CSEE 4840 - Embedded Systems Lab

ObTrack Design Document

Date: 01/04/2004

Project Team Members:
Marcio Buss (mob2101@columbia.edu)
Anuj Maheshwari (atm2104@columbia.edu)
Brian Yanity (bby2001@columbia.edu)

1. Introduction

Image and pattern recognition has been an area of much research and development in recent years. An immaculate number of complex software have been written to get the best level of pattern recognition. The applications of this type of software range from security systems to simple forms of targeting in missile launch systems. The basis for this project are primitive forms of pattern recognition and possibly if time permits color distinction.

The initial part of this document (sections 2-4) describes the project as a block diagram, highlighting relevant aspects of video image capturing and manipulation. This project uses a XSB Board containing, among other components, (1) a Xilinx SpartanIIE FPGA with 300K system gates (2) a Philips SAA7114H video decoder (3) a 256K x 16 SRAM (4) a Texas Instruments THS8133B video DAC.

2. Project Block Diagram

The object tracker "Obtrack" is sketched in the following block diagram.

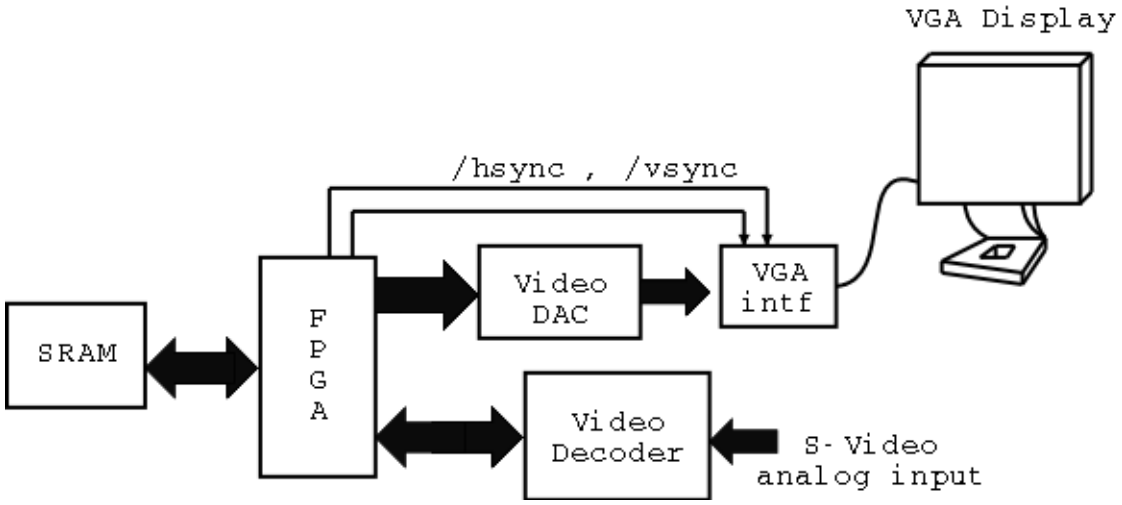


Fig. 1 - Obtrack block diagam.

Basically, from the S-video connector J10 (on the XSB board) comes the analog video signal in PAL mode (or NTSC - to be settle). The analog signal is digitized by the video decoder and arrives at the FPGA through the IPD (SAA7114H I-port) and HPD (SAA7114H H-port) buses. Inside the FPGA, the digital video frame is scanned and the object to be tracked is searched. Microblaze CPU is used for these tasks. The following figure shows in more details the modules inside the FPGA. *opb2xess bridge* arbitrates the access to memory among the various modules such as VGA, CPU and the video decoder interface (we still have to decide if the data portion of video decoder interface is going to be connected directly to OPB Bus or as shown in the figure). *i2c controller* is used to drive the I2C bus that configures the video decoder, and it is already working.

