Robotic Vision System

Design Documentation

Embedded Systems EE 4840

Prof. Steven Edwards

Team Members:

Han Cheng Liang

Dawit Bekele

Edward Mung

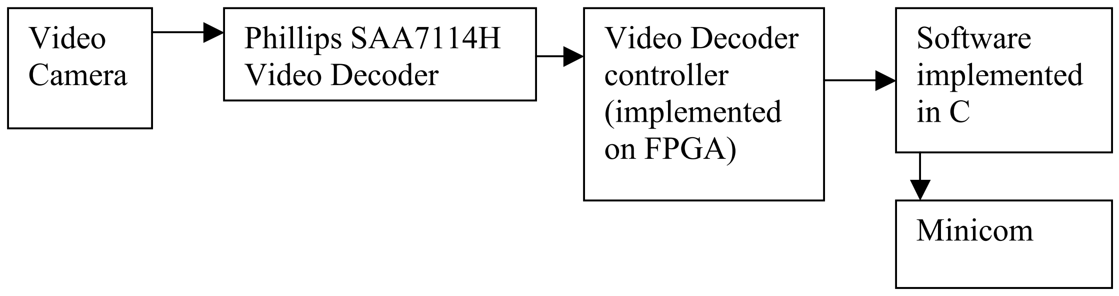
Alison Leonard

Overview

We plan to design the vision system for a Lego robot. Ultimately, the robot will have the ability to follow a moving target. The functional requirement for our vision system is to be able to capture video signals from the surrounding environment, process those video signals, and output action commands to the Lego robot. We plan to get the video decoding, and image processing, decision making working first, and then see if we have time to interface with a lego robot that acts upon those decisions. We expect that programming the lego robot will be simple, so we hope to have the time. However, getting the robot to work could involve lots of trial and error. Thus we will concentrate on the machine vision part. We can output action commands (such as "turn left", "turn right", etc), and intermediate results in the video signal processing stage to the minicom, or the display using the video controller Prof. Edwards implemented.

Our image processing will be color based. We will choose an unusual color for the moving target, so that it will be easily distinguished from the background.
When processing the pixel images, we will mark them as color or no color based on whether they match our interested color and are above a defined intensity threshold. We will need to test the video input to find the correct threshold to identify the color of our target.

High Level Block Diagram

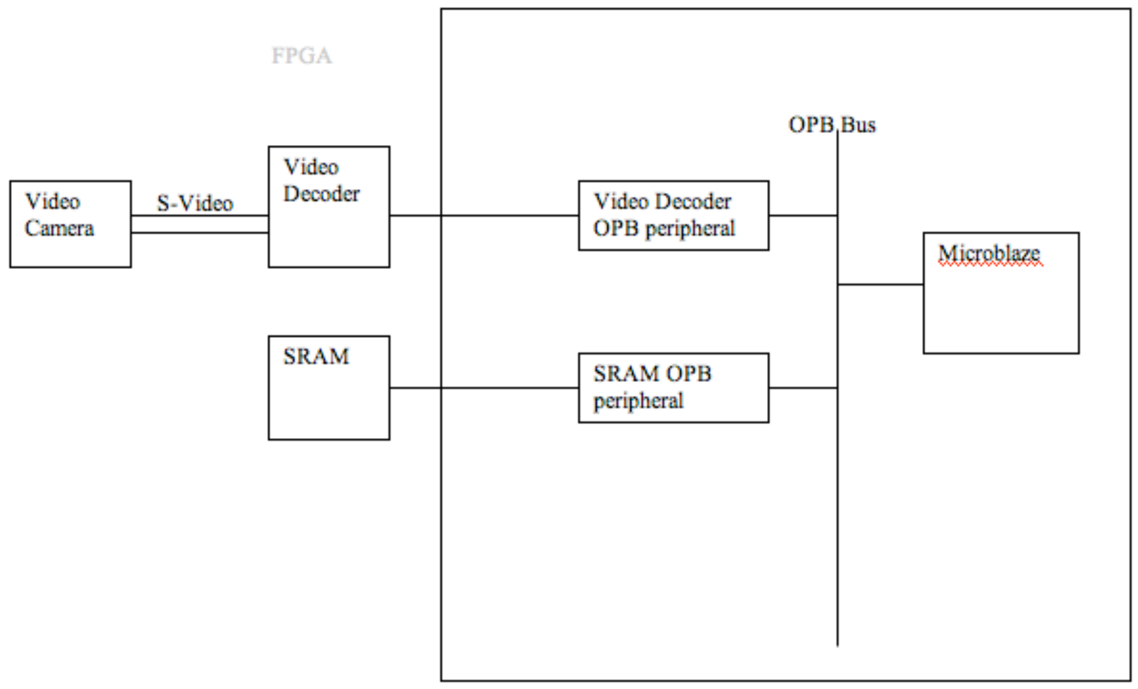

Video Camera
Phillips SAA7114H Video Decoder
Video Decoder controller (implemented on FPGA)
Software implemented in C
Minicom

The digital camcorder we are using is Canon ZR45, which supports both S-video and RCA analog video output. The camcorder will be connected to the on-board Phillips video decoder. We will use the S-video inputs.

