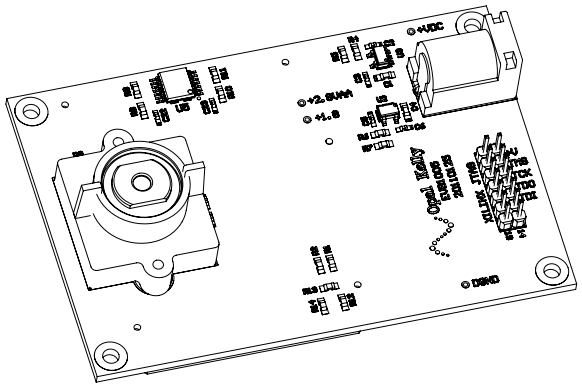
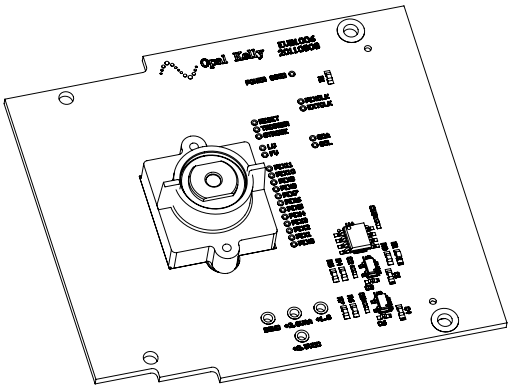
Opal Kelly





EVB100x User's Manual

A 5-Mpixel image capture board for the XEM6010, XEM6110, XEM3050, XEM3010, and Shuttle LX1 FPGA integration modules

The EVB1005 is an image capture module for use with the Opal Kelly XEM6010, XEM6110, XEM3050, and XEM3010 FPGA modules. The EVB1006 is an equivalent capture module for the Shuttle LX1 (XEM6006) or other FMC carrier. Both modules include a Micron MT9P031I12STC 5 mega-pixel color image sensor and necessary power supply circuitry. Designed as evaluation boards for Opal Kelly integration modules, the modules provide an excellent platform for getting accustomed to the FrontPanel SDK.

Software, documentation, samples, and related materials are

Opal Kelly Incorporated
Portland, Oregon
http://www.opalkelly.com

Revision History:

Date	Description
20120101	Initial release.
20121112	Fix EVB1006 Schematic with production version.

Contents

Introducing the EVB100x

The EVB1005 is an image capture module for use with the Opal Kelly XEM6010, XEM6110, XEM3050, and XEM3010 FPGA modules. The EVB1006 is a similar capture module for the Shuttle LX1 (XEM6006) or other FMC carrier. Both modules include a Micron MT9P031I12STC 5 mega-pixel color image sensor and necessary power supply circuitry. Designed as evaluation boards for Opal Kelly integration modules, the modules provide an excellent platform for getting accustomed to the FrontPanel SDK in a demanding, real-world application.

Image Sensor

The heart of the EVB100x is a Micron MT9P031I12STC with the following features and specifications:

- 1/2.5-inch, 5-Megapixel CMOS digital image sensor, 12-bit ADC resolution
- Active pixel array of 2592H x 1944V
- RGB Bayer color filter array
- Snapshot and Electronic rolling shutter
- 96 Mp/s readout rate
- Full resolution frame rate to 14 fps. VGA frame rate to 53 fps.

Mechanical Information

The EVB1005 is designed to mate directly with the expansion bus on the XEM6010 and similar devices. The dimensions of the board are identical to those of the XEM6010 so that the mating is natural and the combination can easily be handled. The EVB1006 is a standard FMC device and is compatible with our Shuttle LX1 and most other FMC carriers.

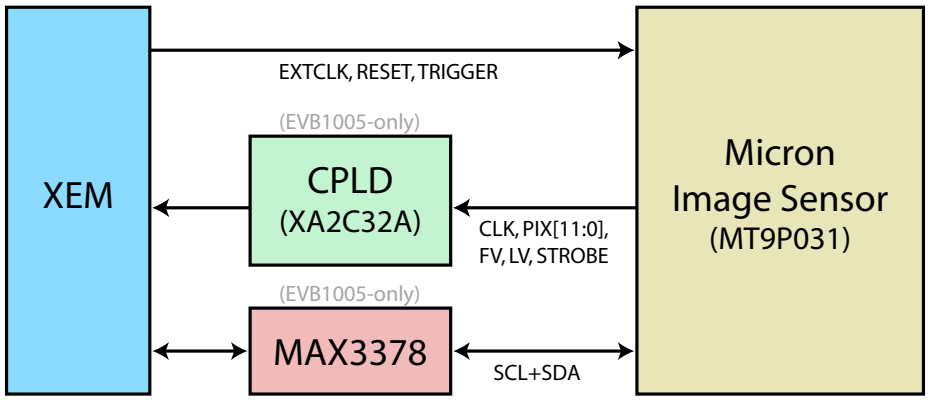
Mechanical drawings of the two modules are at the end of this document.

Lens and Lens Holder (included with EVB100x)

A high-quality glass lens, plastic lens holder, and mounting hardware are included with the EVB1005. The part and supplier information is listed below. Sunex also has a higher-quality lens available with better optics, the DSL944.

Part	Description	Supplier
CMT821	Lens Holder	Sunex (www.optics-online.com)
DSL853C-650	Glass Lens	Sunex (www.optics-online.com)
92005A006	Screw M1.6, 8mm, pan-head	McMaster-Carr
90591A109	Hex nut, M1.6, 0.35mm	McMaster-Carr

Functional Block Diagram



Power Supply

The EVB100x provides +2.8-v and +1.8-v using linear regulators from the +VDC power supply.

+2.8-v is used to power the image sensor's internal analog circuitry as well as the I/O components. The +1.8-v is used to power the image sensor's internal digital circuitry and the CPLD.