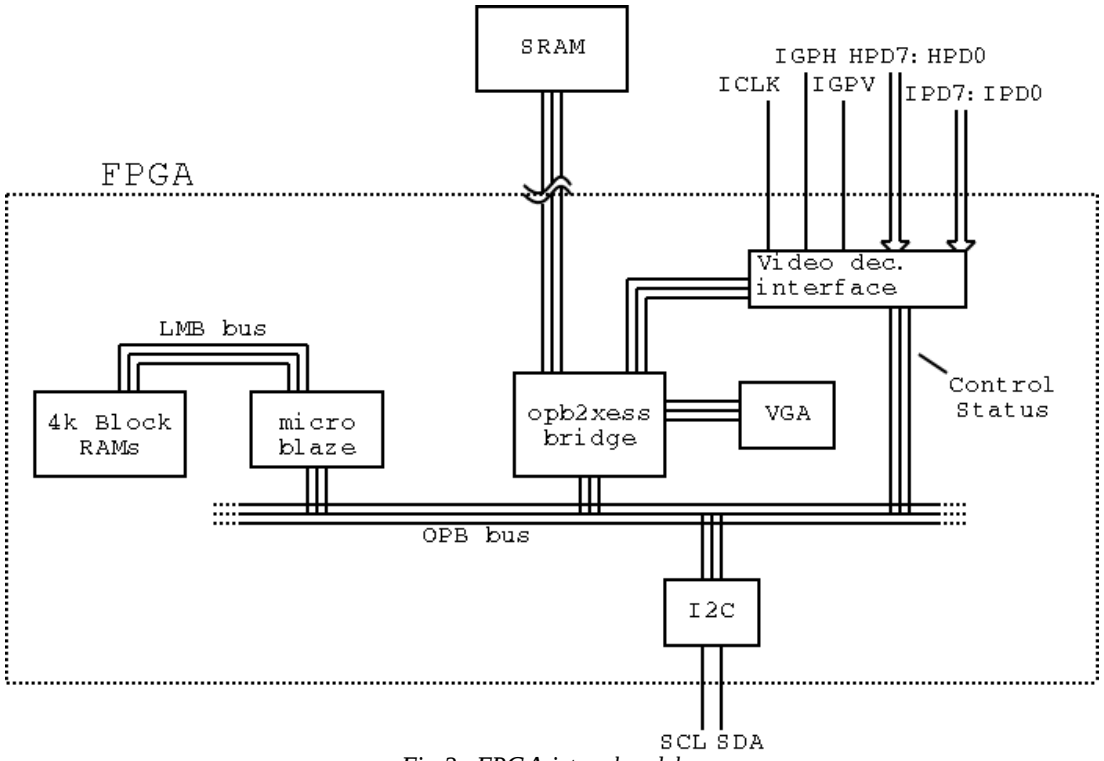
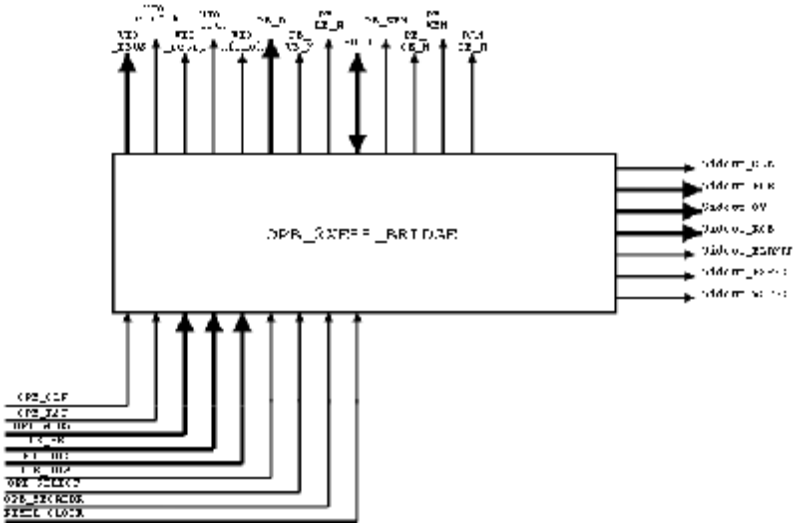


Fig. 2 - FPGA internal modules.

The flow of data starts on the J10 S-video (analog) input and passes through the video decoder to the IPD and HPD buses. The digitized pixels are transmitted to the video decoder interface (shown in the above figure) which is connected to the *opb2xess bridge*. Whenever control is allowed for the video decoder interface, data is passed to the SRAM via the arbiter. There is a FIFO buffer inside the video decoder interface to handle incoming streams of data that have not yet been written into the memory. Whenever an entire frame is in memory, the algorithm described on section 5 is executed, and the image is shown in the VGA display. *opb2xess bridge* acts as the arbiter here as well. A more detailed view of the *opb2xess bridge* interconnections is shown below.



