Table of Contents

Acknowledgements

Throughout this project there have been a number of people who have helped make it successful. I would like to thank the following:

Chris Sepulveda (XUP Coordinator at *Xilinx*) - for providing a free copy of Xilinx ISE BaseX which was the essential software that all my development was performed on.

Forrest J. Arndt (Regional Sales Manager at *Advanced Circuits*) - for providing a significant discount for manufacturing my circuit board made.

Otto Lee (Online Sales Engineer at *Agilent*) and Barnie Floto (Inside Field Engineer at *Agilent*) - for providing a significant discount to purchasing an Agilent bench meter, which was indispensable during the early analog design phase and useful throughout the entire project.

Peter Alfke (Director, Applications Engineering at *Xilinx*) - for providing several samples of the XCF04S device that stores my FPGA configuration data. The devices were not available from a distributor yet, which meant I would not have been able to get them otherwise.

Finally, I would like to thank my parents for their help throughout the project.

1.0 Summary

1.1 System Overview

Imagine having the ability to perform true real-time object tracking in a small package for low cost. It would be possible to use object tracking in a host of new applications – security cameras that move themselves to targets or cheap robots that can follow objects of interest (such as a car or animal) just to name two. Currently, there is much work done in the field of object tracking, and in fact entire businesses have been built around products that perform object tracking. However most current solutions rely on two things – a fast microprocessor and a lot of RAM.

These systems work by running so fast that it appears to be running in real-time. They can use many different algorithms to do the object tracking, as they have enough time and memory to spare to perform these. What happens when the system needs to be implemented on a small or low-cost platform, where there is no room for a high-speed microprocessor with a considerable amount of RAM as it is too expensive, physically large, or complex to design with?

Enter the *Eagle in a Rack*. Although in its current state this system is rather large, it is just large to make development easier. The real key behind this system is that there is no microprocessor, and there is no external RAM chip anywhere in the system. The entire system is implemented in hardware – specifically on a XC2S200E Field Programmable Gate Array (FPGA) from Xilinx with a chip from Texas Instrument used to digitize the video signal. Most of the on-chip RAM in the system is used to create a list of colours in the image, and no frame is ever held in RAM. Since only a small amount of RAM is needed, it can be easily implemented on the FPGA.

Suddenly it becomes possible to actually make an Application Specific Integrated Circuit (ASIC) that does the object tracking – this means the entire object tracking system can be done on *one* chip! Instead of the microprocessor, the RAM, the support logic, and the FLASH ram to store the program you can now just have it done on one chip (if a ASIC is made), or three chips (the FPGA, configuration memory for FPGA, and video digitizer) if normal off the shelf parts are used.

The system does not need the high frequencies that typically plague these normal object trackers. In truth, the system is designed to be actually running at the speed that pixels are scanned in from the video signal (around 30 MHz). The system is running in *true* real-time, responding to information as it receives it.

1.2 Current Status

The unit as it stands now is approximately 80% completed. All core functions are complete, however there was insufficient time to complete the tracking algorithm in VHDL. For the science fair work was frozen slightly before the fair to keep it in a state where the algorithm can be proven to work.

2.0 Comparison to Current Systems

2.1 Description of Current Solutions

This project is designed to show a twist on achieving object tracking with as little hardware as possible, yet still supporting a robust algorithm and a normal analog video source.

One very interesting system was created at Toshiba Corporation. The details of this system are in a paper from the Proceedings of the 2003 IEEE/RSJ entitled *High-speed Object Tracking in Ordinary Surroundings Based on Temporally Evaluated Optical Flow*[1]. This video tracker is designed with a specific purpose in mind, that purpose is dealing with very fast movement. It uses a special image sensor array that is capable of running at 1000 frames per second (fps), compared to normal video at 30 or 60 fps! To be able to run this fast they had to use two microprocessors, one performs the image capture and the other the image processing.

Using a hardware solution such as the *Eagle in a Rack* it should be able to achieve similar high frame rates. This would take a considerable amount of work though, as the sensor used (a Micron MI-MV13) takes quite a bit of high-speed design to work with that is also incompatible with the *Eagle in a Rack*'s input logic. In addition since the speed is so high it would require more experience in maximising the logic speed with VHDL.

The closest system to the *Eagle in a Rack* that I could find was a system described in a paper entitled *Architecture of an Object-based Tracking System Using Colour Segmentation* [2] published in 1996. This system used a FPGA for some of the high-speed data processing, and also did not use very much RAM. However, they used a computer for some of the data processing, and also required one to define the colour of interest specifically (ie: using numbers). The *Eagle in a Rack* has a more useful human interface, as well as working totally independent of a microprocessor.

2.2 Comparison of Software vs. Hardware Solution

Table 1 provides an overview of some of the differences between the two solutions to the problem of object tracking– software which most current solutions use, and hardware which the *Eagle in a Rack* uses. For each condition, the "winner" is highlighted in green.

Condition	***Software***	***Hardware***
Price	-Requires considerable support hardware, however this hardware can be cheap (ie: surplus computers) -If custom hardware will be needed price will be much greater than hardware solution -Could be cheaper if existing hardware is set-up (ie: there is already a computer)	-Very small amount of hardware needed, but this hardware will be more specialized -Will still be cheaper than the hardware required for the software support in most situations

1 http://www.cs.iastate.edu/~lcmi/stuff/high-speed-tracking.pdf

2 http://www.unibw-muenchen.de/campus/LRT6/staff/bif/bif_a04.htm

Condition	***Software***	***Hardware***
Size	-Requires considerable amount of space, unless a pricey specialized board is made for this	-Very small, even in development configuration (*Eagle in a Rack* is MUCH larger than needed) -Able to integrate entire solution onto one chip if needed
Speed	-Low cost solution will not be able to achieve real-time performance -Using fast processors and specialized software able to achieve real-time performance	-System always runs in real-time
Power Consumption	-using normal computer will be very high, at least 100 watts -using specialized hardware over 10 watts	-very low, under 0.5 watts
Space Readiness	-Commercial computers would not work directly since almost none are radiation hardened -Radiation hardened microprocessors available, but system would be more complex because every chip would need to be space ready, and there would be many chips needed	-FPGA from Xilinx available as radiation hardened, ready for space -Configuration memory from Xilinx also available as radiation hardened -Since ASIC could be created, this could be radiation hardened as needed
Flexibility	-Easy to load new software on computer and debug -Since software is popular way of doing object tracking considerable amount of existing code could be used	-Harder to load new configuration data (depending on system set-up), and harder to debug -Fairly specialized, most work would have to be started from scratch
User Interface	-Can use any interface -Option for complex on-screen displays	-Can use most interfaces -On-screen displays more limited

Table 1 - Comparison of hardware and software solutions

3.0 How to use the System

3.1 The Front Panel

A picture of the system is shown in figure 1. Before beginning operation, you should become familiar with the front panel.

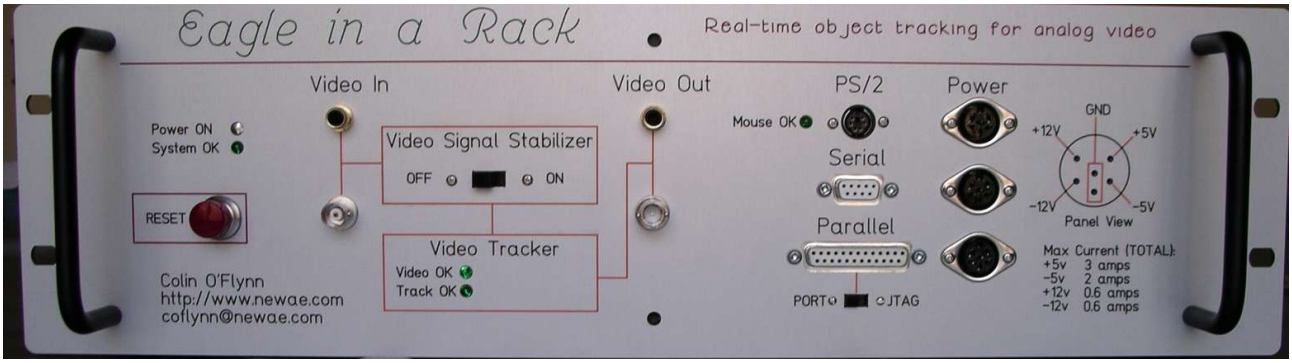


Figure 1 - The Front Panel

The system has a very logical flow to it. When powered on, the blue "Power ON" LED will light up. If the system has passed the built-in test the "System OK" LED will light up. If the System OK light comes on this indicates that the configuration data has been loaded onto the Field Programmable Gate Array (FPGA) without error.

Video input and output is provided in both 50 ohm RCA composite jacks and 50 ohm BNC jacks. Internally both are connected together, so only one can be connected at a time in each bank (ie: do not connect both RCA and BNC video in). If the video is OK then the green "Video OK" light will be lit up. If the video is Macrovision protected the "Video OK" light will flash. If this happens, or the "Video OK" light does not come on at all even when valid video is connected, the "Video Signal Stabilizer" can be tried. Simply move the switch from the OFF position to the ON position, and the incoming video signal will first be run through a stabilizer.

The "Track OK" LED will light up whenever the system is in the track mode (described later).

The video output is provided in both RCA and BNC jacks (as mentioned before). This will be used to display information back to the user. Note that the video is digitized BEFORE the built-in video amplifier and on-screen display generator internally. This means that the video out jack should NOT be used as representative of exactly what the system is seeing, ie: for setting camera aperture or white balance.

When a PS/2 pointing device is connected to the "PS/2" port, the "Mouse OK" LED should be either on or flashing. When the pointing device is idle, the LED will be flashing. When in use the LED will be constantly on.

The serial port is for expansion purposes and not currently used.

The parallel port is either used for expansion or programming. Currently it is only used for programming, and should be connected to a computer using a **straight through DB-25 male to DB-25 male** cable (Belkin F3D111-10 works). Then the device is connected as a Xilinx "Parallel 3" cable, and can be programmed from the iMPACT software that comes free with Xilinx ISE.