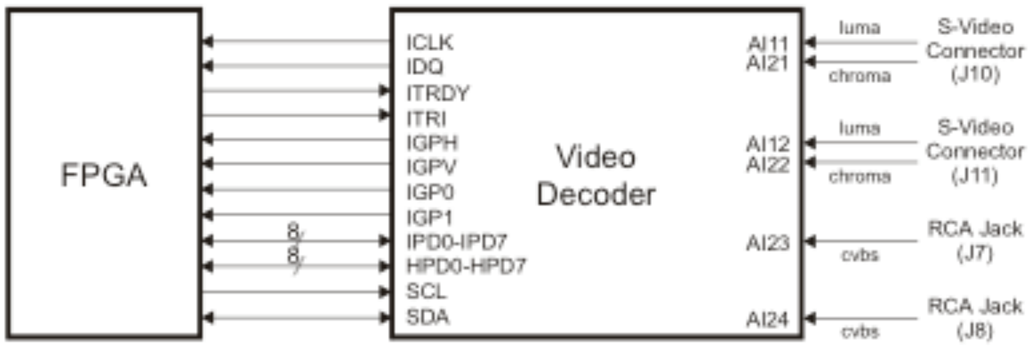
Video Format

The video is in NTSC format. We are capturing 640x480 pixel frames. Each pixel has a chrominance and luminance value. The luminance values register every pixel. There are two chrominance values: U and V. Each registers every other pixel. This is 4:2:2 YUV format. Thus we will take two pixels to be one unit. We wish to extract color information and we want to include both chrominance values for the color signature. Luminance values range from 16 (black) to 235 (white). Chrominance U (C_B) values range from 16 (yellow) to 240 (blue). Chrominance V (C_R) values range from 16 (cyan) to 240 (red). The value 128 indicates no chrominance.

System Block Diagram



Pin Connections Video Decoder to FPGA:

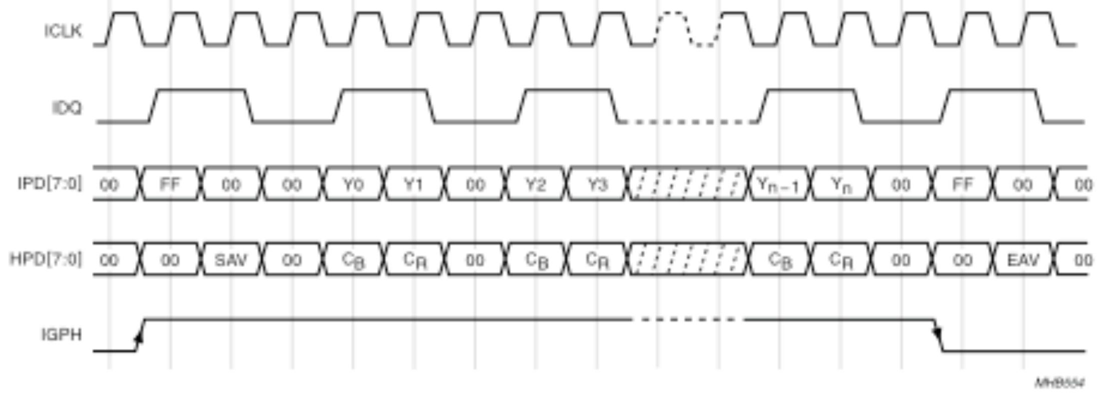


Video arrives at the FPGA over two eight bit buses. The IDP bus is for luminance values (Y) and the HDP bus is for chrominance values (UV).
The ICLK signal clocks received pixels from the video decoder. The FPGA gets 16 bits every clock cycle. The IDQ signal indicates whether the data on the IDP and HDP buses is valid. The pixel data is transmitted with varying size gaps, so the IDQ is crucial to operation. The IGPH (horizontal reference) and IGPV (vertical reference) signals tell when we are at the end of a frame and the beginning or end of a line. We will need the end of frame signal to separate data blocks for processing. IPGH and IPGV work as follows:

