FPGA BASED OBJECT TRACKING SYSTEM

PROJECT REPORT

Design of Embedded System Advanced Course-EDA385
Department of Computer Science, Lund University

Submitted By

HARSHAVARDHAN KITTUR – ***aso10hki@student.lu.se***

CHUANHAI BAI-*aso10cba@student.lu.se*

Abstract

In this project we implement Object Tracking System which uses various video processing algorithms, after investigation we have chosen Object Tracking based on delta-framing and COG(Centre of gravity calculation) which have to be implemented on Digilent Nexys2 development board using Xilinx EDK. A hardware/software design approach will be used to partition this system blocks into hardware and software domains based on their criticality.

Contents

1. INTRODUCTION

Object Tracking is a process of locating the object to associate the target in successive video frame over time and it finds wide scale applications in the field of security and surveillance, video communication, augmented reality, traffic control, medical imaging etc. Object Tracking is a complex process to be implemented in hardware mainly because of the amount of data associated with the video, and hence FPGAs provide a good medium of implementation because of parallelism, low cost and low power.

In this project we investigated the best algorithm which can be implemented on FPGA and choose a optimum algorithm which is used to calculate the centre of gravity (COG) of the object to track it across the frame.

2. ALGORITHIMIC BLOCK DIAGRAM

The following modules construct the Object Tracking System:

1. Pre-processing of the video
2. Delta Frame Generation
3. Thresholding
4. Filtering and COG Calculation and
5. Trace Identification and display.

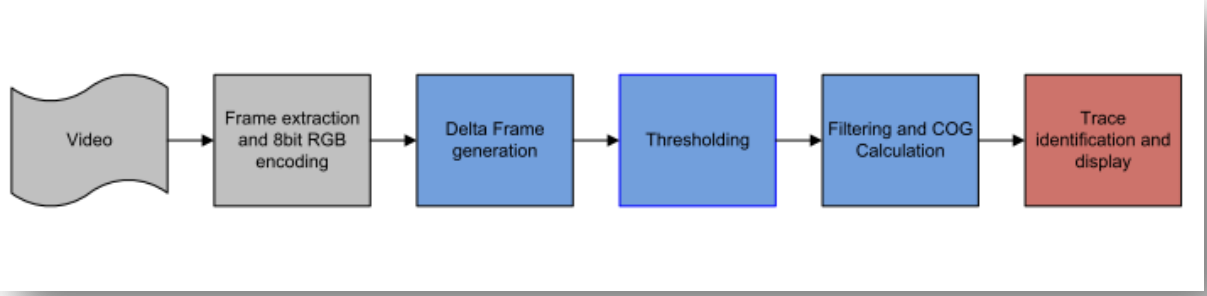

Figure 1: Object Tracking Algorithm Flow

1. Pre-processing of the video

Each frame of the 4sec demo video is extracted at the rate of 25 frames/sec in MATLAB. The 100 frames are initially generated as bmp files (640x480 pixels) and then the image RGB values in the bmp files are encoded as 8 bit RGB with 3 bits for R, 3 bits for G and 2 bits for B and written into binary files which can be directly dumped into the RAM. We differentiate between background and object frame before writing into the memory which is helpful for the delta frame generation and also reduces the coherent effect of the environment and allows us to easily separate the object from the background.

2. Delta Frame Generation

Once the pre-processing of the video has been completed, the respective frames are subtracted from one another. The resulting frame is called the Delta Frame. This method of image subtraction eliminates the background and brings the object into focus, giving us information about its shape and size. The Delta frame also reduces the number of pixels that the system will have to process.

3. Thresholding

In order to further enhance the resolution of the delta frame Thresholding is done. Example – Figure 2. The individual pixels in the grayscale image are marked as object pixels if their value is greater than some threshold value (initially set as 40) and as background pixels otherwise. In this case the object pixel is given a value of "1" while a background pixel is given a value of "0."