Finally there are three power connectors with the pin out of them engraved into the front panel for convenience. Note that if the system appears to be "cycling" on and off, it means there is a short

somewhere and the power supply is turning on and off. Remove all cables from the "Power" section to locate the source of the fault. You may not notice the cycling even depending on the fault. If the system does not appear to be coming on, remove the top cover and see if the neon lamp in the power supply appears to be blinking slightly (will do dim-bright-dim-bright). If this is the case there is a short.

The far left of the panel has a red "RESET" button. This will reset internal state machines in the design. Note that a power-on reset is always performed so this reset button is not normally used.

3.2 Connecting Power

A power jack is present on the back of the unit. This is the input to a universal switching power supply, and can accept 120 volts at 60 Hz. It should be able to accept other voltages as well in the range 100-240 volts at 50-60 Hz (it is rated to work at these), but is totally untested.

Figure 2 - Swing open fuseholder door with screwdriver

The power switch on the back uses normal notation, the | means on (think of it as a binary 1, which is on) and the o means off (think of it as a binary 0, which is off). There are two 24V fans inside the power supply, one of which vents outside. They will be quite noticeable when powered up.

Figure 3 - Pop fuse block up with screwdriver

There are fuses on the power switch and input power module. To replace the fuses remove the power cord, then use a flat screw-driver to open up the fuse holder. Inset the screw-driver blade to the small side of the module closest to the red rectangle (figure 2). Again use the screw-driver to pop the fuse block out (figure 3 and 4). You will find two fuses in this block – one for each side of the power (neutral and hot). Each one is a 1.5 fast-blow 250 VAC fuse, NEVER REPLACE WITH A HIGHER AMPERAGE. Since the power supply has its own built-in short protection, the fuse blowing should be looked at with extreme suspicion. Under almost any fault the fuse should not blow, instead alternate protection methods should have been engaged.

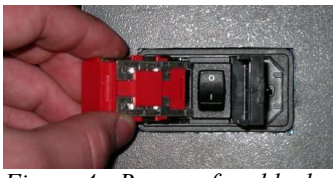


Figure 4 - Remove fuse block all the way

WARNING

The case and front panel are connected to both electrical ground (0 volt reference) and earth ground from the three-pronged plug. DO NOT USE this device where another device with a metal case may be floating or at a high voltage (such as a metal-cased oscilloscope).

If you must do this, you can disconnect the device's metal case from earth ground by removing the grounding lead internally – see the section on the power supply for more information.

3.3 How it will be Used

This will be a quick description of how the system will soon work – however currently it is not quite

finished up to this stage!

When started up and everything connected, moving the mouse will show a cross-hair with a box around it as in figure 5. Scrolling the mouse wheel will increase or decrease the size of this box. To track an object the user must first set the box size so the object roughly fills up the entire box.

Figure 5 - Bounding Box

Then the left mouse button is held down and the object manually tracked. During this stage the unit collects information about the object by dividing the box into four quadrants. The most common colour that is inside the box is decided to be the colour of interest. As well this can be compared to the entire frame, and made sure that the colour selected is fairly distinct. For instance, if there is a red background, then the unit would NOT select red as the colour of interest even if there was a fair amount of red on the object. Then the average ratio of how much of each colour of interest is in each quadrant is calculated.

Once the left mouse button is released, the unit goes into track mode. In this mode it will attempt to keep the ratio of the number of the colour of interest between quadrants the same. This way more than just colour is noted, as shape comes into play. Then the cross-hair will follow the object, and information on the track is outputted over the serial port.

It can be seen that the design is very user-friendly, with a familiar point-and-click interface. In fact it would only be a matter of cost to replace the mouse with a touch-screen that is put over the output TV, so one could just touch the object on the screen to be tracked. This would be useful in a more cramped environment where there is no room for a mouse.

3.4 How it is Currently Used

Currently the unit can demonstrate the ability to find colours of interest. When powered up the unit has the cross-hair with bounding box that is controlled by the mouse. Clicking the mouse's left button will cause the unit to analyse 30 frames of data (1 second). During these 30 frames it will find the most prevalent colour that occurred inside of the bounding box (note: this bounding box may be referred to by the following names: bounding box, tracking box, bounding window, tracking window, target box, target window, capture box, and capture window).

Then every location on the screen that is that exact colour will have a white pixel. As well if the current colour that is most prevalent in the bounding box is the same colour that was decided to be the colour of interest, the box will go white.

This works quite well – in a normal video a specific object can be shown to be picked out with a high degree of accuracy using just the colour method.

4.0 Tracking Algorithm

The key part of this project was to develop a tracking algorithm that can be implemented in hardware and needs little RAM. This was quickly discussed earlier, but this section will detail more specifics about the tracking algorithm.

The basic idea is shown in figure 6, 7, 8, and 9.

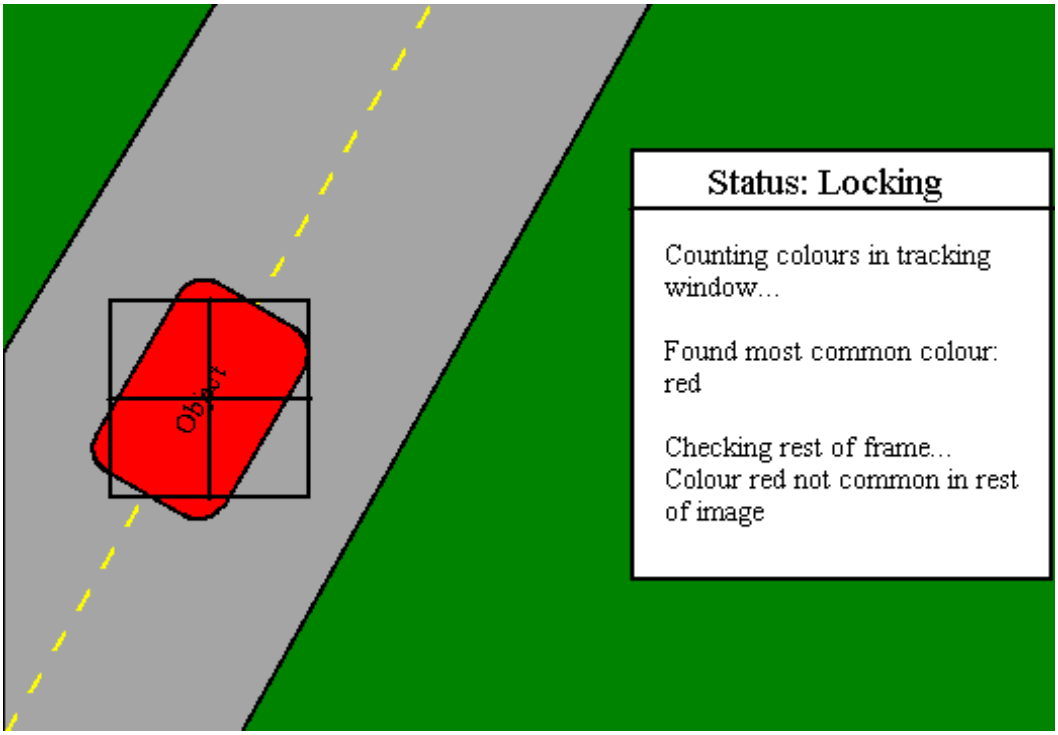


Figure 6 - Step 1: Locking onto Object

The first step is to lock onto the object. When the user holds down the left mouse button, there are two frequency tables started. One will contain the eight most common colours that occur inside the tracking box (the cross-hair with box around it), the other will have the eight most common colours that occur outside the tracking box.

Then the most common colour inside the box would be compared with the top occurring colours outside the box. For example the grey highway would be one of the top colours that occurred in this example outside the tracking box. So grey would be ignored inside the box since it is not unique enough.

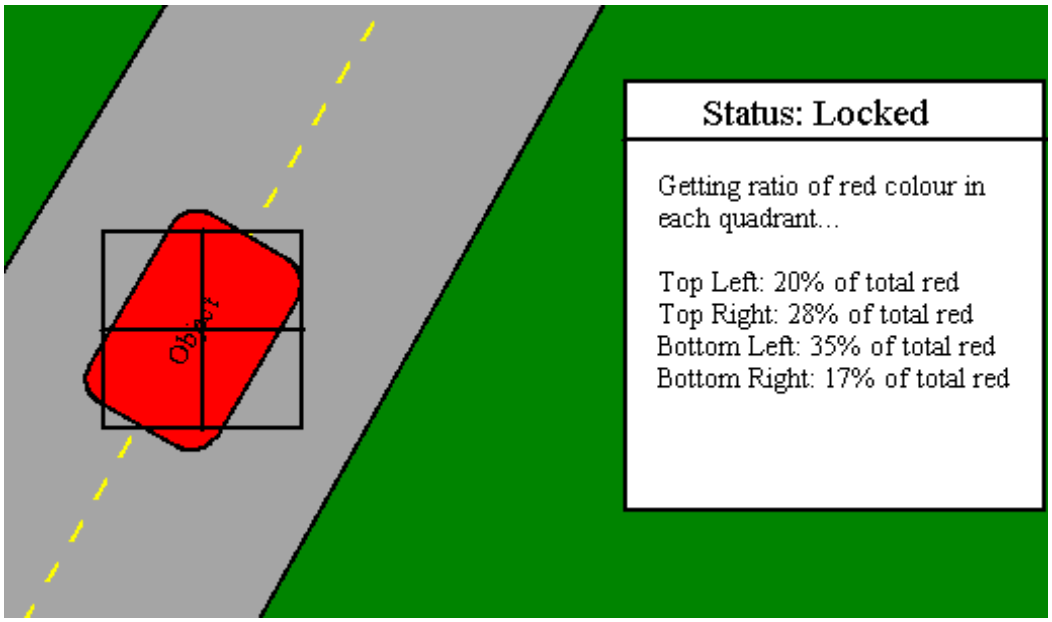


Figure 7 - Step 2: Locked onto object, count ratio

The next step is to split the tracking box into four quadrants. Then the amount of the colour that was identified in step 1 as unique can be counted. Depending on the object shape this amount will differ slightly, so by trying to keep the ratio the same this will keep the tracking box on the object.

