

Zilog Ethernet Camera

Matthew Condit

CS 339

5/5/2005

Design Overview

The purpose of this project is to create a CMOS camera module interface over Ethernet. An ez80 Acclaim! Zilog microcontroller is used to read information from a camera module and then send the raw picture data to a PC. The microcontroller is also responsible for accepting camera configuration commands and then setting the camera register to the appropriate values using the I²C bus. A GUI on the PC facilitates the display of the captured pictures as well as camera configuration settings.

The proposed requirements that the project must meet are:

- Interact with PC over Ethernet interface.
- Support TCP/IP
- Be able to grab a single frame from camera module.
- PC must be able to modify camera settings.
- Microcontroller will do basic formatting of image data before sending back to PC

Hardware Modules

The project design consists of two main hardware modules: the ez80 Acclaim! Modular Development Board and the C3088 CMOS Camera.

eZ80F91 Modular Development Kit

The EZ80F91 modular development kit allows the user to design and evaluate projects using the EZ80F91 microcontroller. The kit contains an EZ80F91 module, which contains the EZ80F91 device running at 50 MHz, with 256 Kbytes of Flash memory and 8 Kbytes of internal SRAM. Off chip, the processor has access to 128 Kbytes of external SRAM. This module also contains an Ethernet Media Access Controller (EMAC) and Ethernet port. In the Zilog literature, this module is referred to as the mini-module because it plugs into a larger modular development adaptor board through 56 pin mini-module connectors.

The adapter board contains an RS-232 connector circuit for UART0 of the EZ80F91. Furthermore, it contains JTAG and Zilog's proprietary ZDI for debugging and programming. This board is actually made to plug into yet a larger board through two 60-pin headers on the bottom of the adapter board. This project does not need the larger board, so the header pins are used to directly interface with the microcontroller instead. There is also a footprint on the adapter board for a GPRS modem. Finally, the adapter board provides voltage regulation from 9V to 5V and 3.3V respectively.

C3088 CMOS Camera Module

The C3088 is a 1/4" color camera module with digital output. It uses the Omnivision OV6620 image sensor. Together, the module provides a continuous stream of 8 or 16 bit-wide image data through a digital video port. All the camera functions, including gain, brightness, white balance, image size, etc, are configurable over the I²C bus.

The C3088 supports multiple data formats using different channels (RGB, UVY). The format used in this design is the simplest one possible: YUV 4:2:2. For the project, only the Y channel (intensity) is used so that each pixel is represented by 8-bits and corresponds to a grayscale value. The UV channel (chrominance) represents the color aspects of the picture, but this project leaves them unconnected. This information is sent continuously from the camera and the VSYNC, HREF, and PCLK signals are used to synchronize the picture. The assertion of the VSYNC signal indicates the start of a new frame. The HREF signal is used for horizontal synchronization and indicates when data is valid. The PCLK signal is used to clock out picture data. The period of PCLK can be configured over the I²C bus. The default period of the camera is 17.76 MHz.

Figure 1 shows the timing signals used in the C3088.