Figure 2: Gray Level Thresholding

4. Filtering and COG Calculation

Next step is that we run a median averaging filter over the images to remove the noise in the background. And then center of gravity (COG) is used for tracking the target. COG is a geometric property of any object which is the average location of the weight of an object. The COG (ro,co) of a 480x640 image I (x,y) is given by

$$(ro, co) = \frac{\sum r.I(ri, ci)}{\sum I(ri, ci)}, \frac{\sum c.I(ri, ci)}{\sum I(ri, ci)}$$

In general, determining the centre of gravity is a complicated procedure because the mass may not be uniformly distributed throughout the object. In order to simplify the problem we assume the object is composed of uniform material. An operator scans the entire length of the image frame for the first white pixel. This is a clear indication of the 2D position of the object within that time frame. This is iterative process and it repeated over all the frames.

5. Trace Identification and Display

We can get the center of gravity of the project from each iterative scanning. The updated locations of the pixel points are collected to provide the approximate path taken by the object. The iterative program acquires the frames and plots the individual points to display the object path.

3. MATLAB BEHAVIOURAL MODEL

A matlab behavioural model was written in order to check the system output at every processing block to verify the system functionality. The following screen dumps show the matlab output at various stages.

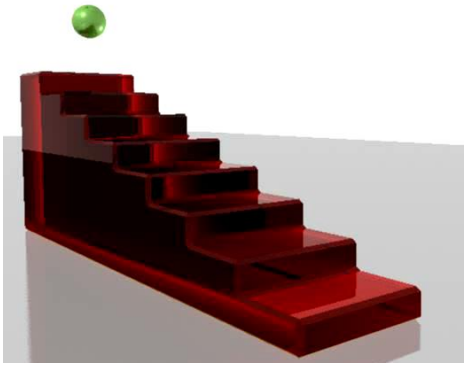

Figure 3: Bitmap of frame captured

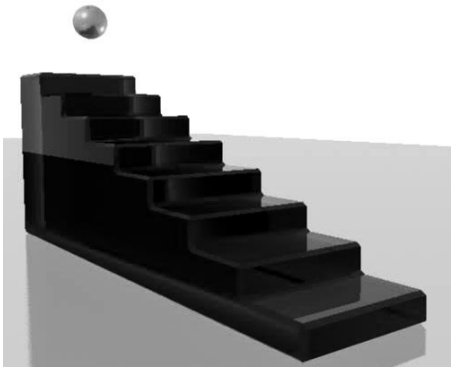

Figure 4: Gray scale of the Bitmap

Figure 5: Delta frame to isolate the object

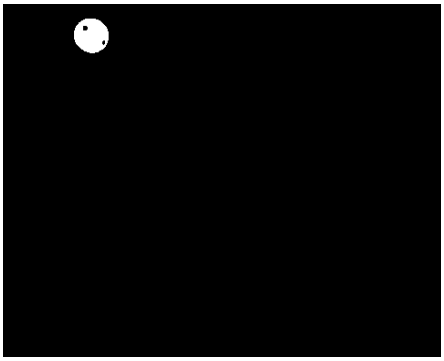

Figure 6: Result after thresholding

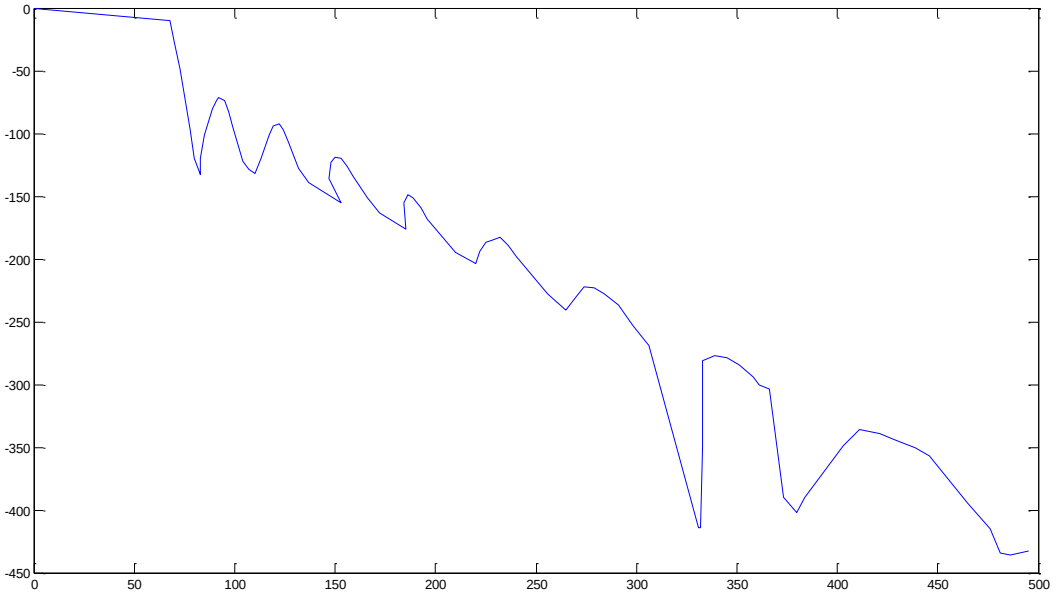

Figure 7: Plot of the centre of gravity of the objects for different frames

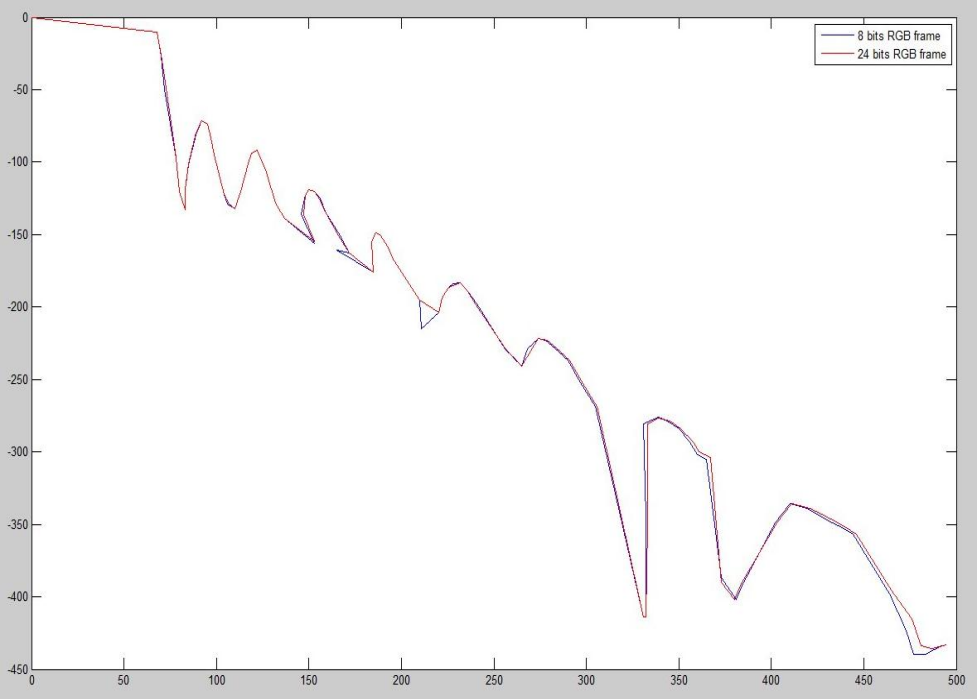