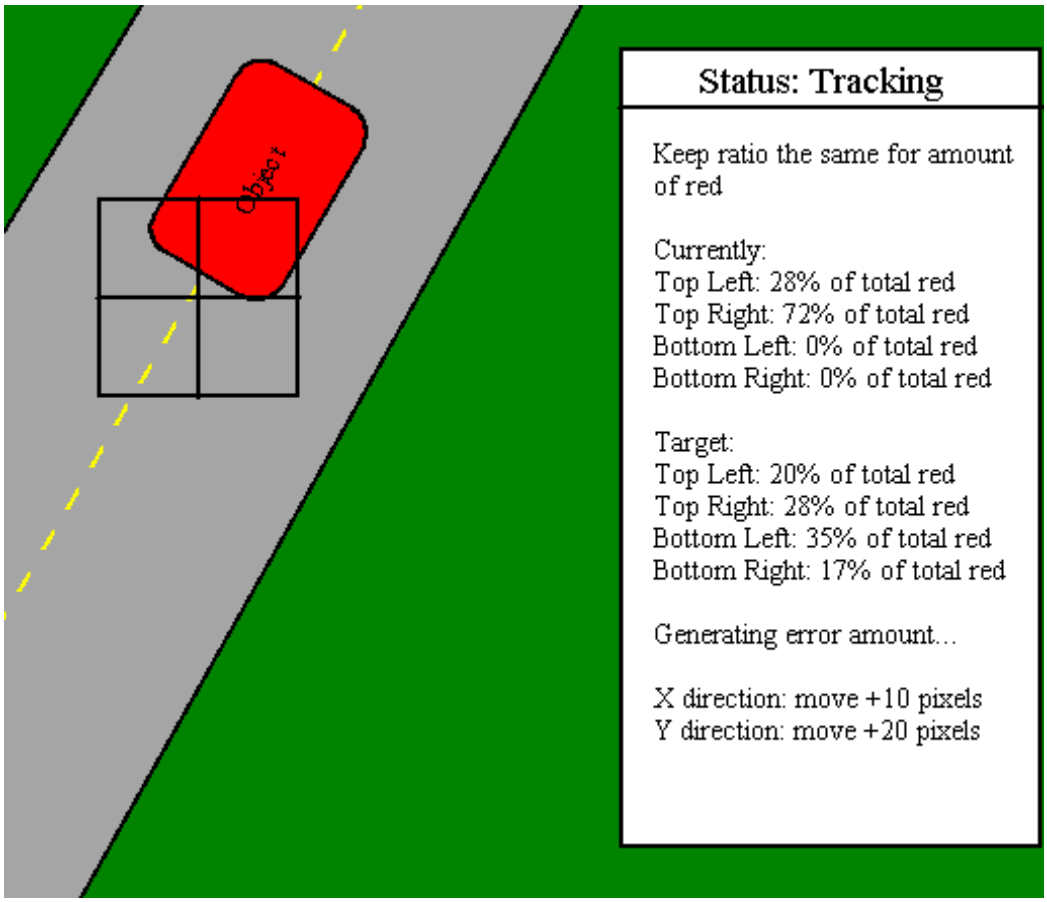


Figure 8 - Step 3:Track object by generating error

Once the ratio is established, the system will loop between counting the current ratio of the colour of interest and moving the bounding box. In this example the car on the highway has move a bit to the right and up. Now the top right quadrant has almost all of the colour of interest. For example, then an error value is generated based on the current pixels of interest inside the tracking box. It needs to move 35% of the pixels into the lower right quadrant, 17% to the lower left quadrant, and 8% more to the upper left quadrant.

Since the system will always be adjusting the location of the box, the exact error amount is not too important – the error just has to be related to how many pixels need to be 'moved'. It does not have to do everything in one frame. For example the jump from figure 8 to figure 9 would not happen in real life. The tracking box would instead by slowly moving its way up the object, as each time it occurred the upper right quadrant would still have too high a percentage of the red colour in it.

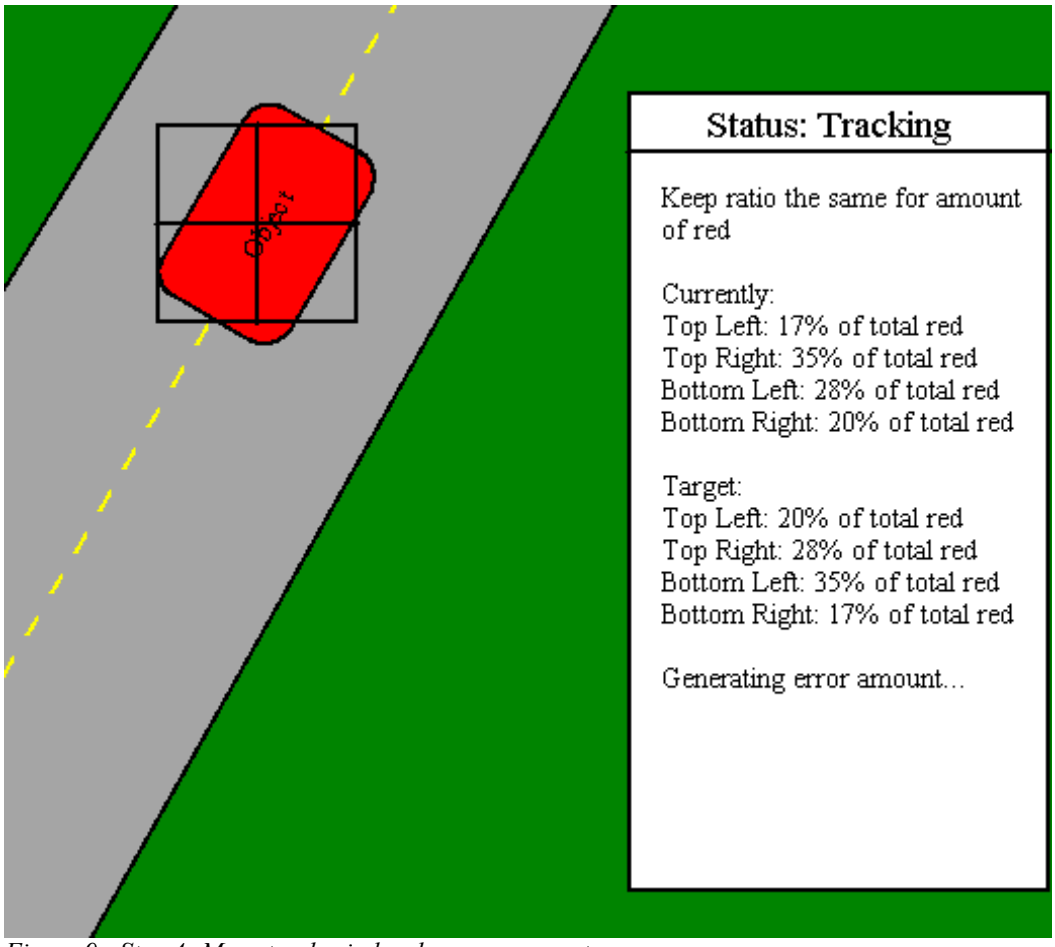


Figure 9 - Step 4: Move track window by error amount

Since the system only moves the tracking box, it would actually be very hard to confuse other objects with the one of interest. For example if a second red car appeared, it would not even be seen unless it drove very close to the target car and part of it appeared inside the tracking box.

This tracking system requires no frame buffer, as it does not need to go back in the frame and compare data. Instead important information is recorded as it appears, for example a running total is kept of what colours have occurred how many times.

5.0 Internal Hardware

This section will detail the internal hardware of the *Eagle in a Rack* as well as the steps that lead to its development. To access or view this hardware, the top cover must be removed. This is achieved by removing the eight screws that secure the top panel in. There are four directly on the top of the unit, and two on each side. Only remove the two screws at the top that are closest to the front panel, there should still be 4 remaining screws on each side panel when the two for the top are removed (see figure 10).

Grab the back of the top cover, and carefully lift it clear.

Note the power supply will have a neon light that is on if the main capacitor has enough energy to light up the neon bulb. The capacitor does have a bleeder circuit, so unless the power is on this light should NOT be on.