XEM6110

When attached to the XEM6110 (which does not have a power connector), the +VDC supply is provided to the EVB1005 through the barrel connector. It must be within the range +4.5-v and +5.5-v. The center conductor is power. The ring conductor is ground.

+VDC is then provided through the expansion connector to the XEM6110.

XEM6010, XEM3050, and XEM3010

When attached to the XEM6010, XEM3050, or XEM3010, the +VDC supply is provided to the EVB1005 through the expansion connector. The external power supply may be connected to either the EVB1005 or the XEM, but only one supply should be attached at a time.

XEM6006

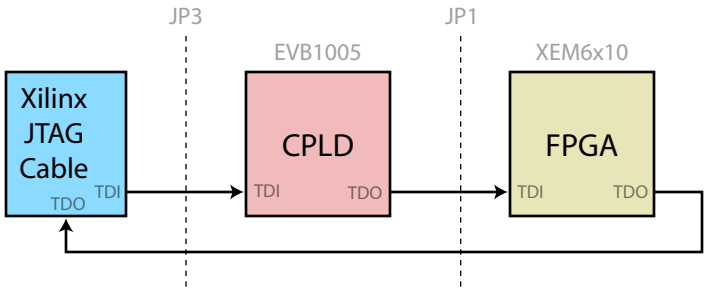
The EVB1006 derives power from the FMC connector on the XEM6006. A power connector is not available on the EVB1006.

JTAG Chain

The EVB1005 has a 14-pin connector (JP3) which is compatible with the Xilinx Platform USB JTAG connector to allow JTAG programming of the CPLD as well as JTAG communication with the FPGA for ChipScope.

XEM6010 and XEM6110

The JTAG chain when used with the XEM6010 and XEM6110 is shown in the diagram below. The Xilinx JTAG adapter is attached to the EVB1005 and both the CPLD on the EVB1005 and the FPGA are part of the chain.



XEM3010 and XEM3050

The JTAG chain is not configured to be used with the XEM3010 or XEM3050.

Electrical Interfaces (EVB1005)

The EVB1005 was designed to operate with the default 3.3-volt I/O on the XEM6010 and XEM6110 devices. Although both devices allow a daughterboard to set the VCCIO on each bank, this requires removal of a ferrite bead on the FPGA module. To avoid this modification, voltage level translation is used instead.

The Micron image sensor digital I/O voltage can also be set to 1.8-v or 2.8-v, but even at the 2.8-volt level, the Output HIGH voltage is below the Input HIGH threshold of the Spartan-6 FPGA.

I^2C Interface

The image sensor uses an I^2C interface for reading and writing several control registers that command the operation of the sensor, set acquisition gains and offsets, and determine how pixel readout is performed.

The HDL design will require a simple I^2C controller to communicate with this interface. A Maxim MAX3378EEUD+ is installed to perform the bidirectional level translation require for I^2C interfaces.

Pixel Interface

The Micron sensor is set to use 2.8-volts for VDDIO. In this configuration, the unidirectional signals from the FPGA to the image sensor (EXTCLK, RESET, and TRIGGER) may be sent directly without level translation.

The unidirectional signals from the image sensor to the FPGA, however, go through a Xilinx CoolRunner II CPLD for level translation. The bank facing the image sensor is set to +2.8-v. The bank facing the FPGA is set to +VCCO, the FPGA's bank voltage (+3.3v).

The CPLD programming files are included assets with the EVB100x Developer's Release.

Electrical Interfaces (EVB1006)

The EVB1006 is an FMC device and contains an EEPROM with IPMI configuration data that configures the FMC carrier for the proper interface voltages. Due to this electrical definition of the interface voltage, no jumpers or ferrite beads need to be manipulated.

I^2C Interface

The image sensor uses an I^2C interface for reading and writing several control registers that command the operation of the sensor, set acquisition gains and offsets, and determine how pixel readout is performed.

The HDL design will require a simple I^2C controller to communicate with this interface. A Maxim MAX3378EEUD+ is installed to perform the bidirectional level translation required for I^2C interfaces.

Pixel Interface

The Micron sensor is set to use 2.8-volts for VDDIO. The FMC interface is configured to a 2.5-v I/O voltage compatible with the Spartan-6 FPGA. In this configuration, the unidirectional signals from the FPGA to the image sensor (EXTCLK, RESET, and TRIGGER) may be sent directly without level translation and will satisfy the sensor's input thresholds. Similarly, the image sensor outputs may be sent directly to the FPGA without level translation. The higher output voltage is below the recommended operating input voltage for the Spartan-6 and will not cause damage.

Sensor to FPGA Pin Mapping

The table below lists the pin mapping from the sensor to the FPGA on the XEM6010, XEM6110, XEM3010, XEM3050, and XEM6006 (EVB1006).

Sensor	Direction	XEM6010 / XEM6110	XEM3010	XEM3050	XEM6006
EXTCLK	O	A11	F9	B13	C10
RESET	O	A15	D1	H3	E7
TRIGGER	O	C15	E2	H4	E8
SCLK	O	A13	E1	J2	A14
SDATA	I/O	C13	F2	J3	F10
STROBE	I	A7	H3	L1	D11
PIXCLK	I	C11	E9	A13	E10
LV	I	A6	G5	M1	D12
FV	I	C7	F5	L2	B14
PIX0	I	A18	B1	E4	A9
PIX1	I	A17	C1	E1	B8
PIX2	I	C17	D2	E2	C9
PIX3	I	B18	C3	E3	A11
PIX4	I	A16	C2	D1	A10
PIX5	I	B16	D3	D2	A13
PIX6	I	A14	E4	G1	C11
PIX7	I	B14	E3	G2	B10
PIX8	I	A12	F4	H1	D9
PIX9	I	B12	G4	H2	C13
PIX10	I	A9	G3	K3	F9
PIX11	I	C9	H4	K4	E11

Pin descriptions

`EXTCLK` - Output clock to the image sensor.

