that will cause the failure

_____ A description of any environmental condition that can cause the problem

All of this information will be logged into the RMA report so it's there for the technician when your product arrives at FastVision. Put boards inside their anti-static protective bags. Then pack the product(s) securely in the original shipping materials, if possible, and ship to:

FastVision LLC.
71 Spit Brook Road, Suite 200
Nashua, NH 03060
USA

Clearly mark ***the outside of your package:***

Attention **RMA #90XXX**

Remember to include your return address and the name and number of the person who should be contacted if we have questions.

Reporting Bugs

We at FastVision are continually improving our products to ensure the success of your projects. In addition to ongoing improvements, every FastVision product is put through extensive and varied testing. Even so, occasionally situations can come up in the fields that were not encountered during our testing at FastVision.

If you encounter a software or hardware problem or anomaly, please contact us immediately for assistance. If a fix is not available right away, often we can devise a work-around that allows you to move forward with your project while we continue to work on the problem you've encountered.

It is important that we are able to reproduce your error in an isolated test case. You can help if you create a stand-alone code module that is isolated from your application and yet clearly demonstrates the anomaly or flaw.

Describe the error that occurs with the particular code module and email the file to us at:

support@FastVision.com

We will compile and run the module to track down the anomaly you've found.

If you do not have Internet access, or if it is inconvenient for you to get to access, copy the code to a disk, describe the error, and mail the disk to Technical Support at the FastVision address below.

If the code is small enough, you can also:

FAX the code module to us at 603-891-1881

If you are faxing the code, write everything large and legibly and remember to include your description of the error.

When you are describing a software problem, include revision numbers of all associated software.

For documentation errors, photocopy the passages in question, mark on the page the number and title of the manual, and either FAX or mail the photocopy to FastVision.

Remember to include the name and telephone number of the person we should contact if we have questions.

FastVision LLC.
131 Daniel Webster Highway, #529
Nashua, NH 03060
USA

Telephone: 603-891-4317

FAX: 603-891-1881

Web site:
http://www.FastVision.com/

Electronic Mail:
sales@FastVision.com
support@FastVision.com