Figure 10 - Remove these side screws on both sides

5.1 Hardware Overview

The system is in a 3U size rack enclosure currently. It uses a D2E Board from Digilent Inc, a CSA/UL approved switch-mode power supply, and two custom boards. All the components are shown in figure 11

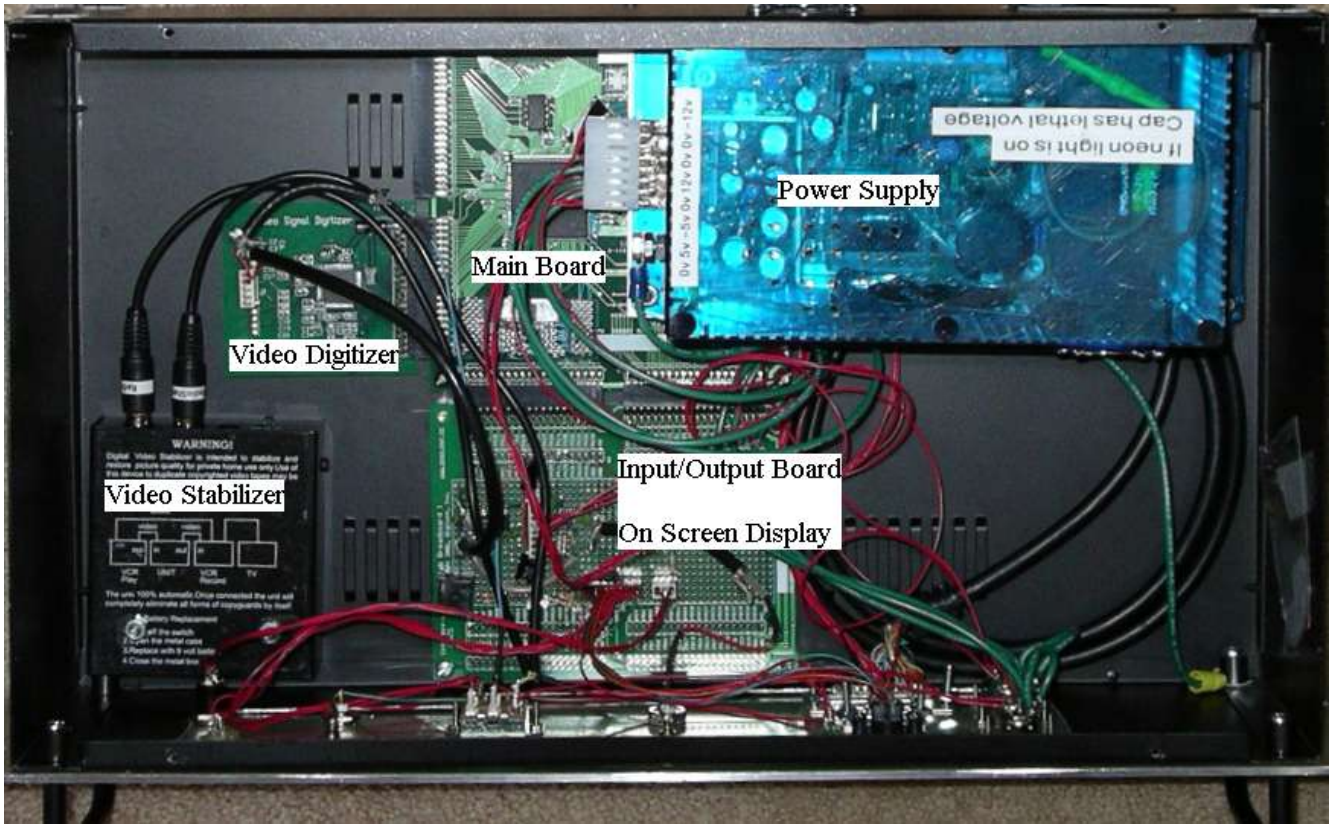


Figure 11 - All boards

The main parts are the D2E Development board with XC2S200E FPGA from Xilinx, the analog video digitizer board, the switch-mode power supply, the video overlay generator and amplifier, and the video stabilizer.

The D2E Development board (will be referred to from now on as the main board, since it has the FPGA

on it) holds the most important part of the project, the FPGA itself. It's function is to provide a stable regulated power supply for the FPGA of 2.5 volts for the core logic in the FPGA and 3.3 volts for the input/output (I/O) pins.

The analog video digitizer board is based on the TVP5145 from Texas Instruments, and can accept any sort of analog video (component, composite, S-Video) in any format (NTSC, PAL). In this project it is configured to use the National Television Standards committee (NTSC) video signal, which is the type used in Canada (and all of North America).

The video overlay generator's job is to generate white pixels on the screen at specific places. This is used to provide feedback to the user in an easy to use way. As well the video signal is amplified to help give the signal more strength.

The video stabilizer is only used when needed. Some video signals will be corrupted, as extra pulses are inserted in the vertical blanking interval. This corrupted video signal may be actually a patented and purposely generated signal, and may be called the Macrovision® system. The video signal stabilizer will remove these extra pulses, which would normally trigger a protection circuit in the video digitizer, thus allowing the device to continue to operate normally.

Finally the switch-mode power supply provides power to the boards, as well as providing power to three power connectors mounted on the front panel.

5.2 Main Board

At the beginning of this project, the idea of using a pure analog solution was tested. Although it seemed possible, it was clear that the complexity would be far too great with no clear advantage over a digital solution. It was decided though that the system would run in true real-time, using the idea of running almost at the same speed as the input signal.

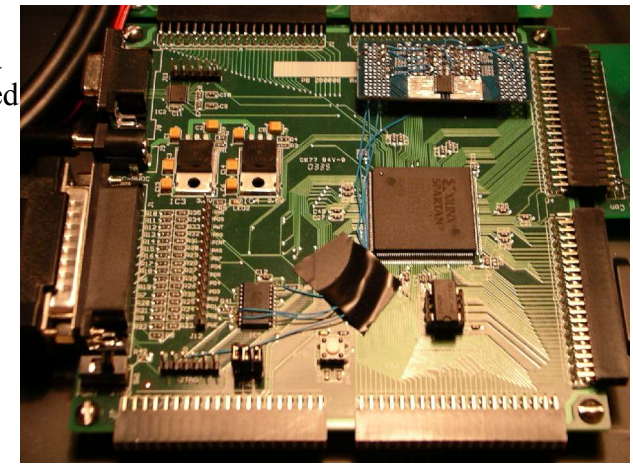


Figure 12 - The Main Board

Next it was decided to use programmable logic, specifically a FPGA. Eventually a low-cost development board was found; the D2E from Digilent Inc shown in figure 12 with some slight modifications performed.

It contains a XC2S200E FPGA from Xilinx Inc. This includes two linear voltage regulators mounted on the board, which take the 5 volt input and convert it to 2.5 volts (for the FPGA's core logic) and 3.3 volts (for the input/output connections). There are four distinct 40-pin female connectors (six total, but two of them are repeats), two of which are used in this project.

Physically the board is a four-layer Printed Circuit Board (PCB) with two full power planes.

The FPGA only has RAM-based configuration storage, which means that after the configuration data is

loaded onto it a power-down will result in loss of configuration data. The board does have an 8-pin socket for a configuration Programmable Read Only Memory (PROM), but these memories are only programmable once. Since this is a development project a reprogrammable memory was needed. An XCF04S chip from Xilinx was used. This chip can configure a FPGA using the same lines as the configuration PROM, which means that little modifications would need to be done to the circuit board. As well it can be programmed via JTAG, which is the same method used to program the FPGA and the board already had a JTAG interface on it. By adding the device into the JTAG chain it could be easily programmed. A SchmartBoard was used to do all this, a SchmartBoard being a SMD development platform. The 8-pin socket was removed, and the SchmartBoard was soldered in its place. As well four additional lines were taken from the JTAG programmer and added onto the SchmartBoard – see figure 13.

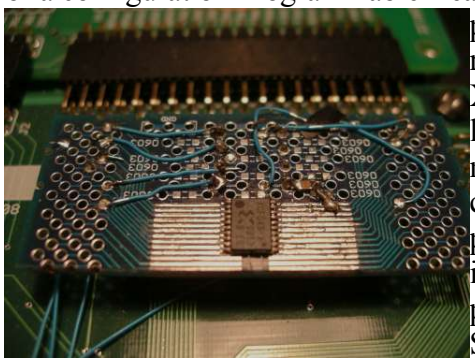


Figure 13 - XCF04S Detail

5.3 Video Digitizer

Digitizing the video signal is one of the most important parts of the project, and also went through a number of revisions. Originally it was considered to only use the luminance information from the video signal. This would only allow the tracker to tell how bright or dim an area of the picture was though, which makes the tracking algorithm almost entirely reliant on some sort of shape detection.

It was then decided that at least some sort of colour (chroma) information was needed. Several quick circuits were tested that attempted to separate the chroma information from the luminance information on the video signal. On the video signal the luminance information and chroma information can be roughly separated using two filters – the luminance information extracted from a low-pass filter, the chroma from a high-pass filter. This does not provide very good results though, and would still require more post-processing!

A number of chips designed to separate chroma from luminance information were researched. However, almost every chip researched was obsolete as most were used in older TVs that there was no longer demand for. At that point this would not even be digitizing the video signal, so a considerable amount more circuitry would be needed!

Finally a chip was found that did it all – the TVP5145 from Texas Instruments. Not only did it perform the chroma and luminance separation, but it even digitized the signal. Most importantly though – the chip was available. There were a number of other chips found that were either vapourware (don't exist in real life), obsolete, or only available in quantities of thousands – however the TVP5145 was available in quantities of one from Digikey at a reasonable cost.

In the video digitizer board the main chip (the TVP5145) is in a Thin Quad Flat Pack (TQFP) with 80 pins, so it was necessary to make a circuit board for it since prototyping would be difficult. As well the chip has fairly high requirements for frequencies (internal clocks running at least 30 MHz) which makes careful layout of the circuit board a must.

This board gives a number of output signals to the FPGA. These signals are the pixel clock (one clock pulse for every pixel), the vertical synchronization pulse (for every new frame), the horizontal synchronization pulse (for every new line), the chroma (or colour) information, and the luminance information.