`RESET` - Output reset signal to the image sensor. Active low.

`TRIGGER` - Trigger signal to the image sensor.

`SCLK` / `SDATA` - I^2C control interface to the image sensor.

`STROBE` - Input from the image sensor

`PIXCLK` - Input clock from the image sensor. This is sent with data as "source-synchronous" and is the clock that should be used when capturing the incoming sensor data.

`FV` - Image frame-valid signal from the image sensor.

`LV` - Image line-valid signal from the image sensor.

`PIX[11:0]` - 12-bit pixel data from the image sensor.

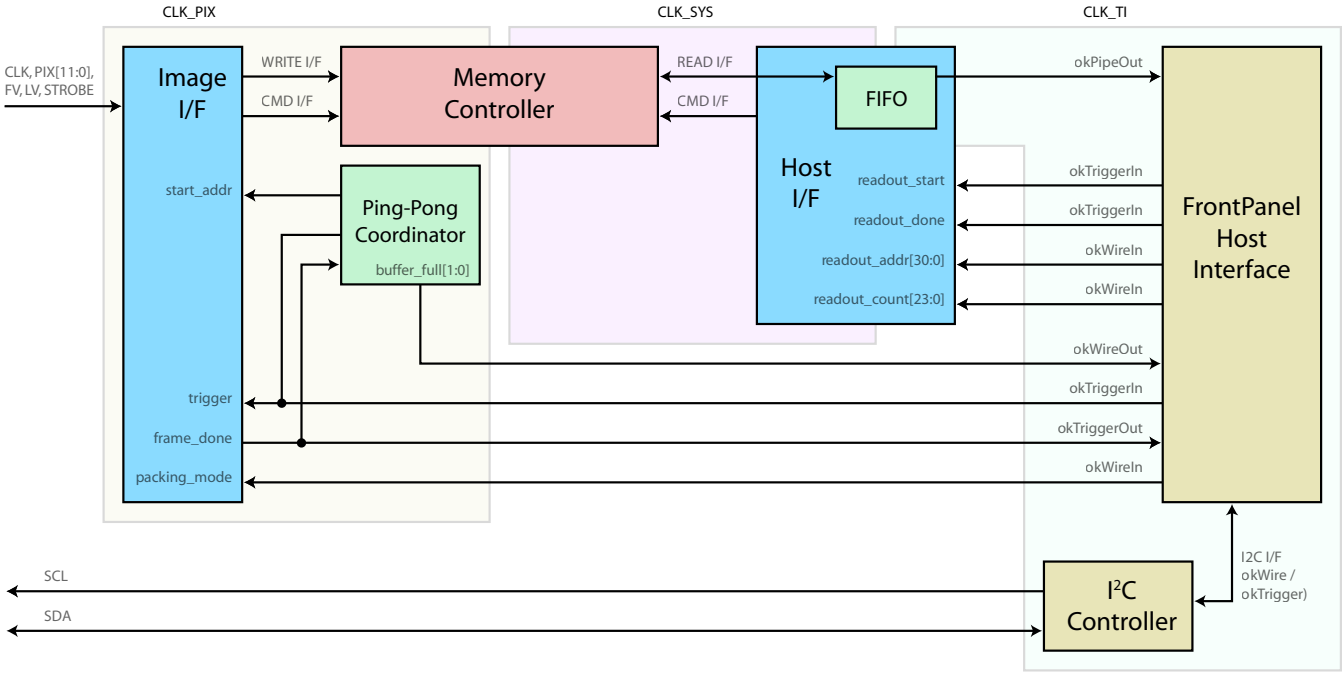
For detailed information on these signals, please refer to the Micron MT9P031 data sheet.

Image Capture Processor Architecture

The EVB100x includes FPGA logic which performs the function of an image capture processor (ICP). This logic is designed with Opal Kelly's FrontPanel HDL so it may communicate with software using the FrontPanel API. This section describes the architecture of the ICP so that you may use it in your own applications. Of course, more advanced processing is possible with your own logic processor.

There are a number of ways to apply the FrontPanel API and HDL to achieve image capture and processing with a given image sensor. This ICP is just one example architecture. Others may be more efficient depending on the specific requirements at hand. The architecture description below is not intended to be complete documentation for the ICP -- just an overview of its design. For further detail, please see the source code.

Top-Level Architecture



Clock Management

At FPGA configuration, the following clocks are available to the design:

`CLK_TI` - This is the target interface clock from the FrontPanel okHost. This clock is 48 MHz for USB 2.0 hosts. It is typically locked with a DCM in the okLibrary source for okHost and is available on one of the FPGA's clock networks.

`CLK_SYS` - This is a 100 MHz clock input on the XEM3010, XEM3050, XEM6010, and XEM6110. On the first three, it is provided by the on-board PLL. On the XEM6110, it is provided by the on-board clock oscillator. This signal is not avaialble on the Shuttle LX1 (XEM6006).

`CLK_PIX` - This is the clock signal received from the image sensor and synchronized by a DCM in the FPGA. `CLK_PIX` is then used for all pixel-clock logic in the design.

After the FPGA is configured, it outputs `PIX_EXTCLK` to the image sensor. The frequency of this signal varies depending on the FPGA module. On the XEM6006, it is 24 MHz. On all other modules, it is 25 MHz. The image sensor has an on-chip PLL which is configured by the camera initialization sequence to output a pixel clock of 96 MHz. This clock is then received by the FPGA as `CLK_PIX`.

The general clock domain organization can be seen in the top-level architecture block diagram above. The clock domains are shaded behind the various blocks represented on that domain.