3. Video Decoder Input/Output Buses

The following figures shows a more detailed interface between the FPGA and the video decoder.

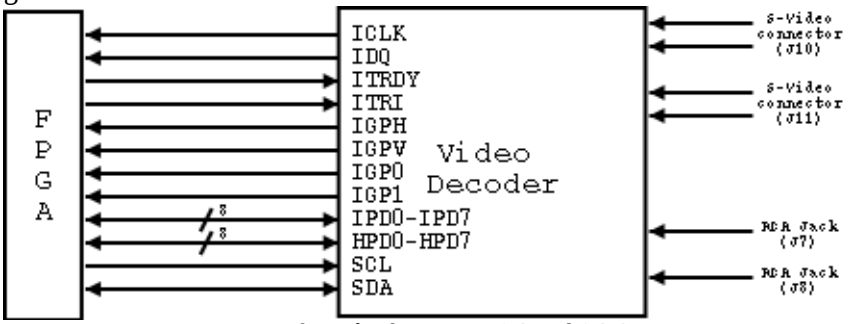


Fig. 3 - External interface between FPGA and SAA7114H.

IPD0-IPD7 and HPD0-HPD7 are the data buses used to pass digital image information from the video decoder to the FPGA. IDQ is asserted to '1' whenever there is a valid data on those buses. IGPH and IGPV are the counterpart of horizontal and vertical sync signals (not exactly sync signals, but they do establish end-of-line and end-of-frame boundaries). SCL and SDA correspond to the I2C interface used to program the video decoder. A complete list showing the correspondence between the FPGA and each signal or bus in Fig. 3 is described on Table 1.

Table 1.

Video input Bus	Video Decoder Pin	FPGA Pin	Function
VIDIN-ICLK	ICLK	185	Image interface clock output
VIDIN-IDQ	IDQ	205	Image data qualifier (data valid)
VIDIN-ITRDY	ITRDY	206	Target ready input
VIDIN-ITRI	ITRI	204	Image port tristate input
VIDIN-IGPH	IGPH	200	Multi-purpose horiz. reference
VIDIN-IGPV	IGPV	201	Multi-purpose vertical reference
VIDIN-IGP0	IGP0	203	General-purpose output
VIDIN-IGP1	IGP1	202	General-purpose output
VIDIN-IPD0	IPD0	188	Image port data line 0
VIDIN-IPD1	IPD1	189	Image port data line 1
VIDIN-IPD2	IPD2	191	Image port data line 2
VIDIN-IPD3	IPD3	192	Image port data line 3
VIDIN-IPD4	IPD4	193	Image port data line 4
VIDIN-IPD5	IPD5	194	Image port data line 5
VIDIN-IPD6	IPD6	198	Image port data line 6
VIDIN-IPD7	IPD7	199	Image port data line 7
VIDIN-HPD0	HPD0	174	Image port data line 8
VIDIN-HPD1	HPD1	175	Image port data line 9
VIDIN-HPD2	HPD2	176	Image port data line 10
VIDIN-HPD3	HPD3	178	Image port data line 11
VIDIN-HPD4	HPD4	179	Image port data line 12
VIDIN-HPD5	HPD5	180	Image port data line 13
VIDIN-HPD6	HPD6	181	Image port data line 14
VIDIN-HPD7	HPD7	187	Image port data line 15
I^2C-SCL	SCL	6	I^2C clock

I^2C-SDA	SDA	5	I2C data

The output from the video decoder can be either 8 or 16 bits wide: in the 16-bit mode, adopted in this project, data pins HPD7 to HPD0 are used for chrominance data. However, we are only going to use the luminance ("Y") values coming from I-Port, essentially discarding the H-Port bytes.

4. Video DAC Input/Output Buses

The following figures shows a detailed interface between the FPGA and the video DAC.