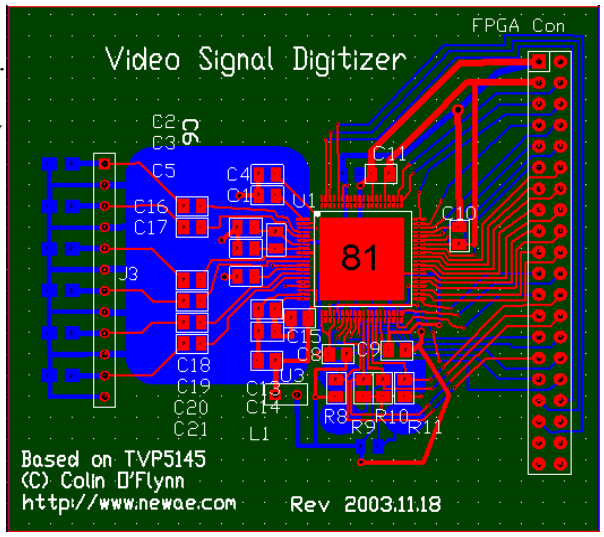


Figure 14 - Video digitizer PCB

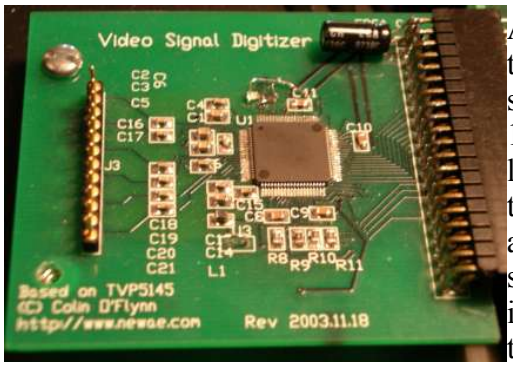


Figure 15 - Video digitizer real PCB

An overview of the PCB is shown in figure 14 – the bottom layer is blue, the top layer is red, and the silkscreen layer is white. The analog portion of the circuit includes a ground plane to help reduce noise. This revision of the circuit board includes three errors. The first being that the connector needs the power lines moved to the other side of the connector (ie: pin 39 and 40 instead of pin 1 and 2), the second being that the ground and 3.3 volt rail are shorted together, and the final error is that there is no pull-up resistors on the I^2C lines. All these errors were easily fixed with an Xacto knife, some solder, and wire.

The final board is shown in figure 15.

5.4 Input/Output Board and On Screen Display Generator

There is one general purpose board illustrated in figure 16 that has a number of functions, including the connector for the PS/2 mouse, the connections to the LEDs on the front panel, and the on-screen display generator.

The original plan was to use a touch-screen that is put over the output LCD. By doing this the interface would be a touch based interface, which would be the easiest and most practical to use. However touch-screens were found to be prohibitively expensive, so this option was turned down. Instead a mouse was used – and eventually the mouse could be replaced with a touch-screen.

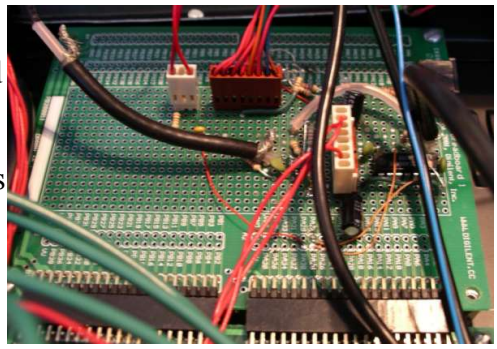
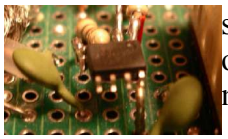


Figure 16 - Input/Output board

The on-screen display generator uses a 4066 CMOS analog switch to generate white pixels. Normally the switch will select the "video in" as the video output. Whenever a white pixel must be generated, the switch selects a voltage divider that is set to a white-level. Then after the location of the white pixel is passed, it is switched back to the "video in" source.

A video amplifier is added before the signal is passed to the video out connector, since otherwise the



system's performance will be affected by what sort of device is connected to the video out jack. The amplifier helps to buffer the video out jack, and the amplifier chip is shown mounted in figure 17.

Figure 17

5.5 Video Signal Stabilizer

The video signal stabilizer is a simple commercial device intended to clean up a video signal if it has been degraded. When the front-panel switch selects the device, the video input signal is first routed through it. Otherwise the video bypasses this device and goes directly to the video digitizer. It can be seen the unit is simply bolted to the chassis in figure 18.

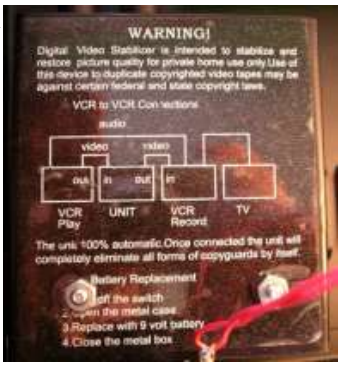


Figure 18 - Video Stabilizer

5.6 Power Supply

The power supply is a commercial UL approved device. It was deemed far too dangerous and complex to design a specialized power supply, especially when a good power supply can be bought for $15 (surplus) from Sayal Electronics (located in Burlington, Ontario).

This power supply is specifically a SRW-45-4001 from Integrated Power Designs. This is a switchmode power supply that can deliver 45 watts of output power in four voltages, +5 volts, -5 volts, +12 volts, and – 12 volts. It will automatically shut down at 110% overload, and cycle power until the fault is removed. It is mounted in a translucent plastic case from Hammond Mfg.

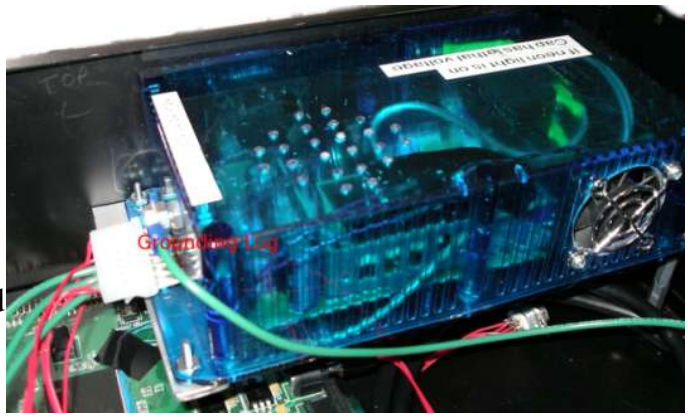


Figure 19 - Power Supply

The UL file number is E137708, and is recognized in Canada as well (CAN/CSA-C22.2 No. 950-M95).

> **WARNING**
>
> The case and front panel are connected to both electrical ground (0 volt reference) and earth ground from the three-pronged plug. DO NOT USE this device where another device with a metal case may be floating or at a high voltage (such as a metal-cased oscilloscope).
>
> If you must do this, you can disconnect the device's metal case from earth ground by removing the grounding lead internally – read on for more information. THIS IS NOT RECOMMENED.

To disconnect the earth ground look on the side of the power supply with the connection to the low-voltage DC. You will see a bolt protruding from the case, with three nuts on it. Remove the outermost nut, and slip the grounding cable off (the other side of the grounding cable goes to one of the bolts that

holds the handles on). Be sure to insulate the grounding terminal on the power supply to prevent accidental contact. Be sure to confirm with a multimeter's that the electrical ground (such as the screws on the power socket) are no longer connected to the ground prong on the plug. DISCONNECTING GROUND CAN BE EXTREAMLY DANGEROUS, USE CAUTION!!

6.0 Overall VHDL Description

The actual hardware that is on the FPGA is designed to be very structural, in that there are many high-level modules that are pulled together to create the entire system. The VHDL code is presented in the appendix, as well as schematics of the synthesized logic.

Currently there are effectively two tasks the system does – display white pixels where a colour match has occurred, and to turn the entire tracking box white if the most common colour in it is the same as the target colour of the object.

These two tasks are shown from a behavioural standpoint in figure 20 and figure 21.

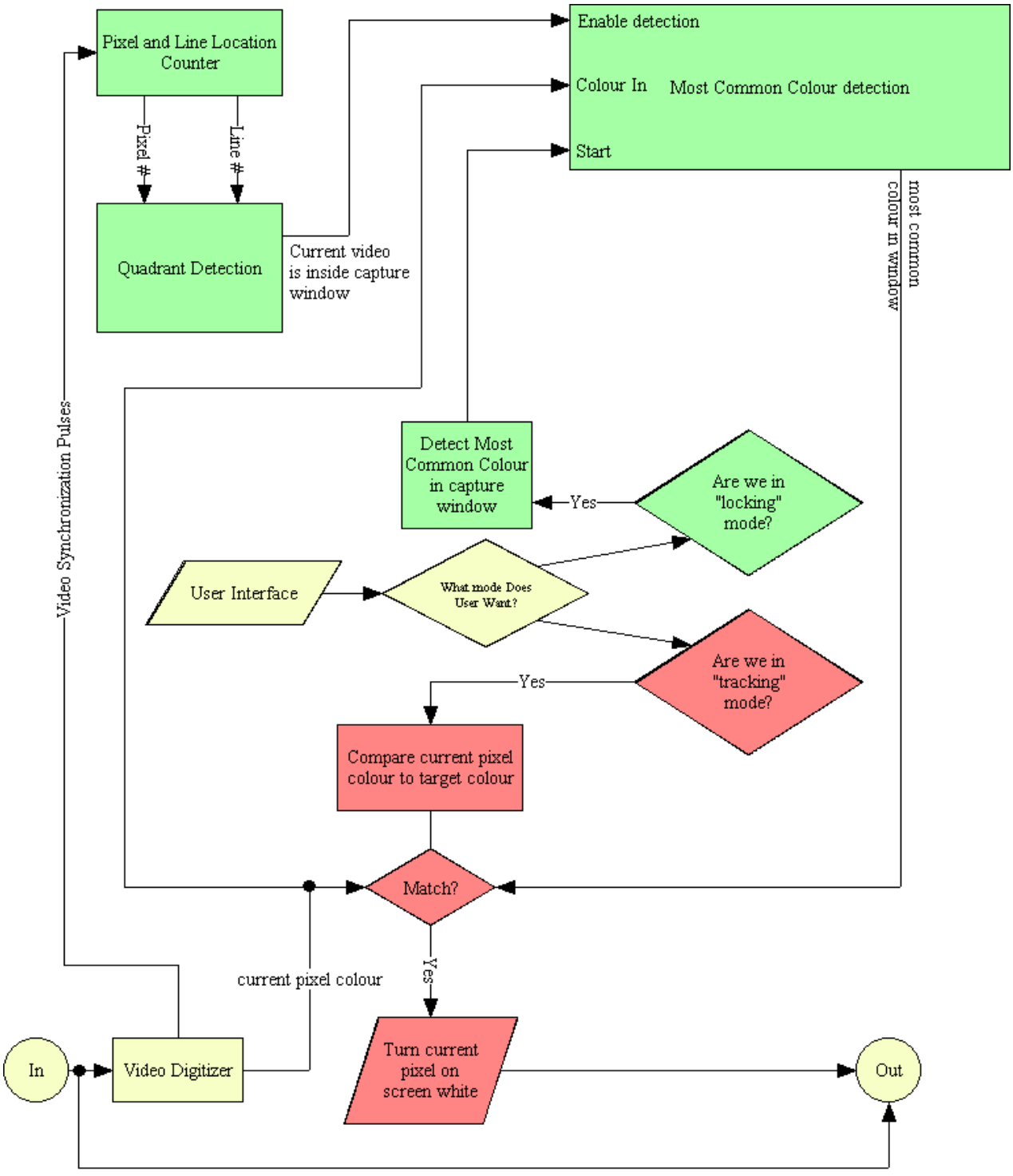


Figure 20 - Behavioural description of drawing a white pixel on the screen wherever a colour match occurs. Yellow = used in lock and track mode, green = used in lock mode, red = used in track mode