Figure 8: Plot of the difference in object trace for 8 bit and 24 bit RGB frames

4. DESIGN COMPONENTS : HW AND SW PARTITIONING

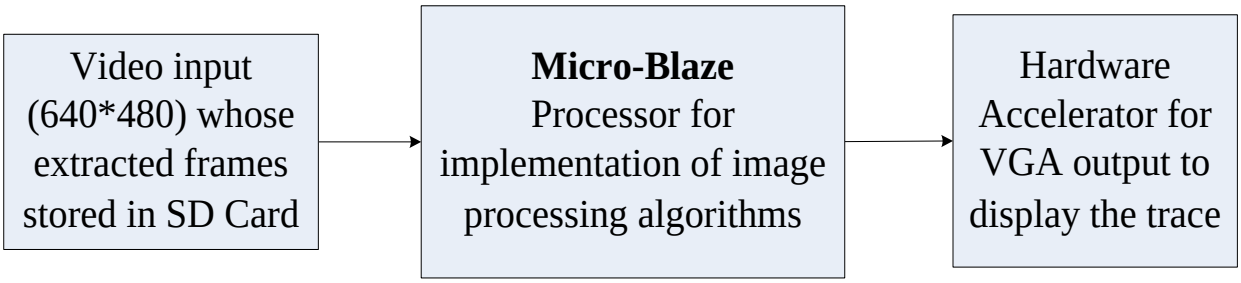

Figure 8: Components for implementation of Object Tracking

Figure 3 shows different components for our implementation of Object Tracking Algorithm. We use SD card formatted in FAT12 format to store the preprocessed images in a 8bit RGB encoded machine format which can be directly dumped into SDRAM. The pre-processed image of each frame of demo video utilize Micro-Blaze as processor to implement the image processing algorithms such as delta frame generation, thresholding and filtering and also for calculation of center of gravity (COG).

Finally, the trace of the object is be displayed on VGA monitor. We partition VGA controller to be implemented into hardware since producing VGA output RGB values for a 480 x 640 image at a refresh rate of 60 Hz is exhaustive process and is not optimal to be developed in software.

5. CONSTRAINTS

Frame buffering requires large amounts of memory. After pre-processing of each frame of demo video in MATLAB, each image is around 300 KB. We need about 50 frames, which includes effective image elements, to do the subsequent processing. That means we need at least 15 MB memory to store these images. We put the images in SD card formatted in FAT12 then the program transfers the images from SD card to SDRAM from where the program can access the image data for subsequent processing. Since the BRAM of FPGA is very small we use off-chip memory SDRAM (16 MB) to store the images temporarily [1].

The Stack and the heap memories are configured to be present in BRAM along with program code, this would negate the problem of SRDAM holding the images being overwritten by the stack and heap contents.

Also the SD card is formatted in FAT12 system, and 3rd party FAT12 drivers are used to access the files from the SD card.