Reset Logic

Multiple reset signals are delivered to the ICP from the host interface. All resets are asserted high. These are:

`RESET_SYSPLL` - Resets the overall system PLL

`RESET_SENSOR` - Reset the image sensor

RESET _ PIXDCM - Resets the DCM used to synchronize to the pixel clock

RESET _ LOGIC - Reset all state machines and ICP logic

The logic reset (RESET _ LOGIC) is an asynchronous reset to the ICP. From this asynchronous reset, the ICP generates clock-domain-specific resets which assert asynchronously and deassert synchronously with the corresponding clock domain. These are:

RESET _ CLKPIX - Deassertion synchronous to CLK _ PIX

RESET _ CLK0 - Deassertion synchronous to CLK0

RESET _ CLKTI - Deassertion synchronous to CLK _ TI

Image Sensor Interface (ISI)

The image sensor interface (image _ if.v) has a small state machine that observes the incoming image stream (PIX _ FV, PIX _ LV, and PIXDATA) and generates control signals for the memory interface. No FIFO or other buffering is required because the image sensor interface makes use of the buffering already present in the memory interface.

The state machine has a configuration input (PACKING _ MODE) which specifies how incoming image data is packed into memory:

8-BPP - When PACKING _ MODE=0, the eight most-significant bits of each 12-bit pixel are packed into the words written to memory. The least-significant bits are discarded. Since the memory interface is 64-bits wide, a memory word is written every eight pixel clocks.

16-BPP - When PACKING _ MODE=1, each 12-bit pixel is padded with 0's to a 16-bit word and packed into the words written to memory. A memory word is written every four pixel clocks.

At the end of a frame, the memory interface is commanded to write another burst to memory. This effectively flushes the memory interface write buffer in case additional pixels were written to the buffer but it did not completely fill to the burst size.

Host Interface

The host interface (host _ if.v) interfaces the memory interface to the Opal Kelly FrontPanel PipeOut HDL module to transfer image data back to the host. This is a one-way communication and crosses a clock boundary with CLK _ SYS on one side and CLK _ TI on the other. A small block memory FIFO is used to cross this boundary.

The state machine in the host interface sits IDLE until a readout is initiated by the host. Once initiated, the host interface captures the readout start address (READOUT _ ADDR[29:0]) and the readout byte count (READOUT _ COUNT[23:0]). It then successively issues memory read commands as long as there is space available in the FIFO until the full readout count has been read. The host will be reading from this FIFO.

Because the memory interface can easily keep pace with the host reads, no additional throttling is required. The FIFO is reset at the beginning of each readout to make sure it is empty prior to a readout.

Memory Interface

The memory interface is generated by the Xilinx Core Generator / Memory Interface Generator (MIG). For the Spartan-6 FPGA, this utilizes the hard-core memory controller block and consumes very little additional fabric resources. The MIG is configurable to support a number of different interface ports. For our purposes, we only use two ports: one for writing to the memory from the image sensor interface and one for reading from the memory to the host interface.

MIG ports each have their own read and write clock domains. Operations are performed by writing commands to a command FIFO on each port. Complete documentation on the operation of the memory interface is available from Xilinx.

Ping-Pong Coordinator (PPC)

The Ping-Pong Coordinator is a simple state machine that is used when the ICP is placed in ping-pong mode and performs continuous image capture. In this mode, memory is segmented into two separate buffers so that one may be read out by the host while the other is being written to by the image sensor interface. The PPC becomes the commanding process for the image sensor interface and dictates when image capture should proceed and which buffer should be utilized.

The PPC maintains two flags (`BUFFER _ FULL[1:0]`) that are set when a buffer is writen by the ISI. When the host completes readout of one of these buffers, the corresponding flag is then cleared by the PPC and may be written again.

I²C Controller

In order to communicate with the image sensor's register interface and configure the sensor, a simple I²C controller is provided. This controller is commanded through the FrontPanel host interface using only Wires and Triggers.

Source code for the I²C controller is not provided.

Aptina Sensor Notes

At the time this documentation was written, the MT9P031 datasheet was published at Ref. F dated May 2011. There are a few notes and errata that should be mentioned. Aptina application support was very helpful in resolving these issues.

Data Valid Window

Figure 29 of the datasheet represents the I/O Timing Diagram. According to Aptina, the correct way to interpret Tpd from this diagram and the timing characteristics is that data is valid as early as 0.8ns before the rising edge of PIXCLK. So the minimum time should probably be written as -0.8ns.

This modified timing is reflected in the constraints files provided in the Developer's Release.

Line Valid and Frame Valid at High Clock Rates

Pixel data, LV, and FV are launched from the sensor on the rising edge of PIXCLK and Aptina suggests capturing these on the falling edge. However, the maximum time from PIXCLK (rising edge) to LV and FV becoming valid is 5.9ns. At 96 MHz pixel rates, the clock period is 10.4ns. Therefore, these transitions would occur after the falling edge. This causes synchronization problems.

Aptina has released Technical Note TN-09-148 that addresses this issue. Their recommendation is to run the image sensor at 96 MHz, but enable an undocumented internal FIFO to allow pixel data to be read out during the horizontal blanking period. This makes the internal sensor clock run at 96 MHz, but the PIXCLK and data outputs are run at 72 MHz. Our software configures the image sensor according to this technical note.

Software Host Interface

Asynchronous Ports (Wire In / Wire Out)

The following are asynchronous inputs and outputs to the ICP. They are provided via WireIn / WireOut HDL modules and are re-timed to individual state machine clock domains as necessary.