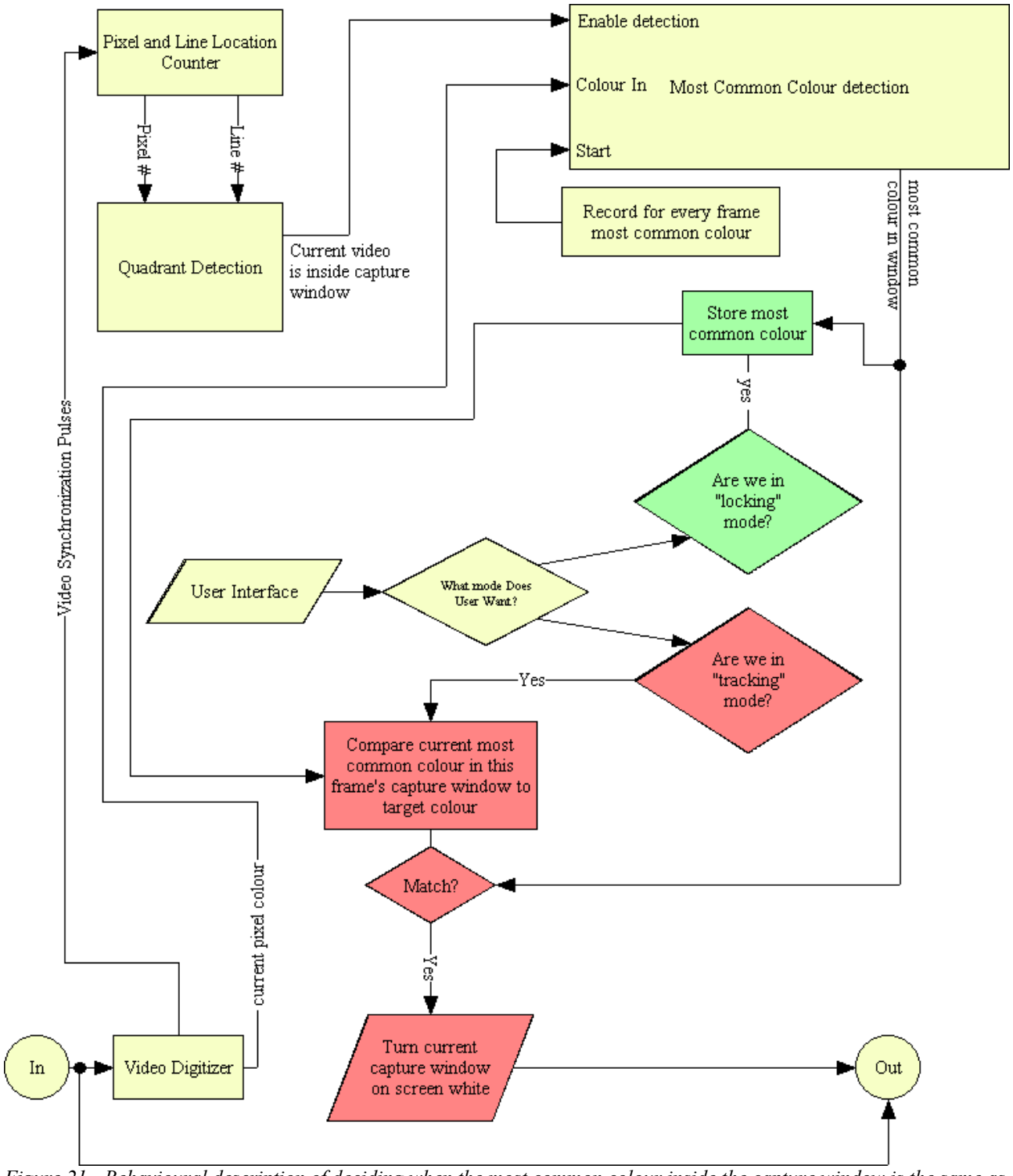


Figure 21 - Behavioural description of deciding when the most common colour inside the capture window is the same as the target colour. Yellow = used in lock and track mode, green = used in lock mode, red = used in track mode

7.0 VHDL Core Descriptions

The main VHDL file used is called `video_chip.vhd`. It relies on the other VHDL modules that will be described in this section. It also relies on the file `video_utils.vhd`, which are just several modules that were removed from the file `video_chip.vhd` to make it less crowded.

Please see the appendix for entire VHDL listings, as well as synthesis reports and schematics.

7.1 Frequency Table Counter Core

This module will keep track of numbers being inputted, and will output when requested a list of what numbers have occurred the most, and how many times each one occurred. It is the most important core in this system, as the object tracking algorithm depends heavily on it.

File: `core_freq_table_counter.vhd`
Dependencies: `freqtable_ramcount.xco`

VHDL Component Declaration

```
package freq_table is
component frequency_table is
    Port    (
            reset_in          : in std_logic;
            reset_out         : out std_logic;

            dataclk_in        : in std_logic;
            dbldataclk_in     : in std_logic;
            data_in           : in std_logic_vector(7 downto 0);
            data_rdy_in       : in std_logic;
            record_in         : in std_logic;

            lowerlimit_in     : in std_logic_vector(7 downto 0);
            upperlimit_in     : in std_logic_vector(7 downto 0);

            data_rdy_out      : out std_logic;
            sort_addr_in      : in std_logic_vector(2 downto 0);

            data_out          : out std_logic_vector(7 downto 0);
            datacount_out     : out std_logic_vector(11 downto 0)
            );
end component frequency_table;
end package freq_table;
```

Interface Description

The interface is described in table 2, including all the pins.

Name	***Direction***	***Description***
reset_in	In – 1 bit	Pulling this line high will initiate a reset of the entire frequency table and other internal logic. Every time the RAM needs to be cleared (ie: you are done with the frequency table and want to make a new one) you need to pull this line high and **must remain high until reset_out goes high.** Also note that the reset line must go from low to high to reset properly, as the reset logic uses the reset_in line as an active low reset.
reset_out	Out – 1 bit	This line will go high when the reset is complete (around 256 cycles of dbldataclk_in)
dataclk_in	In – 1 bit	This is the normal clock. The data should be set-up on the falling edge of this clock (it must not actually be ready yet, as it is not used for ¼ a clock cycle from the falling edge)
dbldataclk_in	In – 1 bit	This clock runs at exactly twice the rate of dataclk_in. The first falling edge **must** be synchronized to the falling edge of dataclk_in. Data at data_in **must** be ready by the first rising edge of dbldataclk_in
data_in	In – 8 bits	The input data, If the system is in the record mode and data_rdy_in is active ('1') then every falling clock on dataclk_in will result in the count for the number of times 'data' has appeared to increase by one
data_rdy_in	In – 1 bit	This line should be pulled high whenever data that should be used is present at data_in
record_in	In – 1 bit	when active ('1') module will record the data that is coming though on falling edge module will start to generate a list of the data that has occurred the most and output it
lowerlimit_in	In – 8 bits	Any data that is lower than this will be ignored. Useful for example blanking out the lower extreme of data if needed. This must stay constant from when record_in goes low until the data is read out. It is recommended this line is just connected to a constant value.
upperlimit_in	In – 8 bits	Any data that is higher than this will be ignored. Useful for example blanking out the higher extreme of data if needed. This must stay constant from when record_in goes low until the data is read out. It is recommended this line is just connected to a constant value.
data_rdy_out	Out – 1 bit	This line will go high some time after record_in has gone low. When this line goes high it means data is ready to be read out.
sort_addr_in	In – 3 bit	This line selects which data to read out. Address "000" is the data that has occurred the most, address "001" is data that has occurred the 2nd most, all the way down to "111" which is the data that has occurred 8th most. This line is internally connected to a multiplexor, so after the address is put in there is no need for any clocking, the data will be ready on the outputs after the propagation delay.

Name	***Direction***	***Description***
data_out	Out – 8 bits	When sort_addr_in is set as described, this line will have the data of interest. For example if "11011011" has occurred the most at 1568 times, and sort_addr_in is set to "000", the data_out will have "11011011" on it.
datacount_out	Out – 12 bits	This outputs how many times the data on data_out occurred in the data set In the above example where "11011011" occurred 1568 times, these bits would have "0000011000100000" (1568 dec).

Table 2 - frequency table interface

The timing for the clock is shown in figure 22, as well as showing the data_in. Note that the data_rdy_in line does not need to be pulsed at all, it can just be held high.