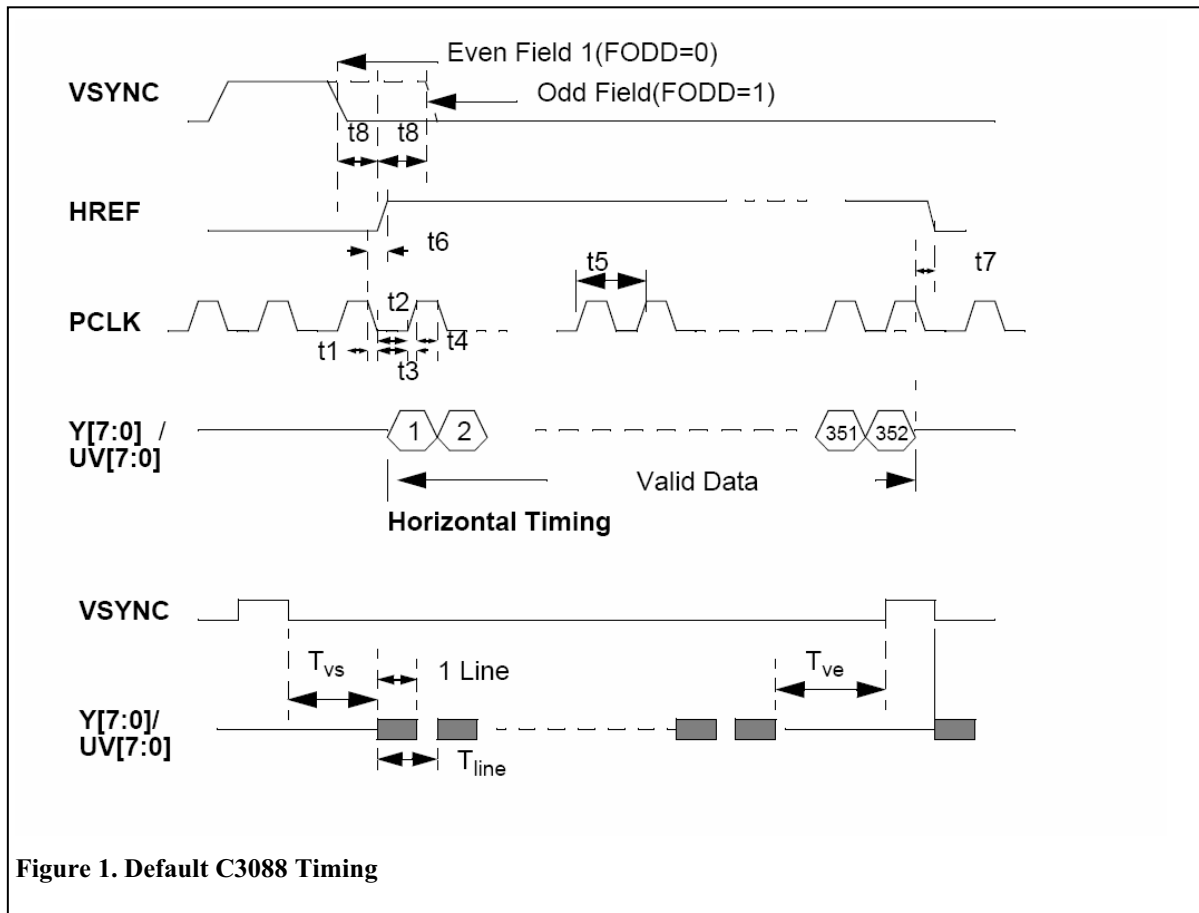


Figure 1. Default C3088 Timing

Here, the default C3088 timing is shown. The transition of VSYNC from high to low, signals the start of a new frame. A frame consists of a number of rows. In this default case, there are 288 rows of 352 pixels each. After the start of a frame, HREF will become asserted. While HREF is high, on the rising edge of each PCLK, picture data can be sampled from the Y and UV buses. In the default case, there should be 352 PCLK cycles while HREF is asserted. Once a complete row has been sent, HREF is set to low. Once the start of the next row begins, HREF is asserted once again. In the default case, this process takes place 288 times.

Table 1 shows the pinout of the C3088.

Pins	Name	Description	Design Comment
1-8	Y0-Y7	Digital Y output	
9	PWDN	Power Down	Grounded
10	RST	Reset	Grounded
11	SDA	I ² C Serial Data	
12	FODD	Odd Field Flag	Not Used
13	SCL	I ² C Serial Clock	
14	HREF	Horizontal Reference	
15, 17, 21	AGND	Analog Ground	
16	VSYNC	Vertical Sync	
18	PCLK	Pixel Clock	
19	EXCLK	External Clock	Not Used
20, 22	VCC	5V Power Supply	
23-30	UV0-UV7	Digital UV output	Not Used
31	GND	Common Ground	Not Used
32	VTO	Video Analog Output	Not used

The main signals used in the project at the Y bus, I²C bus signals, VSYNC, HREF, PCLK, VCC, and AGND.