Register	Signal	Description
WI[00|0]	RESET_SYSPLL	System PLL reset (active high)
WI[00|1]	PIX_RESET	Image sensor reset (active high)
WI[00|2]	RESET_PIXDCM	PIXCLK DCM reset (active high)
WI[00|3]	RESET_ASYNC	Asynchronous logic reset
WI[00|4]	-	Capture mode (0:Trigger, 1:Ping-pong)
WI[00|5]	-	Image packing (0:8-bit, 1:16-bit)
WI[01|7:0]	I2C_MEMDIN	I²C controller input data
WI[02|15:0]	READOUT_COUNT_L	Image readout count (least-significant word)
WI[03|15:0]	READOUT_COUNT_H	Image readout count (most-significant word)
WI[04|15:0]	READOUT_ADDR_L	Image readout address (least-significant word)
WI[05|15:0]	READOUT_ADDR_H	Image readout address (most-significant word)
WI[20|10:0]	PIPE_OUT_RD_COUNT	Image readout count
WI[22|7:0]	I2C_MEMDOUT	I²C controller output data
WI[23|7:0]	SKIPPED_COUNT	Skipped image count (ping-pong mode only)
WI[23|9:8]	BUFFER_FULL	Buffer status for ping-pong buffers

Synchronous Trigger Ports (Trigger In / Trigger Out)

The following are synchronous trigger ports that are established on specific clock domains within the ICP. They perform "event-based" communication with the PC over FrontPanel TriggerIn / TriggerOut HDL endpoints.

Register	Clock Domain	Description
TI[40|0]	CLK_PIX	Triggers a single image capture
TI[40|1]	CLK_TI	Signals the beginning of image readout
TI[40|2]	CLK_SYS	Signals completion of image readout from buffer A
TI[40|3]	CLK_SYS	Signals completion of image readout from buffer B
TI[42|0]	CLK_TI	Initiates an I²C operation
TI[42|1]	CLK_TI	I²C memory pointer reset
TI[42|2]	CLK_TI	I²C memory write
TI[42|3]	CLK_TI	I²C memory read
TO[60|0]	CLK_PIX	Frame available
TO[61|0]	CLK_TI	I²C operation done

Synchronous Data Transfer Port

PipeOut port 0xA0 is the only synchronous data transfer port utilized. It is used to read out image data after an acquisition.

Host Software

okCCamera C++ Class

We have wrapped the basic functionality of the camera into a lightweight C++ class which should serves as the only interface to the device. The `okCCamera` object should handle all communication with the device via the FrontPanel API. Both the `okSnap` command-line application and the `okCamera` GUI application make use of this common class to communicate with the camera.

Image Capture Processor Initialization

Initializing the camera for operation involves the following sequence:

1. Create an instance of okCFrontPanel
2. Open a device
3. Configure the PLL (for boards with an on-board PLL)
4. Configure the FPGA with the appropriate bitfile
5. Perform logic reset as required

These five steps are performed by the `Initialize()` method. Until this method is called successfully, the camera remains in an uninitialized state and cannot be used.

Image Capture and Data Transfer

A single image capture cycle is initiated and read using the method `SingleCapture()`. This method performs the following sequence:

1. Sets the image length to the READOUT_COUNT register.
2. Sets `READOUT _ ADDRESS` to `0x00000000`.

3. Triggers the capture
4. Waits until the frame done trigger
5. Triggers readout start
6. Reads the complete image data
7. Triggers readout done

okSnap Command Line Application

okSnap is setup to be a very simple command-line application that connects to the camera, configures the image sensor and image capture processor, then retrieves a single image with some default settings. It is not intended to be a full-featured application but highlights the primary interface capabilities required to talk to the camera. Using okSnap as a starting point, it is possible to expand to a more capable command-line interface.

Command Line Usage

okSnap takes two arguments. The first is the image sensor acquisition mode taken from the MT9P031 datasheet. "0" means normal image acquisition. "9" is a diagonal gradient. The gradient and other test modes are useful for ICP testing.

```
> okSnap 0 image.bin
```

The resulting output file (image.bin) will be the full image acquisition output from the sensor.

okCamera GUI Application

wxWidgets

The okCamera application is built using the wxWidgets cross-platform C++ GUI class library. More information may be found at the following site:

http://www.wxwidgets.org

Threaded FrontPanel API Communication

To create a well-behaved GUI application, it is important that any long-running processing be performed in a separate thread from the application's GUI thread. In the case of okCamera, certain operations such as image readout can take a while. Therefore, we have moved all camera communication (and, in effect, all communication over the FrontPanel API) into a separate thread. This thread is abstracted in the okCThreadCamera class and represents a disciplined method of device communication.

FreeMat 3rd-Party Software Sample

FreeMat is an open-source development environment, similar to MATLAB, designed for rapid engineering, scientific prototyping, and data processing. It has extensive script capabilities for processing vector and matrix data (such as images) and also has the capability to interface with external (3rd-party) interfaces such as the Opal Kelly FrontPanel API. Together, these features make it an attractive platform for a sample interface to the EVB100x.

More information about FreeMat may be found here:

http://freemat.sourceforge.net/

FrontPanel API Support

Basic FrontPanel API support has been imported into FreeMat using the `import` command which creates FreeMat commands from DLL entry-points. In some cases, the preferred functional representation of the API command is not directly available with this method. For example, `GetDeviceSerialNumber` returns a string. FreeMat needs to be called with an empty string of the proper length in order to use this direct import.

The preferred functionality could be accomplished with a separately-defined FreeMat command that wraps the imported DLL command. For simplicity, we simply work around these deficiencies in the example code. To a great extent, a one-to-one mapping of the API commands has been accomplished. The exceptions are simple to deal with and understand.

