JAY-Cam

Design Document

Version 1.0

Yaniv Schiller
Avrum Tilman
Josh Weinberg

Table of Contents

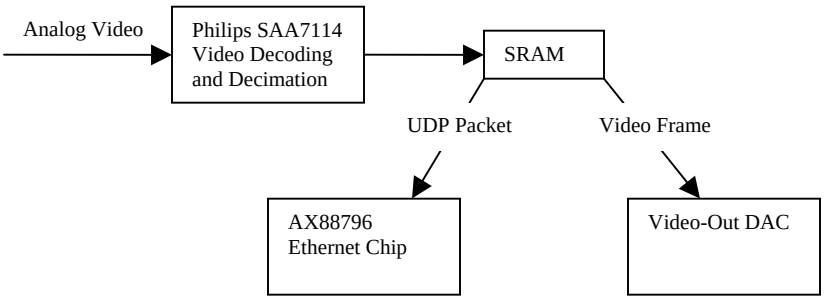
Introduction

System Design

Overview

In broad terms, we are using the FGPA and its peripherals to construct a simple video cam. This camera takes in analog video, converts it to a digital stream and broadcasts it through the Ethernet port as UDP packets.

The flow of data can be summarized in the following block diagram:



Details

The input into the Video Decoder is a Split, Differential S-Video(Y/C) Input. Our input is going to be in PAL format. We will configure the decoder to scale (decimate) the input into 128x128 pixels. Each pixel will be represented by 1 byte.

As the data streams in, it will reside temporarily in the Video Decoders's 32 word by 4 Byte FIFO register for Video Output. When possible we will remove the data from the FIFO buffer and move it into SRAM.

Concurrently, we will display the data out to a screen buffer. One trick that we hope to employ is to dump the Video-In data into the screen buffer directly.

Furthermore, every so often we will create a new packet on the Ethernet card, dumping some UDP, IP and ETHERNET data into the packet data. Then as the data is available in the SRAM, we push the data into the Packet's data portion and pass it along to the Ethernet controller.

Data sizes

The following preliminary calculations were made:
Data input: 128x128 Bytes = 16384 = 16K
This data is moved cleverly into the 640x480=307200= 300K Screen Buffer that is used to display the current image on the screen

The Ethernet card creates Ethernet Packets. This includes a 6-byte source, 6-byte destination, 2 bytes of type/length and Data.

The data portion of the Ethernet packet will consist of the following:
IP Header – 20 Bytes
UDP Header – 4 Bytes
Screen Position Information – 1 byte
128x4 = 512 bytes of Screen data.

With this setup, we send 128x4 bytes each time, requiring the application t send out 32 packets.

Components

The Clock

Implementation

Cristian suggested and helped implement a 100 MHz clock that will be used to drive the I/O operations. The System Clock will still run at 50 MHz, and the vga clock will run at 25 MHz.

Expected times

This design allows us to perform an Ethernet read (16 bit), SRAM read (32 bit), SRAM write (32 bit) in 2 clock cycles.
An Ethernet write (16 bit), SRAM read (16 bit), SRAM write (16 bit) will occur in 1 clock cycle.

Ethernet Controller (ASIX AX88796L)

Hardware

We are using the 10/100 Ethernet Interface on the MAC+PHY chip (AX88796).

Initialization

To initialize this device we will have to write to certain registers setting certain parameters.

Transmission details

Writing to Ethernet memory is not enough to initiate transmission. In order to send out data, the memory must be moved from the on-chip SRAM into the transmission FIFO. This is accomplished by an on-chip DMA operation, which is controlled by setting certain control registers. The following steps portray a high level view on what is required to transmit a packet:

1) Send Packet data to Ethernet's on-chip SRAM
    a. This includes the 128*4 data Bytes, 1 positioning Byte, 4 UDP Bytes and 20 IP header Bytes
2) Send "0x22" to the Command Register to activate the controller.
3) Start a Remote DMA write operation to transfer data from the on-chip SRAM to the FIFO.
4) Send "0x26" to the command register to set the transmit bit, thereby initiating transmission.

Registers

The Ethernet controller consists of ne2000 compatible control registers. (For a list of these registers see http://www1.cs.columbia.edu/~sedwards/classes/2004/4840/ax88796.pdf, pages 32 and 33).