Table 2. Pin Summary

Camera Pins	Pin Description	Board Pin Number/Header	Name
1-8	Y0-Y7	JP3: 7-0	PA0-PA7
11	SDA	JP3: 47	SDA
13	SCL	JP3: 45	SCL
14	HREF	JP3: 11	PB7
16	VSYNC	JP3: 13	PB5
18	PCLK	JP3: 15	PB3
9	PWDN	JP3: 46	GND
10	RST	JP3: 46	GND
15	AGND	JP3: 46	GND
17	AGND	JP3: 46	GND
21	AGND	JP3: 46	GND
22,28	VCC	J3: 4	VCC_5V

Parts List

Table 3 contains the part numbers used for the project as well as cost.

Table 3. Parts List

Description	Part Number	Cost
EZ80F91 Dev Board	EZ80F910100KIT	\$99
C3088 CMOS Camera	C3088-4928IR	\$55
15 pin flex cable x2	A9BAG-1508F-ND	\$15.88
16 pin flex cable x2	A9BAG-1608F-ND	\$16.64

Software

The software design of the project is divided into two distinct halves: the microcontroller camera interface and the client-side GUI. The majority of the work took place in the microcontroller firmware, so it warrants a detailed examination.

File Listing:

Application Files

- main.c – main entry point of the program. Initializes UART and camera. Launches main application thread
- cam.c, cam.h – camera specific functions
- i2c_cam.c, i2c_cam.h - I²C functions used to read and write bytes

- network_transfer.c, network_transfer.h – main thread of the application. Handles requests from PC over a socket as well as initiates frame grabs.

RZK Files

- config.c – memory map for processor
- ez80eval.c – devboard specific functions
- ez80HWConf.c – device driver configuration file
- F91PhyInit.c - handles initialization of Ethernet port
- RZKStartup.s – assembly code for the startup of the OS
- tty_conf.c – defines number of TTY devices to support
- XTLInit.c – launches the main.c file
- XTLSysgen.c – sets initial defines for memory
- XTLZDevice.c – defines the device drivers loaded into OS
- ZTPConfig.c – sets up IP addressing, DHCP requests, etc.

Firmware

RZK/ZTP Framework

The eZ80F91 development board comes with its own real time operating system (RTOS) called RZK. Specifically, RZK is used to support Zilog’s own TCP/IP stack called ZTP. For this project, RZK version 1.1.2 and ZTP version 1.4.2 are used. The OS provides facilities for thread creation and management, memory allocation and partitioning, built in support for serial communication via a UART, among many other things. The TCP/IP stack has many built in features such as telnet, ftp, http, SSL, and standard sockets. Within this framework of design, the project was created.

Compiling the project

Note: To properly compile the project, ZDS II for the ez80 Acclaim! version 4.8.0 must be used. Newer versions of the IDE do not support RZK version 1.1.2 and ZTP 1.4.2.

In order to successfully compile the project, the necessary project options had to be determined. The basis for the settings for this project come from the ZTPDemo_F91_Mini sample project provided with ZTP. Please see Appendix A for project compilation settings.

Serial Communication

Simple debugging was accomplished using the processor’s UART and a terminal program running on a PC. Within RZK as a default, UART0 is selected for serial communication. After using RZK’s open command on SERIAL0, UART0 is properly configured. After this, standard printf() statements can be used to convey debug information. This was the exact scheme used in the project to help diagnose problems.

I²C Communication

RZK has built-in support for I²C, however, their libraries never worked properly in the design. Instead, a set of I²C functions was created for the project that used the processor's I²C resources accessed by using the I2C_CTL, I2C_CCR, and I2C_DR registers. These functions include I2C_Init(), I2C_Write(), and I2C_Read() among some others. These functions can be found in the i2c_cam.c and i2c_cam.h files.

The functions in these files are application specific in that they automatically place the proper device address onto the bus. A write cycle begins by issuing a start condition. The next step is to place the write address of the camera (0xC0) onto the bus. Next, the register to write to is placed. Following this, the byte to write is issued. A read cycle is slightly more complex. The read will always return the contents of the last write. To select a different register, a dummy write must be enacted where only the register is selected and nothing written. After the dummy write, the read address of the camera is placed on the bus (0xC1). After this, the camera sends two bytes back. The first is the register address, and the second is the actual value.