6. EDK SYSTEM BLOCK DIAGRAM

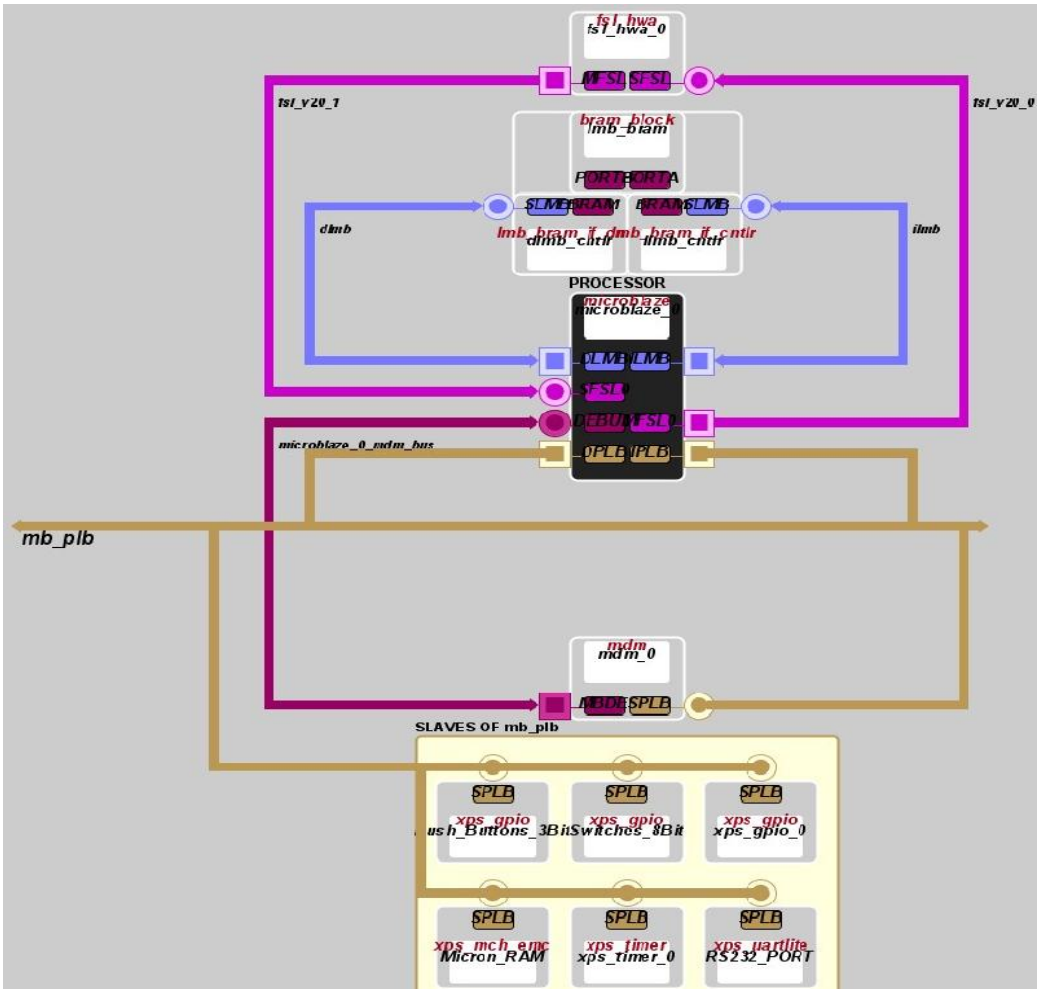

Figure 9: EDK System Block Diagram

As shown in Figure 9 EDK System block diagram contains a MicroBlaze processor core along with external micron SDRAM memory, GPIO interface for SD Card (SD-PMOD), RS 232 interface for display on the HyperTerminal, XPS timer for performance measurements all connected via processor local memory bus, it also contains a hardware accelerator for VGA display connected to MicroBlaze core via Fast Serial Link (FSL) bus.

7. SOFTWARE IMPLEMENTATION

The software running on MicroBlaze core mainly consists of FAT12 file system drivers and a main function which does the following.

1. Reads the image files from SD card and stores it into SDRAM accessible to main program.

2. Scan through the image row by column and perform algorithms for delta frame generation, thresholding, filtering and COG calculation.

3. Encode the COG (ro,co) values and send to the VGA hardware interface in a loop with delay so that we can get a animated trace of the object on the VGA monitor.

The SD card interface (SD-PMOD) is configured as a General Purpose IO in the EDK System from which the main function reads image data byte by byte and dumps it into SDRAM which reduces the latency and setup times of reading from SD card every time we want to access the image data. While writing the files from SD card to SDRAM we take care to ensure that the first frame called the background image is written first which is used as a common factor while calculating the delta frame. After the subsequent processing algorithms we get COG value (x,y) which are encoded in following format- (512*x+y). Because