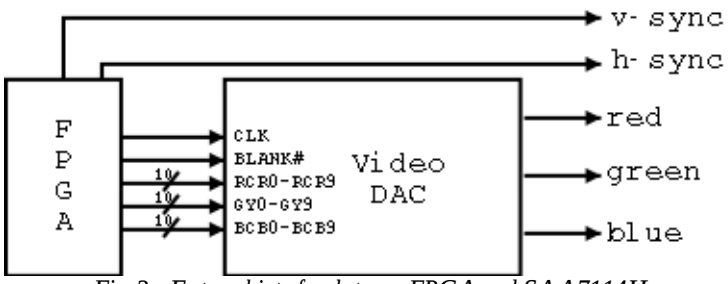


Fig. 3 - External interface between FPGA and SAA7114H.

Table 2 below briefly describes each signal or bus used to communicate the video DAC and the FPGA.

Table 2.

Video Output Bus	Video DAC Pin	FPGA Pin	Function
VIDOUT-CLK	CLK	23	Pixel clock
VIDOUT-BLANK#	BLANK#	24	Blanking signal
VIDOUT-HSYNC#	NC	8	Horizontal sync
VIDOUT-VSYNC#	NC	7	Vertical Sync
VIDOUT-RCR0..9	RCR0..9	41,40,36,35,34 33,31,30,29,27	Pixel red components
VIDOUT-GY0..9	GY0..9	9,10,11,15,16, 17,18,20,21,22	Pixel green components
VIDOUT-BCB0..9	BCB0..9	42,43,44,45,46 47,48,49,55,56	Pixel blue components

The horizontal and vertical sync signals are generated by the VGA timing module inside the FPGA.

5. Object tracking algorithm

In the following section, the pseudo code for searching for a basic white rectangle (dimension M x N) in a frame is described (image is monochrome, with 1 signifying black, 0 signifying white):

```
Step 01
Grab image from video processor / camcorder, perform resizing / color
manipulations

Step 02
Store the image to memory (into a A x B matrix)

Step 03
Set counter TOTAL = 0

Step 04
Set counters X = 1, Y = 1, MATCH_X=0, MATCH_Y=0

Step 05
G = X, H = Y

Step 06
TOTAL = TOTAL + CAM_IMAGE[G,H]

Step 07
If G < (X+M),  G = G+1; JUMP to Step 06

Step 08
If H < (Y+N), H = H+1; Jump to Step 06

Step 09
If TOTAL = 0 (all locations 0) MATCH_X = X, MATCH_Y = Y; JUMP to Step 12


Step 10
If X < (A-M) X = X+1; JUMP to Step 05

Step 11
If Y < (B-N) Y = Y+1; JUMP to Step 05

Step 12
If MATCH_X !=0 PRINT “MATCH FOUND AT” MATCH_X , MATCH_Y
```

An alternate implementation of this algorithm where the dimension of the square is unknown is given below. First an assumption for the minimum dimension to be considered a square must be made, let it be an integer ‘MIN_DIM’. The following algorithm would be used thereafter:

```
Step 01
```

```
Grab image from video processor / camcorder, perform resizing / color
manipulations

Step 02
Store the image to memory (into a A x B matrix, called CAM_IMAGE)

Step 03
Set counter TOTAL = 0 , DIMENSION = 0 , T = MIN_DIM

Step 04
Set counters X = 1, Y = 1, MATCH_X=0, MATCH_Y=0

Step 05
G = X, H = Y, T = MIN_DIM
Step 06
TOTAL = TOTAL + CAM_IMAGE[G,H]

Step 07
If TOTAL = 0 GOTO Step 08; ELSE GOTO Step 13

Step 08
If ((G-X) < T) THEN G = G + 1, GOTO Step 06 ; ELSE GOTO STEP 09

Step 09
If ((H-Y) < T) THEN H = H + 1, GOTO Step 06 ; ELSE GOTO STEP 10

STEP 10
MATCH_X = X, MATCH_Y = Y, DIMEMSION = T

Step 11
T = T + 1, GOTO STEP 08

Step 12
If TOTAL = 0 (all locations 0) MATCH_X = X, MATCH_Y = Y; GOTO Step 15

Step 13
If X < (A-T) THEN X = X+1; GOTO Step 05

Step 14
If Y < (B-T) THEN Y = Y+1; GOTO Step 05

Step 15
If MATCH_X !=0 PRINT “MATCH FOUND AT” MATCH_X , MATCH_Y, DIMENSION
```

The algorithms above could be altered to look for a color by making a change to the test where the TOTAL variable is checked to be zero (Step 6). Instead of adding up the values of the pixel colors, it should be repeatedly checked against the appropriate color code.