One major hurdle that had to be overcome was to reliably get the I²C functions to work properly. As it turns out, after the camera has been reset, the next I²C instruction must be repeated twice for the value to be properly written. To account for this, the wrapper functions found in the cam.c file always read the value back after a write. If the value was not correct, it will write it again. These functions will be further elaborated later.

Sockets Programming

One advantage to using ZTP 1.4.2 is its support for standard sockets programming conventions that have been used in C programs for years. After the main application thread is launched, a socket is opened and binded. After this, the thread listens for an incoming connection on the appropriate port. Once the connection is accepted, the thread waits for data to be sent. It is here then that it implements the camera protocol defined by the design.

Protocol Implementation

The project supports a number of commands to be sent from PC to the Zilog chip. These commands are detailed in Table 3.

Command Name	Hex Value	Description
GRAB_FRAME	0xbb	Initiates the transfer of one frame. After sending the packet, an ACK will be sent back. Following the ACK, the raw data bytes will be sent. Currently only 1 picture size is supported (176x144). Future version will implement the 352x288 picture size.
TERMINATE_CONNECTION	0xcc	Terminates the connection. This will inform the camera interface to shut down the connection socket and accept new connections.
ADJUST_BRIGHTNESS	0x01	Changes the brightness setting of the camera. The camera default is 0x80. 0xFF is the highest setting, 0x00 is the smallest.
ADJUST_CONTRAST	0x02	Changes the contrast setting of the camera. The camera default is 0x48. 0xFF is the highest setting, 0x00 is the smallest.
ADJUST_SHARPNESS	0x03	Changes the sharpness setting of the camera. The default setting is 0xC6. The upper 4 bits set the threshold of sharpness. The lower 4 bits provide the sharpness control.
ADJUST_CLOCK_SPEED	0x04	Changes the clock speed of the camera. The default camera setting is 0x00. Currently, the fastest the picture data can be sampled is when the clock speed is set as 0x14. The slowest clock rate is 0x3f.
ADJUST_SATURATION	0x05	Changes the saturation settings of the camera. The default value is 0x80. 0xFF is the highest setting, 0x00 is the smallest.
RESET_CAMERA	0x06	Resets the camera to its default state
MIRROR_IMAGE	0x07	Mirrors the image.
UNMIRROR_IMAGE	0x08	Unmirrors the image.
BIG_PICTURE_SIZE	0x09	Configures the camera to capture pictures with a resolution of 352x288. This is currently not supported.
SMALL_PICTURE_SIZE	0x0A	Configures the camera to capture pictures with a resolution of 176x144.
BLACK_WHITE_MODE	0x0B	Sets the camera into black and white only mode. This is currently the only option supported.
COLOR_MODE	0x0C	Sets the camera into color mode. Currently, this mode is not supported.

After each command is parsed, the appropriate wrapper function from the cam.c file is called. In general, these commands set register values in the camera over the I²C bus. There are a few that do not, however. TERMINATE_CONNECTION closes the session and deletes the current socket. The program begins to listen for a new connection at this point. GRAB_FRAME causes the camera to obtain picture data and transfer it over the Ethernet connection. This process is further elaborated in the following section.

Obtaining Picture Data

The core function behind the entire project is the `grab_frame()` function found in `cam.c`. This is the function that actually reads raw picture data from the camera. To do this, the function takes in a pointer to a buffer in memory large enough to hold an entire picture. Currently, this means that the buffer size is 176x144 in length for a total of 25344 bytes. This is roughly 1/5 of the available SRAM. In its current configuration, RZK will not allocate a buffer much bigger than this. This is one reason why the larger resolution of 352x288 is not supported in the project.