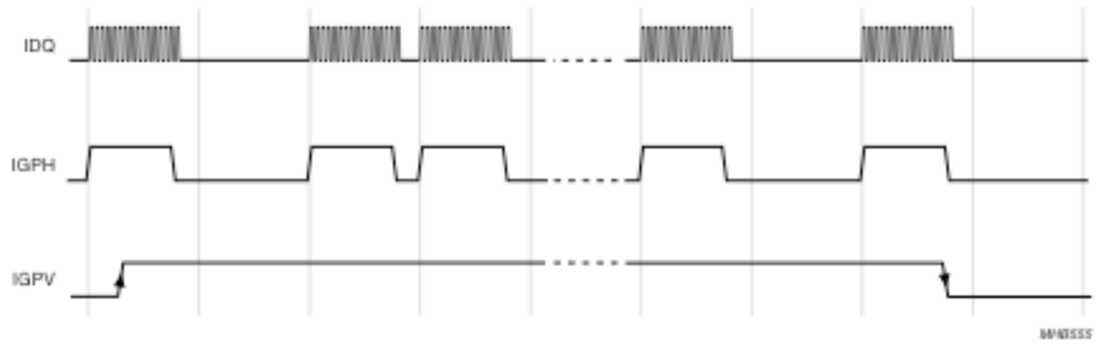
Href: active	Vref: active	next pixel first pixel of active line
Href: inactive	Vref: inactive	prev pixel last pixel of last active line
Href: inactive	Vref: active	prev pixel last pixel of active line
Href: active	Vref: inactive	next pixel first pixel of any vblanking

The ITRI and ITRDY signals will be kept at “1” at all times. We do not need to use the SCL or SDA signals.

Timing Diagram for Image Data coming to FPGA



This timing diagram shows the beginning and end of a line. SAV means Start Active Video and EAV means End Active Video. The diagram shows the role played by IDQ and IPGH. IPGH indicates the beginning and end of each active line.



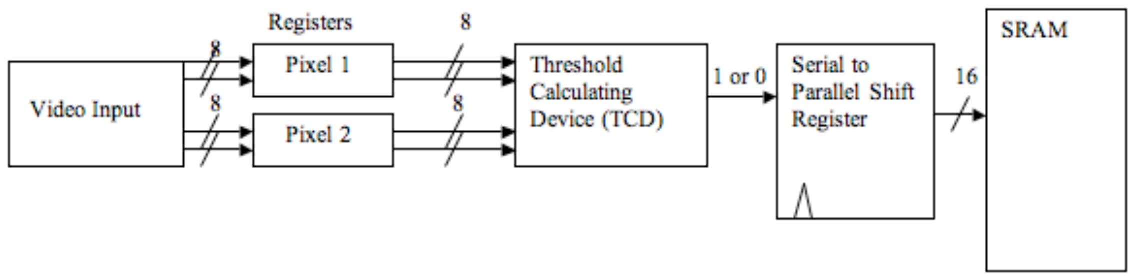
This timing diagram shows the role of the IPGV signal. It indicates the beginning and end of each frame. Also, we see that IDQ is constantly going on and off in the middle of active video lines.

Video Processing on the FPGA

We will do threshold processing of pixels on the FPGA. Each set of two 16-bit pixels will be a one bit threshold mark. The bits will be fed into a shift register that holds 16 pixels. Each set of 16 pixels will be stored in SRAM.

The data stored in SRAM will be read by our C program one frame at a time. We know that each image line will be 640/2 pixels/ 16 bits = 20 rows in SRAM. The entire frame takes 480 * 20 = 17,600 lines. The C program will hold an array with one entry for each pixel.

Block Diagram for Video Processing on FPGA



Registers
8
8
8
8
Video Input
Pixel 1
Pixel 2
Threshold Calculating Device (TCD)
1 or 0
Serial to Parallel Shift Register
16
SRAM

Software

A C program will analyze each frame of data. From the matrix of real numbers, X_t, which represents the image at a given time t, the program will calculate a corresponding action α_i by a function F: $F(X_t) \rightarrow \alpha_i$.

The center of mass of the moving target will correspond to a direction we wish the robot to follow.

The output of the software is commands in a certain format, such as “left, 30 degrees”.

If time permits, we will add the module that executes on these commands and tell the lego robot to follow these directions. Either way, we will output the directions on the minicom or to the display via the video controller we had in the previous lab.

We might also make a velocity function. The robot should travel at different speeds depending on how far away the ball is. We may be able to tell how far away the ball is by how many pixels it covers.