Interfacing to the Image Capture Processor

Initial configuration and setup for the camera is easy using the imported FrontPanel methods:

```
dev = okFPConstruct;
okFPOpenBySerialX(dev, 0);
okFPLoadDefaultPLLConfiguration(dev);
okFPConfigureFPGA(dev, 'evb1005-xem6010-lx45.bit');

% Reset
okCameraFullReset(dev);
```

The `okCameraFullReset` function is defined using API methods as well:

```
function okCameraFullReset(dev)
  okFPSetWireInValue(dev, hex2dec('00'), hex2dec('000f'), hex2dec('000f'));
  okFPUpdateWireIns(dev);
  okFPSetWireInValue(dev, hex2dec('00'), hex2dec('0000'), hex2dec('0001'));
  okFPUpdateWireIns(dev);
  ...

  % REG_PLL_CONTROL = 0x0051
  okCameraI2CWrite(dev, hex2dec('10'), hex2dec('0051'));

  % For the XEM6010 / EVB1005
  N=5; M=72; P1=3;
  % REG_PLL_CONFIG1 = ((N-1)<<0) | (M<<8)
  okCameraI2CWrite(dev, hex2dec('11'), N-1 + M*256);
  % REG_PLL_CONFIG2 = (P1-1)<<0
  okCameraI2CWrite(dev, hex2dec('12'), P1-1);
  ...

  okCameraLogicReset(dev);
```

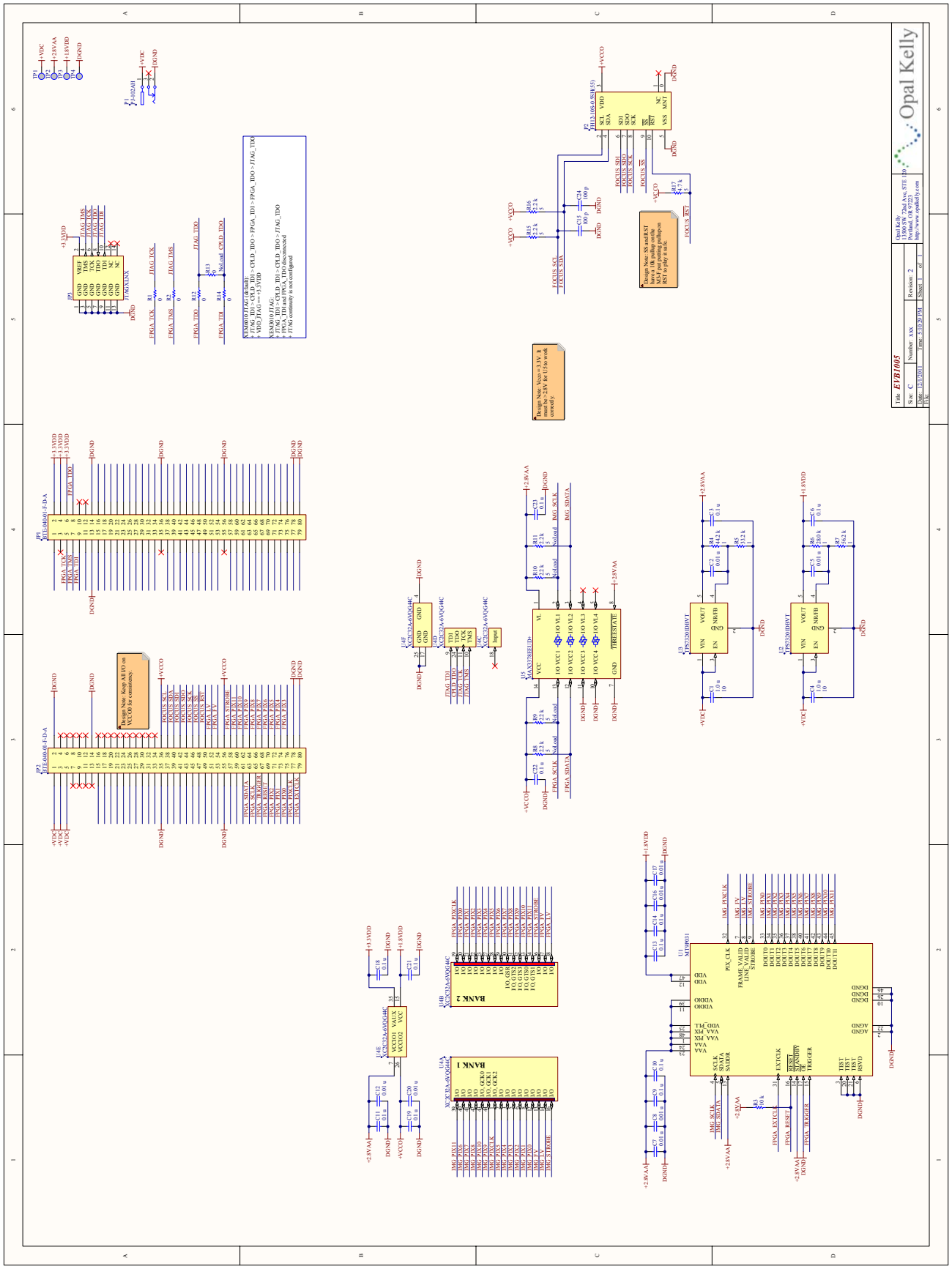
Data Manipulation and Image Display

The captured image data is received as a 5 megabyte `uint8` vector which is packed one byte per pixel. To format for display using FreeMat's `image` command, the vector is manipulated into a 1296 x 944 x 3 array where each plane of the array represents the red, green, and blue pixels from the Bayer pattern. For simplicity, the pattern is decimated to fit the array dimension. Much more could be done to perform higher-quality demosaicing of the image.