The function begins by waiting for the VSYNC signal to go high and then for it to go low. This signals the start of a new frame. After this, it begins a loop that will iterate 144 times – one for each row. It first waits for HREF to assert. It then begins another loop that continues looping while HREF is high. Once in this loop, it waits for the rising edge of PCLK. Once high, the program samples the 8-bits of the Y channel and increments the buffer counter. Once HREF drops low, 176 pixels should have been sampled. The loop then waits for HREF to go high again and the process repeats itself. Once the outer loop has iterated the requisite number of times, the function returns and the raw data is sent over the socket connection to the PC.

There were many issues getting this function to work. The first problem that was encountered was the fact RZK operating system kept preempting the main application thread while picture data was being sampled. It took some work determining that this in fact was the problem. This problem was solved by preventing other threads from preempting the application thread whenever `grab_frame()` was called. Also, the thread priority was increased as well as the thread time slice. This ensures that the application will have preferential treatment of the processor.

Another problem involved finding the correct speed at which to set the camera clock in order to read from it fast enough. The camera clock has to be significantly slowed down to around 135 KHz. This is a direct result of the way the VSYNC, HREF, and PCLK signals are sampled. To increase speed, a detailed timing analysis should be conducted. Interrupts should be used to signal the presence of VSYNC and HREF. Assembly language should then be written that will exactly match the number of cycles to read in pixels. At this time, the current code is not agile enough to properly sample data points. Once the processor can more effectively sample data, the clock speed of the camera can be increased.

Initially, the big picture size (352x288) was to be supported. As was stated previously, only about 1/4 of the total amount of memory needed could be allocated. To get around this problem, multiple loops were used to send a quarter of the image at a time. The problem with this approach was the time it needed to do this. 10 total frames had to be read (read the first quarter, then it must wait for the first quarter to pass and grab the next quarter, etc). Because of the delay caused by sending the data, the image quality suffered. Each quarter of the image was clearly visible. Because of this poor performance, the big picture size was disabled in the project.

GUI

The Java GUI is implemented in two parts: Camera.java encapsulates interaction with the camera itself and CameraGUI.java contains the requisite java code to control the GUI. The GUI itself is rather straightforward and won't be discussed further, but the Camera.java class bears looking into further.

Camera.java implements the protocol discussed earlier. Each command is mapped to its own specific method each keeping track of any GUI related information it may contain. For example, the adjust saturation, contrast, brightness, and sharpness scales the value into a percentage that is displayed on the slider bars on the GUI.

The most important method in the class, however, is the getImage() method. This method borrows heavily on the java.awt.image APIs. After all of the pixel information has been read into a byte array, DataBufferByte instance is created from the read-in information. Next, a component sample model must be created that specifies how the pixel information is to be interpreted. In this case, every byte represents one pixel. The image height is 144 rows, and the image width is 176 pixels. After the sample model is created, a component color model must be created; this holds the color information for the picture. In this case, a grayscale model is used. Once the color model is created, a writable raster must be created that takes in the sample model and the pixel buffer as arguments. Finally, an instance of image can be create which uses the writable raster and color model as inputs. This image class can now be displayed in the gui or even used to save the image in a variety of formats (jpeg, bmp, png, etc).

GUI Operation

The GUI performs several simple tasks. First it takes in the IP address of the camera to connect to. This project loads in a default IP of 192.168.0.50 for the camera. Once the IP address has been entered, the user must initiate a connect. Once connected, the options at the bottom of the GUI become accessible.

The Capture Image button initiates a picture transfer and displays it in the GUI. The image itself has been expanded to better fill the GUI window. Saturation, sharpness, contrast, and brightness are all configurable by the slider bars. Unfortunately in the black and white mode setting of the camera, only brightness changes the image; the other three do not produce anything significant. Finally, the Mirror Image check box toggles whether or not the image should be mirrored or not. Figure 3 shows a screenshot of the GUI.



Figure 3. GUI Screenshot

Future Additions and Changes

The results of the project are by no means perfect. There are many parts that can be improved and many features could be easily added without too much difficulty. Some possible ideas are discussed below.