both maximum value of x and y are less than 500, 9 bits are enough for storing each value in hardware. Also each time hardware will get y and x from FSL in FIFO order at same time, which means the first 9 bits stand for y and the another 9 bits indicate x Then we send this encoded value to the VGA custom IP hardware accelerator via FSL bus so that we get a trace on the VGA monitor.

8. HARDWARE IMPLEMENTATION

The hardware part, which mainly used for VGA part, was developed on Digilent Nexys-2 board with Xilinx Spartan-3E FPGA (XC3S1200E, FG320). The system frequency is 50MHz. Our Vga controller communicates with the Microblaze through the Fast Simple Link (FSL) bus. In our design, only a word of data needs to be sent to VGA controller if data is ready in FSL and memory access is not necessary. So BRAM is only used for the storage of the data and instructions of the software.

The VGA port of Digilent Nexys-2 board uses 10 FPGA signals to create a VGA port with 8-bit colour and two standard sync signals (HS-Horizontal Sync, VS-Vertical Sync). There are eight signal levels on the red and green VGA signals and four levels on blue, so it can display 256 different colours in total, one for each unique 8-bit input data. The image on the screen is obtained by

continuous refreshing of the pixel from top-left to bottom-right. Both the synchronization signals and the pixel colour signals should be generated so that we can plot what we want correctly on the screen. In our design, we use VGA controller to generate synchronization signals and VGA display to generate colour signals for each pixel. The block diagram of VGA part is shown in Figure 10.