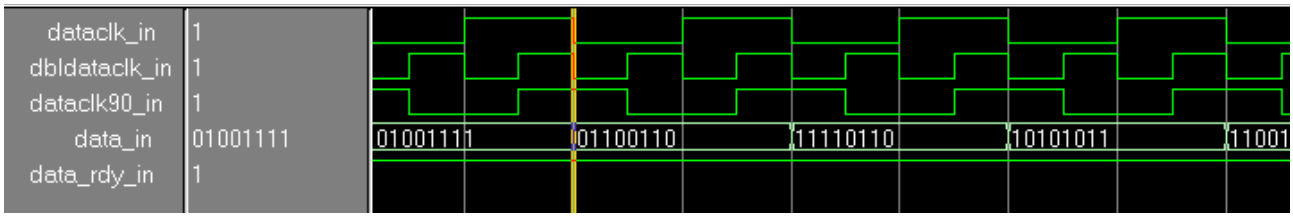


Figure 22 - Basic data timing for clocks

How it Works

The main idea behind this system is a RAM core – it is 12 bits wide, and has 256 words (12-bit words). The address is used to store which number has occurred, and the data at that address is how many times that number has been 'hit'.

So each time new data is clocked in, the current data at the address is incremented by one.

When it comes time to sort the data, the system goes through each address. It makes a list of the eight most occurring colours, which has the colour (ie: the address of the RAM) and the number of times it has occurred (the data at the address). It does this by starting at address 0 of the RAM, and gets the data (number of times it has occurred). It then compares that with the list of most common colours, starting at the 1st position, and if the current colour has occurred more times than the current 'record-holder' the new data is inserted in its place.

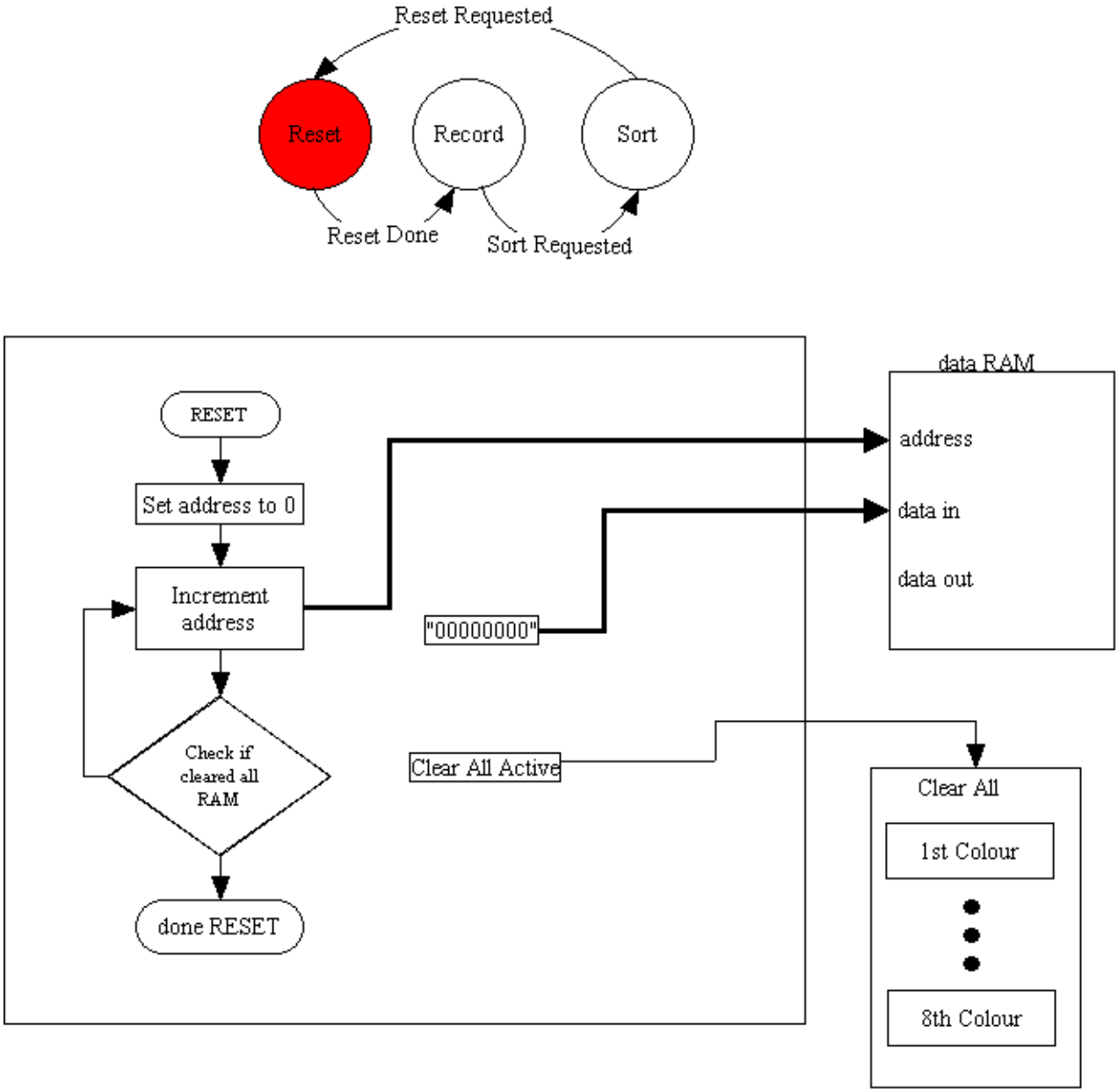


Figure 23 - Reset State

Figures 23, 24, and 25 show how this system works. Each figure is a different state – the main logic is all in one large state machine.

The reset state is shown in figure 23, and this state clears all the RAM or data. Since it must clear every memory location, this takes about 256 cycles of the ram clock.

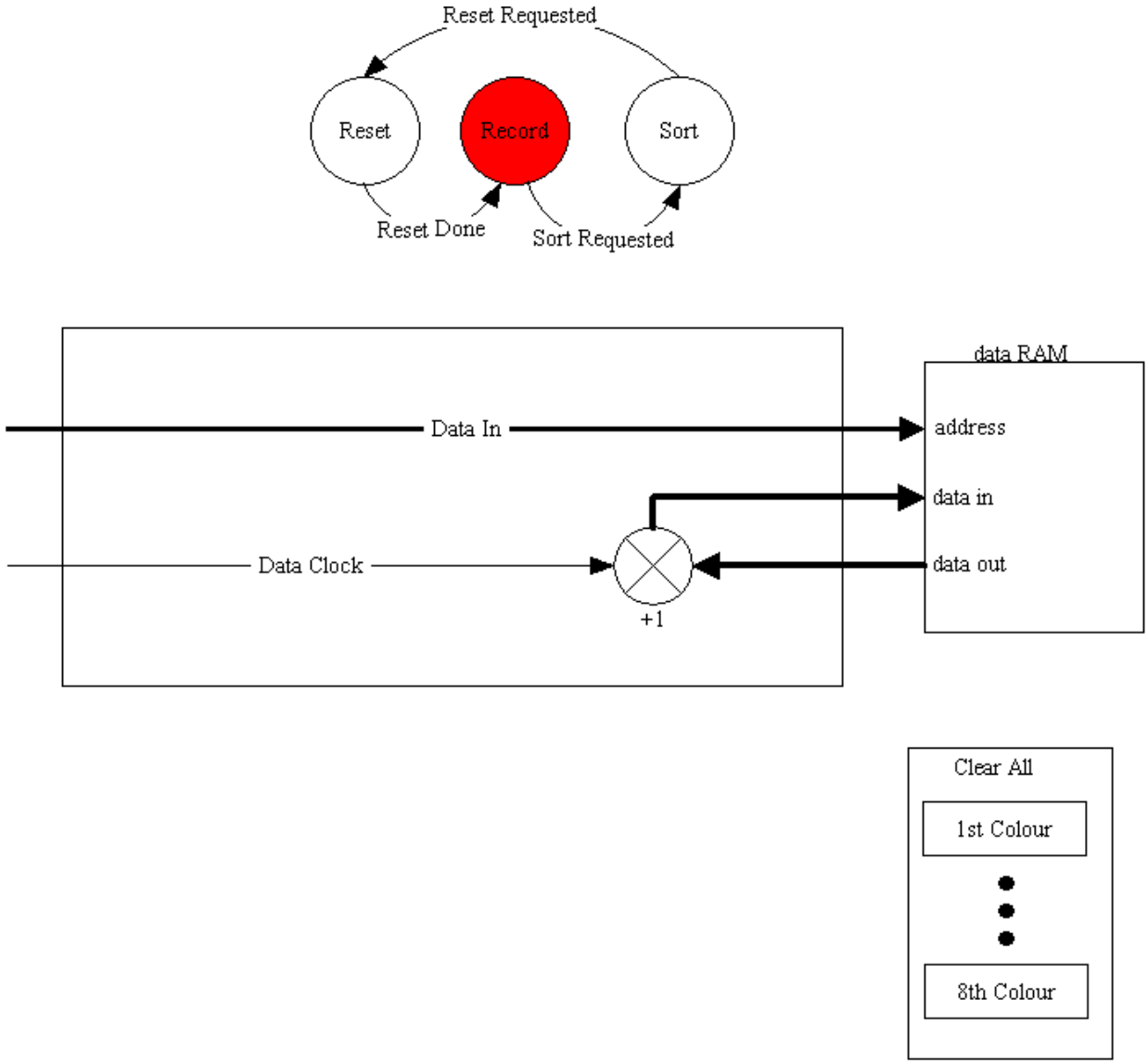
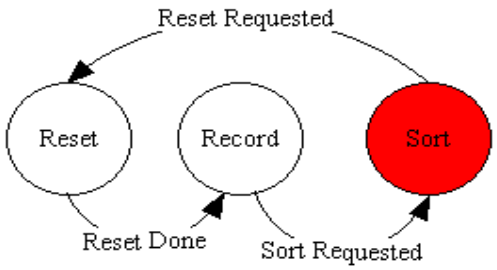
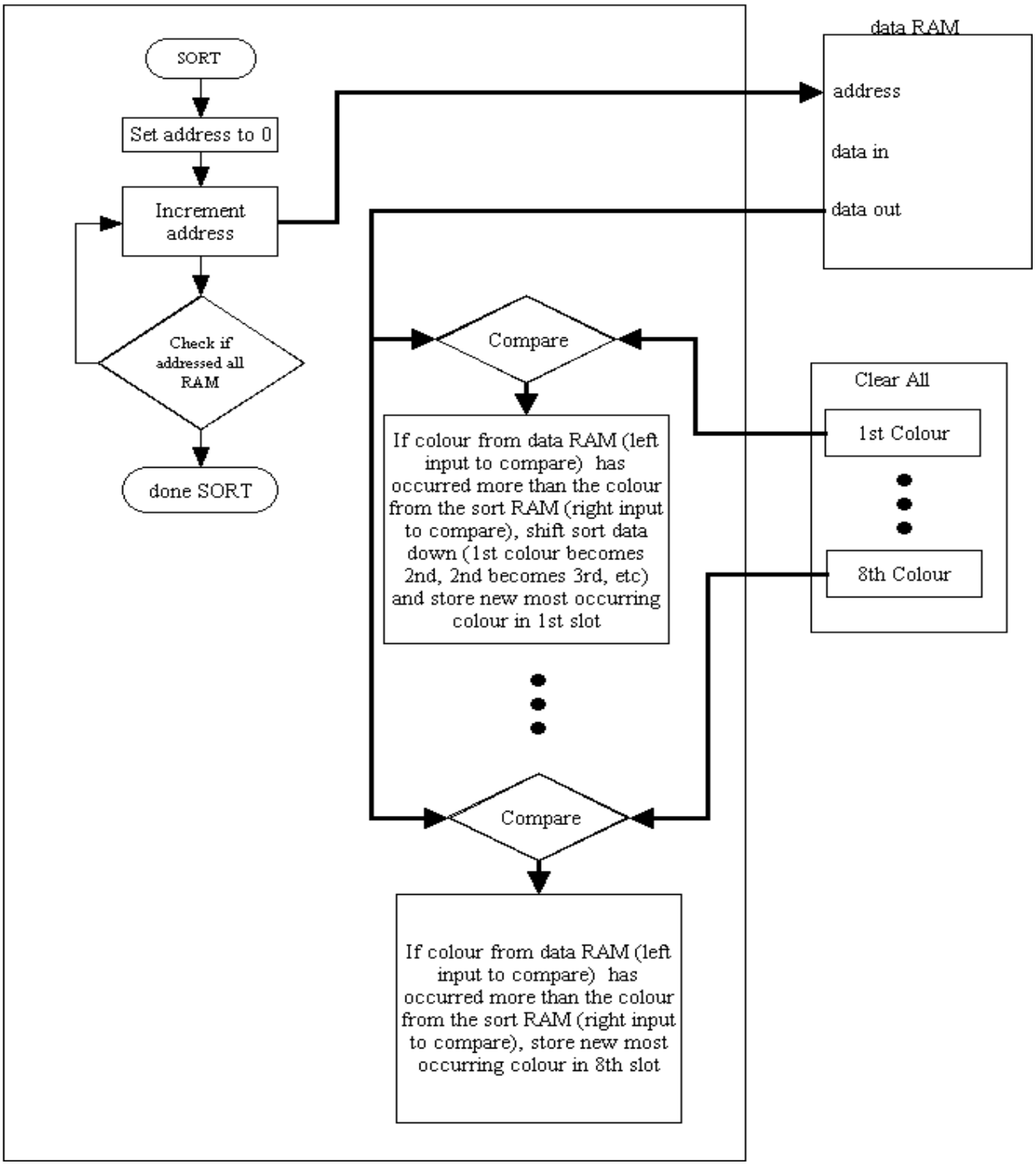