Faster Pixel Capturing

The most important part of the design that needs improving is the pixel-capturing algorithm. By improving this part of the code, an entire picture can be read much quicker which would allow for faster frames per second. To scan an entire picture, it currently takes roughly 188 ms; if the camera clock period could be increased to somewhere around 2.2MHz, an entire frame could be captured in approximately 12 ms.

In order to accomplish this, the pixel data capturing code should be rewritten using hand-coded assembly language. The assembly instructions should be designed keeping in mind the number of cycles to execute and how they relate to the incoming data. For instance, if the camera clock runs at 2 MHz and the processor runs at 50 MHz, every 25 cycles will correspond to one clock cycle. Using a scheme such as this along with external interrupts will allow a tight interfacing between components and summarily increase the speed at which data can be read from the camera.

Another way to increase pixel capture time is to shorten the transmissions lines between the processor and the camera. In the current design, the image data must travel over two ribbon cables connected through a protoboard. By designing a custom PCB where both modules can snap into, propagation delay and clock skew will have less influence on the quality of the data being transferred.

Larger RAM

One of the biggest limitations in the current design is the inability to obtain a large enough block of memory from the operating system. Even though the system has 128Kbytes of SRAM, only one fifth of this can be allocated.

To obtain a larger memory buffer, there are several options that could be explored. The first is simply to increase the amount of memory in the system. The SRAM could be increased from 128Kbyte to 256 Kbyte or 512Kbyte which should easily allow an entire picture with resolution of 352x288 and 16-bit pixels to be stored in RAM. Another option is to optimize the operating system by minimizing the amount of RAM it consumes. Things such as thread stack sizes as well as loaded module would have to be taken into account. This might necessitate a change of operating systems and even processors if memory requirements cannot be met.

Color Pictures

Color pictures could be added relatively easily to the design. The UV channel bus of the C3088 would need to be connected to the microprocessor. The camera supports a variety of picture formats such as YUV 4:2:2 and RGB. A suitable picture format should be chosen so that the proper pixel interpretation could be implemented because each format transfers pixels differently. There are several issues when color is added to pictures, however. First, is that adding color roughly doubles the memory requirements of a picture because each pixel is now expressed as a 16-bit value as opposed to a 8-bit grayscale value. Because the picture is now a larger amount of information, this will impact Ethernet transfer time. The eZ80F91 has a maximum TCP transfer time of approximately 2 MBps. A new Ethernet controller and processor may have to be used in order to transfer information faster.

Data Compression

Another method to reduce transfer time is to implement a data compression scheme before the data is sent. This could significantly reduce the number of bits needed to be sent which will help increase the frames per second that can be displayed. The amount of time to compress the raw pixel data, however, must be operate quickly so that the net time savings for compression plus transmission time is meaningful. Adding compression is not a trivial task.

Image Processing

The processor can accomplish a variety of image processing before ever sending it to the PC. This could include color tracking, motion detection, image windowing, etc. Doing this, however, could add significant processing overhead that could affect the number of supported frames per second. This might be acceptable depending on the application.

Appendix A: Project Settings

General Tab

- CPU Family: eZ80 Acclaim!
- EZ80F91

C Tab -> Category Preprocessor

- Preprocessor Definitions: HELL,I2C,_EZ80F91,EVB_F91_MINI,EMACF91_MINI,XTL=1,UARTDEV0,TCPDEVICE,UDPDEVICE,EMAC,DHCP_REQ,_EZ80ACCLAIM!,RZKMACRODEBUG,RZK_PRIORITYINHERITANCE
- Include Paths: ..\..\Inc\TCPIPCore
- User:
..\..\RZK\ez80f91\Inc\Core;..\..\RZK\ez80f91\Inc\Bsp;..\..\RZK\ez80f91\Inc\FS;..\..\Inc\AppProto;..\..\Inc\Common;..\..\Inc\XTL;