VGA controller must retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel. Thus, it must generate the HS and VS timing signals and coordinate the pixel data output based on the pixel clock. For a 640x480 display with 60 Hz refresh, the signal timings can be found in the user manual of the board. The horizontal-sync counter (hcount) and vertical counter (vcount) can be used to locate the column and row of a pixel respectively. The pixel data is defined only if the counters correspond to a pixel in the visible area when the blank signal is invalid.

In the VGA part, after the initialization, we draw the background (X and Y axis) and the width of each axis is 5. When 'exist' changes to '1', the first COG value is ready on the FSL bus. Then we store x-coordinate and y-coordinate value and drive 'Rd_ack' high. If hcount and vcount equal to the value of x and y, then we plot this pixel on the screen. One pixel is really small and cannot be seen clearly, so we enlarge the area of each selected pixel by 25 times. As the system frequency is 50 MHz while the refresh frequency of display is only 60 Hz, if we plot one pixel each clock cycle without any latency, we almost see nothing on the screen. Therefore, in software, we use 100 clock cycles to draw the same pixel. Then we can obviously see the trace on the screen. Figure 11 shows the flow chart of VGA trace display.

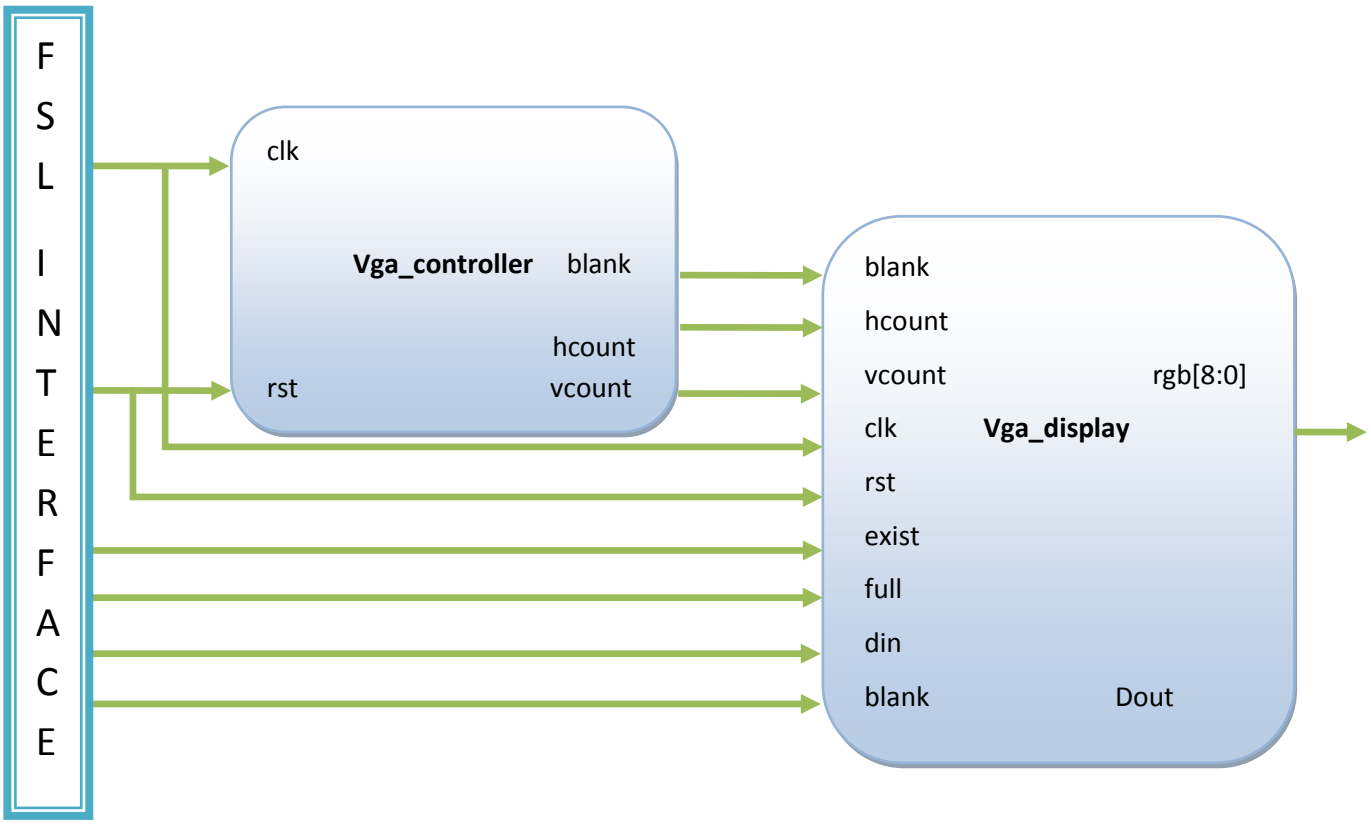

Figure 10 VGA Block diagram

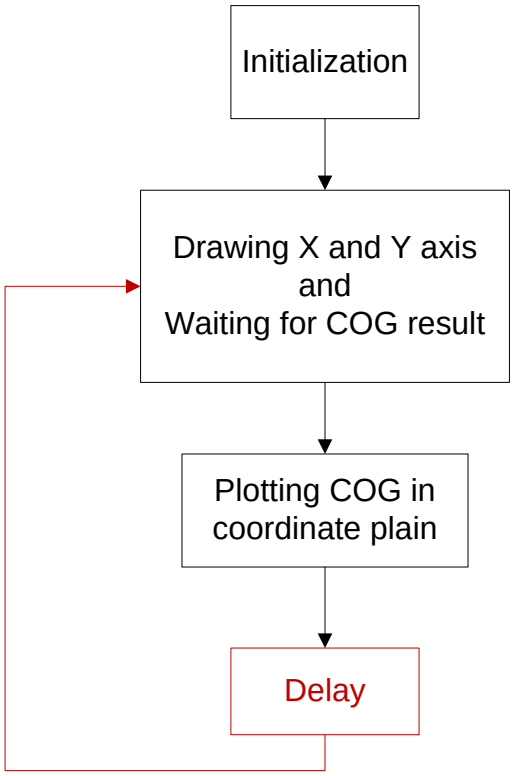

Figure 11 VGA Trace display flow chart (red block is implemented in software)

9. CONCLUSION