Ethernet Packets

As mentioned previously, the Ethernet packets creation involves setting up the packet's source and destination MAC and filling the packet's data with

1. IP Header – 20 Bytes
    a. Version – 4 bits
        i. Value = 4= "0b0100"
    b. IHL – 4 bits
        i. Value = 5 = "0101"
    c. TOS – 1 Byte
        i. Value = 0 "0b00000000"
    d. Total Length – 2 Bytes
        i. Value =512+1+4+20= 537 = "0x217"
    e. Identification – 2 Bytes
        i. A UNIQUE number
    f. Flags – 3 bits
        i. Value = "010"
    g. Fragment Offset - 13 bits
        i. "0000000000000"
    h. TTL – 1 Byte
        i. "11111111"
    i. Protocol – 1 Byte
        i. Value = UDP = 17
    j. Header Checksum – 2 Bytes
        i. COMPUTED On the fly
    k. Source IP - 4 Bytes
        i. Constant Value to Be Determined
    l. Destination IP – 4 Bytes
        i. Constant Value to Be Determined
    m. Data
        i. UDP Header - 4 bytes
        ii. Data – 513 bytes
2. UDP Header – 4 Bytes
    a. Source Port – 1 byte
        i. 5001
    b. Destination Port – 1 byte
        i. 5001
    c. Length – 1 byte
        i. 512+1+4 = 517
    d. Checksum – 1 byte

        i. Value = “0”, disable the checksum
3. Screen Position Information – 1 byte
    a. Because the screen is split into 32 128x4 packets, each packet will have a position number of 0-31
4. 128x4 = 512 bytes of Screen data.
    a. This is the corresponding data read from the SRAM.

Most of the Header information is constant and can be hardcoded into memory.
The UDP checksum is hardcoded to 0, thus disabling it.
Each Ethernet packet requires a unique Identification Number at the IP layer. Because this changes, the IP header Checksum is recalculated on the fly.

SRAM (Toshiba TC55V16256J)

Hardware

Toshiba TC55V16256J.

Features

256K x 16 bits of available memory. This is 512 K.

Usage

We are using this peripheral to dump the image data from the Video decoder to the frame buffer. We are using 640x480 = 300K of memory for the Frame Buffer for the video output. The Video input will go to only a 128x128 bit subset of this. The rest will be zeroed out. We do this in this way to make the video out easier to implement.

Video Out (Texas Instruments THS8133B)

Hardware

We are using the onboard Texas Instruments THS8133B Digital To Analog Converter.

Features

This chip includes a triple 10 bit Digital To Analog Converter (DAC).
Most of the code for this device was written and handled in labs 5 and 6.

Video In (Philips SAA7114H)

Hardware

Video Input: We are using the Philips SAA7114-H decoder chip.

Features

This chip automatically digitizes a NTSC, SECAM and PAL video signals.

Initialization

We will have to send in a bunch of control signals, configuring the device.
This information includes Scalar, pixels… This is set up using the I2C Bus.
Further information will be decided during implementation.

The Output of this device is a 128x128 byte FIFO Buffer that will be placed into the SRAM.

Registers
All the registers need to be set according to the PAL settings listed in the Philips manual (http://www1.cs.columbia.edu/~sedwards/classes/2004/4840/SAA7114H_1.pdf) on pages 127 and 128.

OPB Bus

Arbiter

This is the most complex part of the project. We have to write an arbiter to decide which component has access to the bus. The following 3 components are waiting to use the bus:

1) The Video Input wants to write 128x128 Bytes to SRAM.
2) The Video Output wants to read 640x480 Frame Bytes from SRAM to the DAC.
3) The CPU wants to write packets to the Ethernet Controller.

Memory Mapping

At this point we have a preliminary memory mapping that is being used in the bus arbiter (ie addressing the SRAM and Ethernet memory).

A proper OPB device will be addressed in the following way (Note that the address given to the OPB controller is 32 bits wide).

Bit Position	Value
31:23	“000000001”
22:21	“00” – SRAM “01” - Ethernet
20:1	Device Address

The Ethernet’s onboard memory is mapped as follows (offset from starting address, which by default is set to 0x200):
MAC Core Registers – 0x000 to 0x3FFF
On-Board SRAM – 0x4000 to 0x7FFF

At the time of this writing, memory-addressing issues are being worked on and should soon be fixed.

It should be noted that all reads and writes to the Ethernet are 16 bits, while reads and writes to the memory can be (and will probably be) 32 bits.