Assembler

- Defines: ZDS1=0, RAM_MAP_ASSEMBLY=1, _INITALIZE_F91_PLL

Linker

- Object/library modules:
..\..\RZK\ez80f91\Lib\Core\RZKDebugPI.lib,..\..\RZK\ez80f91\Lib\Bsp\EMACF91MiniLib.lib,..\..\RZK\ez80f91\Lib\Bsp\UARTF91lib.lib,..\..\RZK\ez80f91\Lib\FS\NOFileSystem.lib,..\..\Lib\common.lib,..\..\Lib\NOdhcp.lib,..\..\Lib\NOdns.lib,..\..\Lib\NOFtpServer.lib,..\..\Lib\nohhttp.lib,..\..\Lib\NOigmp.lib,..\..\Lib\NOppp.lib,..\..\Lib\NOarp.lib,..\..\Lib\NOsmtp.lib,..\..\Lib\NOsnmp.lib,..\..\Lib\telnet.lib,..\..\Lib\NOftp.lib,..\..\Lib\TTY.lib,..\..\Lib\xc.lib,..\..\Lib\XTL.lib,..\..\Lib\ZTPCoreF91Mini.lib, ..\..\Lib\Shell.lib

Target

- Char: 8
- Short: 16
- Float: 32
- Long: 32
- Int: 24
- Double: 32
- Bitfield: 24
- ROM: 0000-3FFFF
- RAM: C00000-C1FFFF
- ExtIO: 0-FFFF
- IntIO: 0-FF
- FlashInfo: 0-FF

Debugger

- Driver: Serial Driver
- Target: EZ80DEVPLATFORM/F91
 - Configure
 - Program Counter: 00000
 - SPL Stack Pointer: FFFFF
 - SPS Stack Pointer: FFFF
 - Chip Select Register: CS0
 - Bound: 000000-03FFFF
 - Control Register: E8
 - Bus Mode 02
 - Start in ADL mode: check
 - Enable Data RAM: check
 - Enable EMAC RAM: check
 - Address Upper Byte: 0
 - Enable Flash:
 - Address Upper Byte: 0
 - Phase-locked loop: check
 - System Clock Frequency: 50000000
 - Oscillator Frequency: 5000000
 - Charge Pump Current: 500uA
 - Lock Criteria: 8 cycles
 - Change ZDI clock upon reset: check