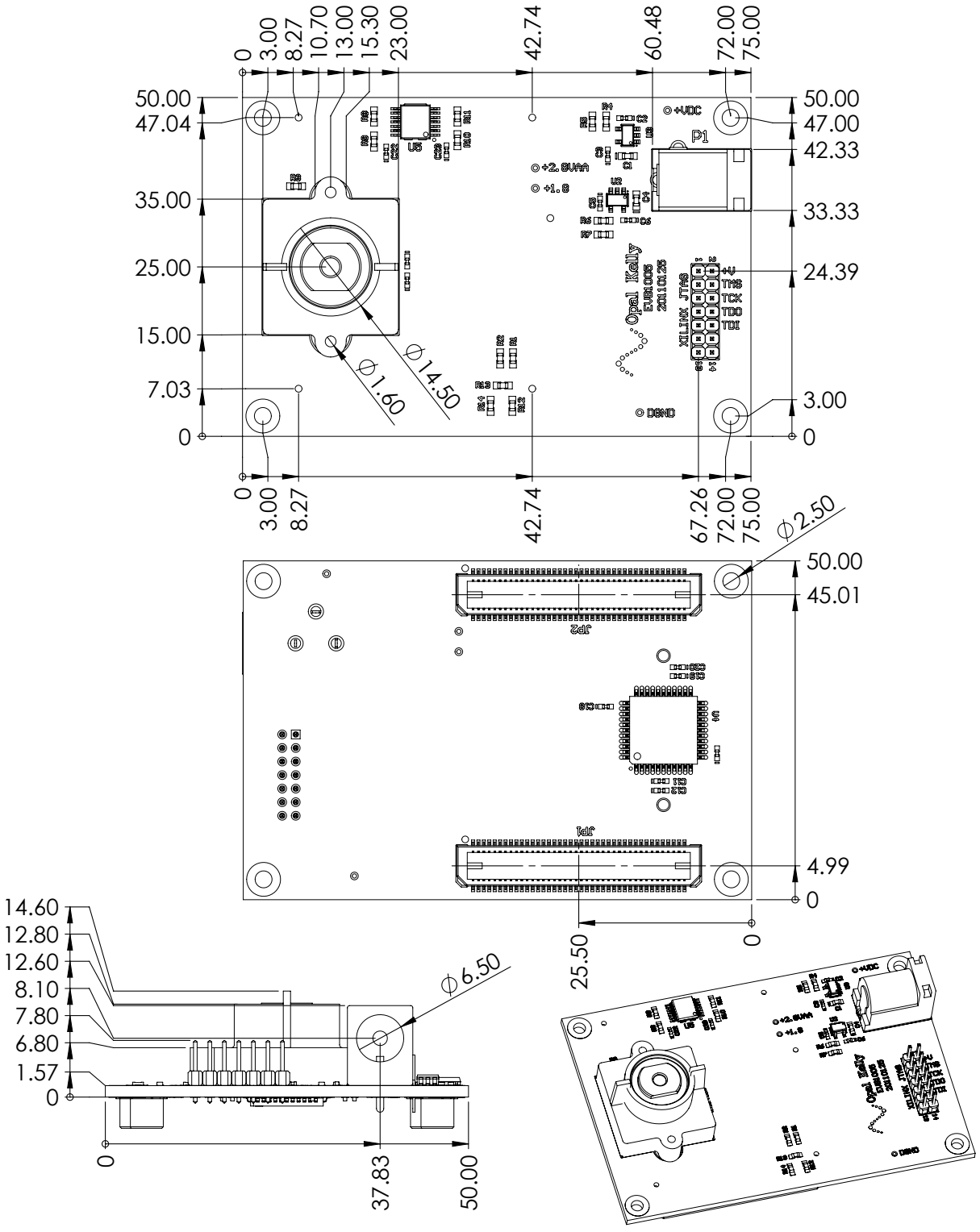
```
% Read the captured image
okFPReadFromPipeOut(dev, hex2dec('a0'), imglen, imgdata);

% Format for image() display
tmp = double(reshape(img(1:x*y), x, y)).';
imgv = zeros(y/2, x/2, 3);
imgv(:,:,1) = tmp(1:2:y, 2:2:x)/256; % RED
imgv(:,:,2) = tmp(1:2:y, 1:2:x)/256; % GREEN
imgv(:,:,3) = tmp(2:2:y, 1:2:x)/256; % BLUE
image(img);
```