The work for this Advanced Embedded project started with the extensive research of papers and articles about some useful algorithm which finds wide application in embedded systems and which can be rapidly prototyped on FPGA. We choose Object tracking which has wide applications in the field of robotics and surveillance, and our work began with need to optimally partition the software and hardware so that we can maximise the performance. In the process of solving deadends and problems we could comprehend how the system and its interfaces work, like for instance FAT12 file system. Also we gained immense experience in developing custom IP hardware. Apart from the familiarity with the EDK tool chain, we could also understand how to configure and use various memories/peripherals on the Digilent Nexys2 board. On the whole we had a good learning experience right from researching about an embedded project and its proposal right down to improve its performance using optimizations at various levels of its development.

10. REFERENCES

[1] MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 12.2
http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/mb/ref_guide.pdf

[2] Digital photo
http://fileadmin.cs.lth.se/cs/Education/EDA385/HT09/student_doc/FinalReports/PhotoFrame.pdf

[3] OBJECT TRACKING ALGORITHM
http://www.cc.gatech.edu/~ksubrama/files/FPGA_Report.pdf

[4] Digilent Nexys Board Reference Manual.
http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

11. APPENDIX

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,048	17,344	11%	
Number of 4 input LUTs	3,044	17,344	17%	
Number of occupied Slices	2,606	8,672	30%	
Number of Slices containing only related logic	2,606	2,606	100%	
Total Number of 4 input LUTs	3,209	17,344	18%	
Number used as logic	2,558			
Number used as a route-thru	165			
Number used for Dual Port RAMs	256			
Number used as Shift registers	230			
Number of bonded IOBs	73	250	29%	
IOB Flip Flops	81			
Number of RAMB16s	16	28	57%	
Number of BUFGMUXs	2	24	8%	
Number of DCMs	1	8	12%	
Number of BSCANs	1	1	100%	
Number of MULT18X18SIOs	3	28	10%	
Average Fanout of Non-Clock Nets	3.45			