Appendix B: Project Pictures



Figure 4. System overview



Figure 5. Another system overview

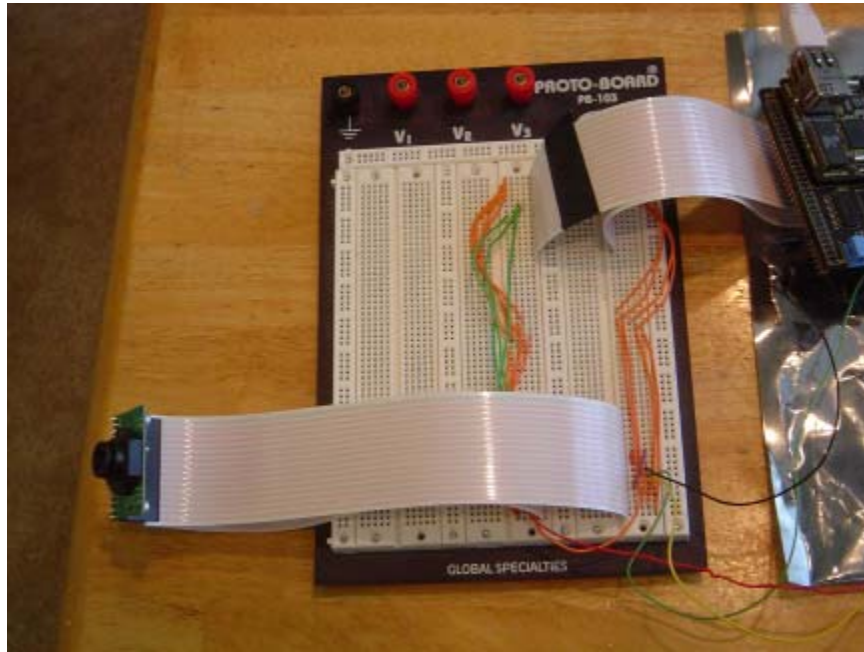


Figure 6. Camera breadboard interface

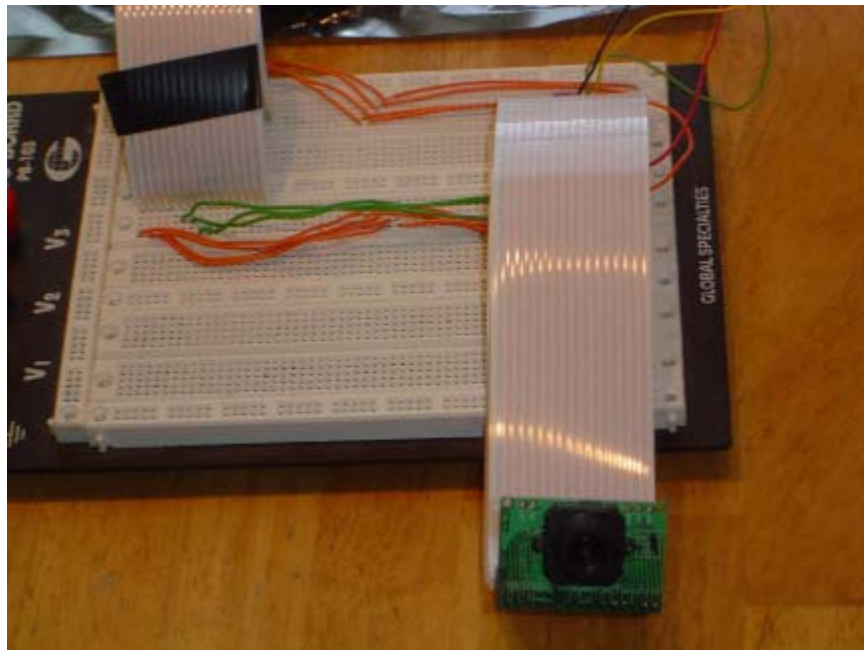


Figure 7. C3088 and breadboard



Figure 8. EZ80F91 to breadboard wiring

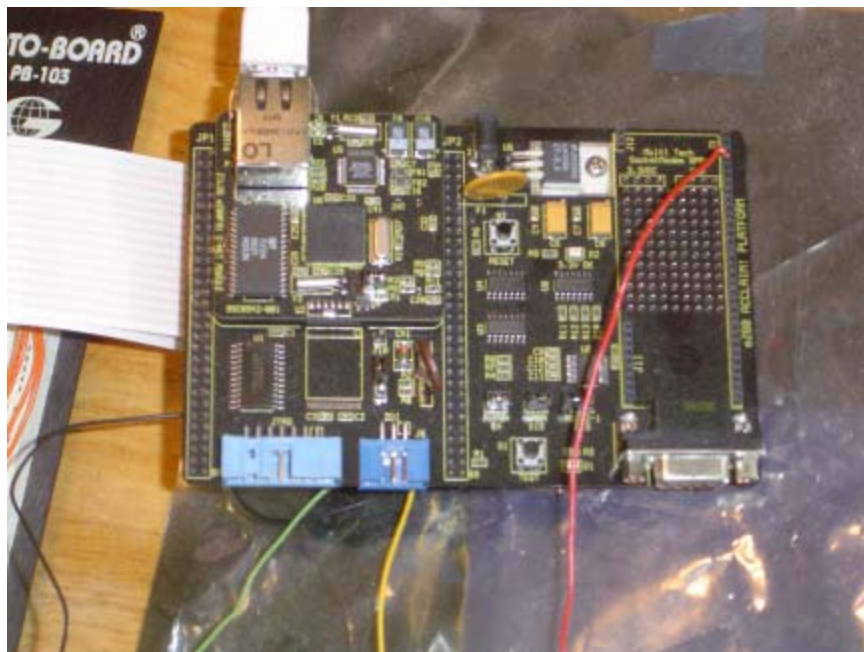


Figure 9. EZ80F91 development board



Figure 10. Bottom of the EZ80F91 development board