EVB1005 Schematic

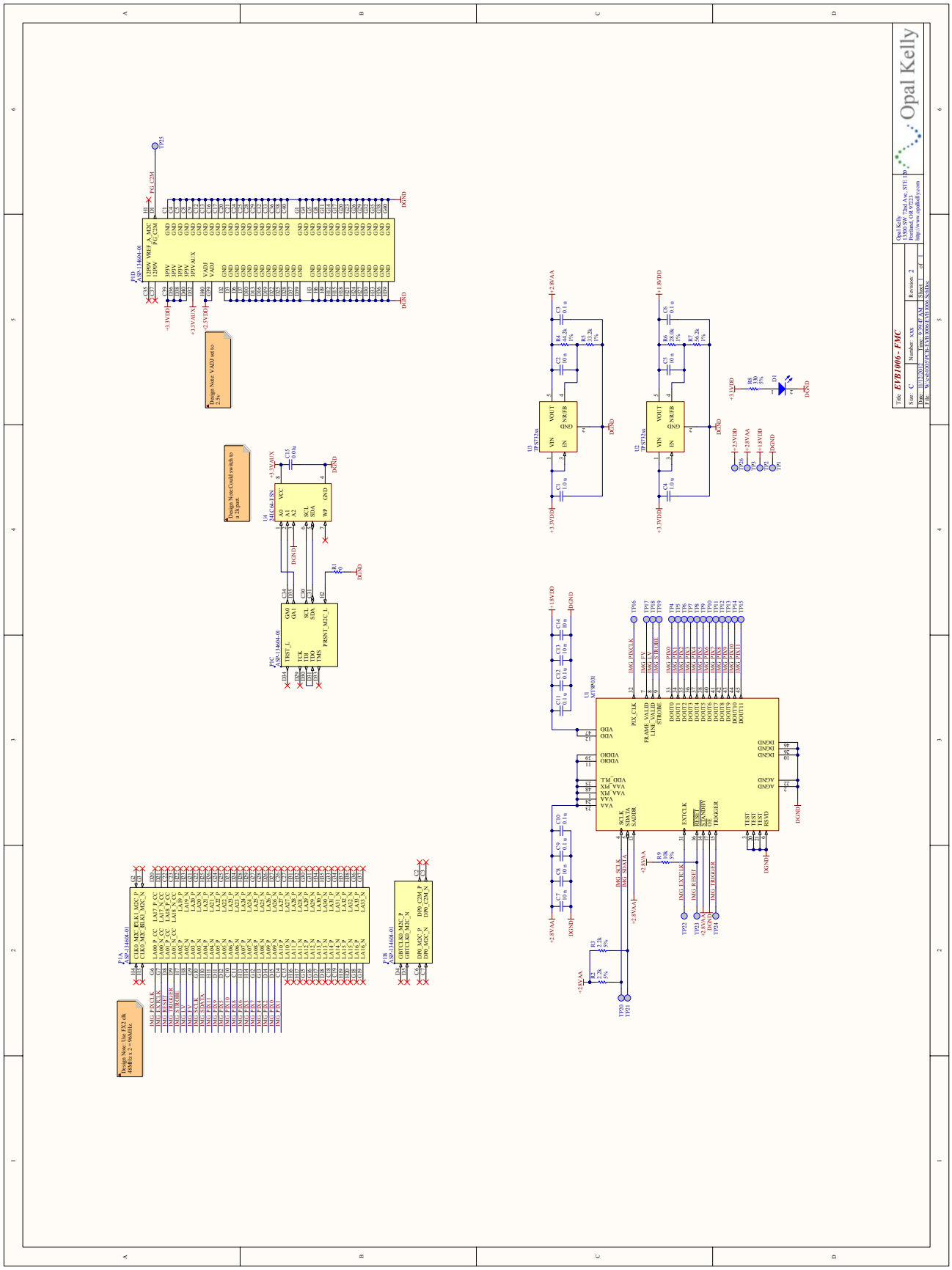


EVB1005 Mechanical Drawing

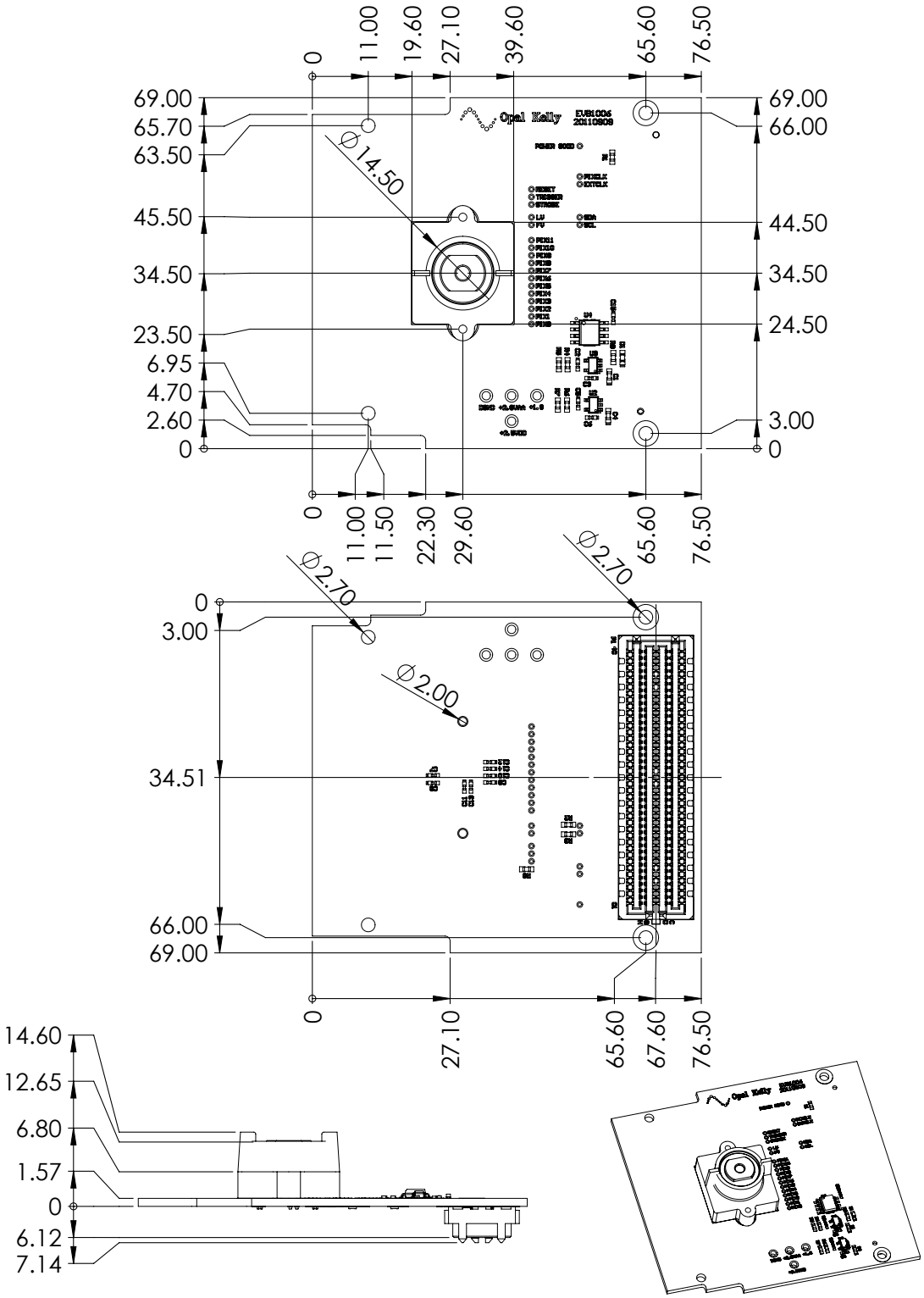


All dimensions in mm

EVB1006 Schematic



EVB1006 Mechanical Drawing



All dimensions in mm