Inter-Component Wiring

The following pin connections are to be placed in the UCF file:

#UCF FILE
```
net FPGA_CLK1 loc="p77";
net PB_LB_N loc="p140";          #BAR9
net PB_UB_N loc="p146";  #BAR10
```

```
net PB_WE_N loc="p123";
net PB_OE_N loc="p125";

net RAM_CE_N loc="p147";

net PB_A<0> loc="p83"; #BAR1
net PB_A<1> loc="p84"; #BAR2
net PB_A<2> loc="p86"; #BAR3
net PB_A<3> loc="p87"; #BAR4
net PB_A<4> loc="p88"; #BAR5
net PB_A<5> loc="p89"; #BAR6
net PB_A<6> loc="p93"; #BAR7
net PB_A<7> loc="p94"; #BAR8
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";
net PB_A<18> loc="p121";
net PB_A<19> loc="p122";


net PB_D<0> loc="p153"; #LEFT_A
net PB_D<1> loc="p145"; #LEFT_B
net PB_D<2> loc="p141"; #LEFT_C
net PB_D<3> loc="p135"; #LEFT_D
net PB_D<4> loc="p126"; #LEFT_E
net PB_D<5> loc="p120"; #LEFT_F
net PB_D<6> loc="p116"; #LEFT_G
net PB_D<7> loc="p108"; #LEFT_DP
net PB_D<8> loc="p127"; #RIGHT_A
net PB_D<9> loc="p129"; #RIGHT_B
net PB_D<10> loc="p132"; #RIGHT_C
net PB_D<11> loc="p133"; #RIGHT_D
net PB_D<12> loc="p134"; #RIGHT_E
net PB_D<13> loc="p136"; #RIGHT_F
net PB_D<14> loc="p138"; #RIGHT_G
net PB_D<15> loc="p139"; #RIGHT_DP

net ETHERNET_CS_N loc="p82";
net ETHERNET_IOCS16_N loc="p74";
net ETHERNET_RDY loc="p81";
net ETHERNER_IREQ loc="p75";

net VIDIN_ICLK loc="p185";
net VIDIN_IDQ loc="p205";
net VIDIN_ITRDY loc="p206";
net VIDIN_ITRI loc="p204";
net VIDIN_IGPH loc="p200";
net VIDIN_IGPV loc="p201";
net VIDIN_IGP<0> loc="p203";
net VIDIN_IGP<1> loc="p202";
net VIDIN_IPD<0> loc="p188";
net VIDIN_IPD<1> loc="p189";
net VIDIN_IPD<2> loc="p191";
net VIDIN_IPD<3> loc="p192";
net VIDIN_IPD<4> loc="p193";
net VIDIN_IPD<5> loc="p194";
net VIDIN_IPD<6> loc="p198";
net VIDIN_IPD<7> loc="p199";
net VIDIN_HPD<0> loc="p174";
net VIDIN_HPD<1> loc="p175";
net VIDIN_HPD<2> loc="p176";
net VIDIN_HPD<3> loc="p177";
net VIDIN_HPD<4> loc="p178";
```

```
net VIDIN_HPD<5> loc="p179";
net VIDIN_HPD<6> loc="p180";
net VIDIN_HPD<7> loc="p187";

net I2C_SCL loc="p6"
net I2C_SDA loc="p5"

net VIDOUT_CLK loc="p23";
net VIDOUT_BLANK_N loc="p24";
net VIDOUT_HSYNC_N loc="p8";
net VIDOUT_VSYNC_N loc="p7";

net VIDOUT_RCR<0> loc="p41";
net VIDOUT_RCR<1> loc="p40";
net VIDOUT_RCR<2> loc="p36";
net VIDOUT_RCR<3> loc="p35";
net VIDOUT_RCR<4> loc="p34";
net VIDOUT_RCR<5> loc="p33";
net VIDOUT_RCR<6> loc="p31";
net VIDOUT_RCR<7> loc="p30";
net VIDOUT_RCR<8> loc="p29";
net VIDOUT_RCR<9> loc="p27";

net VIDOUT_GY<0> loc="p9" ;
net VIDOUT_GY<1> loc="p10";
net VIDOUT_GY<2> loc="p11";
net VIDOUT_GY<3> loc="p15";
net VIDOUT_GY<4> loc="p16";
net VIDOUT_GY<5> loc="p17";
net VIDOUT_GY<6> loc="p18";
net VIDOUT_GY<7> loc="p20";
net VIDOUT_GY<8> loc="p21";
net VIDOUT_GY<9> loc="p22";

net VIDOUT_BCB<0> loc="p42";
net VIDOUT_BCB<1> loc="p43";
net VIDOUT_BCB<2> loc="p44";
net VIDOUT_BCB<3> loc="p45";
net VIDOUT_BCB<4> loc="p46";
net VIDOUT_BCB<5> loc="p47";
net VIDOUT_BCB<6> loc="p48";
net VIDOUT_BCB<7> loc="p49";
net VIDOUT_BCB<8> loc="p55";
net VIDOUT_BCB<9> loc="p56";
```