Figure 24 - Record State

When the system is ready to start recording information, it goes into the record state shown in figure 24. Here it is simply tallying up how many times each data input (which would be a specific colour) has occurred.

Finally figure 25 shows the sort state. It reads out how many times each colour has occurred, and compares it to the record-holders. If one of the colours has occurred more times than one of the record-holders, it inserts the new colour into the list of most common colours.


Reset Requested
Reset
Record
Sort
Reset Done
Sort Requested


SORT
Set address to 0
Increment address
Check if addressed all RAM
done SORT
data RAM
address
data in
data out
Compare
If colour from data RAM (left input to compare) has occurred more than the colour from the sort RAM (right input to compare), shift sort data down (1st colour becomes 2nd, 2nd becomes 3rd, etc) and store new most occurring colour in 1st slot
Clear All
1st Colour
8th Colour
Compare
If colour from data RAM (left input to compare) has occurred more than the colour from the sort RAM (right input to compare), store new most occurring colour in 8th slot

Test System

For testing purposes a testbench of this module was created called `core_freq_table_counter_testbench.vhd`. This testbench takes an input file called `freq_count_test_vectors.in`, and creates an output called `freq_count_test_vectors.out`. The input file must contain a list of any amount of 8-bit test vectors, in the format:

```
01010101
10101011
11111111
```

The output file will contain a list having the input number that occurred the most, and the number of times it occurred. As well it will contain the numbers that occurred the 2nd most to 8th most:

```
00001110            <-- This test vector occurred the most, specifically it occurred
000000000110        <-- 6 times
01011111
000000000110
00010001
000000000101
00101110
000000000101
01011101
000000000101
01100100
000000000101
01100101
000000000101
10110110            <-- This test vector occurred the 8th most, specifically it occurred
000000000101        <-- 5 times
```

To help in creating these test vectors, a simple ANSI C program was created which source code can be found below in listing 1. It can be compiled in gcc by using the command:

```
$ gcc generate_vectors.c -o gen_vectors
```

This program will write all the test vectors to STDOUT, so they can be redirected to a file by running the program as such:

```
$ ./gen_vectors > freq_count_test_vectors.in
```

This would create a file called test_vectors.in with 17500 test vectors in it. These vectors can then be tested in the test bench by simulating the project.

IMPORTANT NOTE

To ensure correct simulation you must add the following files to your project:

`freqtable_ramcomp.vhd`

`core_freq_table_counter.vhd`

`classio.vhd`

`core_freq_table_counter_testbench.vhd`

The simulator will also need the Xilinx Core libraries to simulate the RAM, for this reason it is recommended to use the Xilinx Modelsim version, available for free from Xilinx's website.

```
//Test vector generation file
//Colin O'Flynn
#include <stdio.h>
#include <stdlib.h>

//number of test vectors to generate
#define NUM_VECTORS 17500

      //Put lots of 8-bit std_logic_vector types to stdout
int   main
      (
      void
      )
      {
      int random_number;
      int i;
      int j;

      j = 0;
      while (j != NUM_VECTORS)
      {
            random_number = rand();
            i = 0;
            while (i < 8)
                  {
                  printf("%i", ((random_number >> i) & 1));
                  i++;
                  }
            printf("\n");
            j++;
      }
      return 1;
      }
```

Listing 1 - generate_vectors.c file contents

7.2 I²C TVP5145 Controller Core

This module is designed to connect to the TVP5145 chip from Texas Instruments via I2C, and will enable the chip, and then continually get the information about the status of the video signal.

File: `i2c_control.vhd`
Dependencies: `i2c.vhd` (NOTE: This file from opencores.org)

VHDL Component Declaration

```
package i2c_control is
component controlchip is
    Port    (
            reset_in     : in  std_logic;
            clk_in       : in  std_logic;

            Lock_out   : out std_logic;
            Macro_out  : out std_logic;

            SCL_inout  : inout std_logic;
            SDA_inout  : inout std_logic;
            SAS_out    : out std_logic;
            );
end component controlchip;
end package i2c_control;
```

Interface Description

The interface is described in table 3, including all the pins.

Name	***Direction***	***Description***
reset_in	In – 1 bit	This is the active high reset. Must be driven high at least once to reset the internal state machine.
clk_in	In – 1 bit	This is the 50 MHz clock input to the I²C bus.
lock_out	Out – 1 bit	This output will go high whenever there is a lock on the video signal. If this output is low then assume there is no valid video present.
macro_out	Out – 1 bit	This output will go high when the video signal is protected by the Macrovision ® content control.
SCL_inout	Inout – 1 bit	This is the I²C clock line, and should be connected to the SCL pin on the TVP5145 (needs external pullup of 2.2 Kohm)
SDA_inout	Inout – 1 bit	This is the I²C data line, and should be connected to the SDA pin on the TVP5145 (needs external pullup of 2.2 Kohm)
SAS_out	Out – 1 bit	This is the I²C Serial Address Select line, and should be connected to the SAS pin on the TVP5145 (used to select address the device communicates on).

Table 3 - I²C Control for TVP5145 Interface

7.3 Mouse Controller Core

This module will connect to a PS/2 mouse and provides pointer information as well as the button state.

File: `mouse_interface.vhd`
Dependencies: `ps2.vhd` (NOTE: This file from opencores.org)

VHDL Component Declaration

```
package PS2_mouse is
component mouse is
   Port     (
            clk_i       : in std_logic;
            rst_i       : in std_logic;
            nrst_i      : in std_logic;
            slow_clk    : in  std_logic;

            x_loc_o     : out std_logic_vector(9 downto 0);
            y_loc_o     : out std_logic_vector(9 downto 0);

            rightbut_o  : out std_logic;
            leftbut_o   : out std_logic;

            mouseok_o   : out std_logic;

            ps2_data_io : inout std_logic;
            ps2_clk_io  : inout std_logic
            );
end component mouse;
end package PS2_mouse;
```

Interface Description

The interface is described in table 4, including all the pins.

Name	***Direction***	***Description***
clk_i	In - 1 bit	The input clock, should be 50 MHz. Other values can be used but the internal constant in the PS2.vhd file would need to be changed.
rst_i	In - 1 bit	Active high reset, needs to be pulled high at least once to initialize internal state machines
nrst_i	In - 1 bit	Active low reset, should be connected to NOT rst_i
slow_clk	In - 1 bit	The slow clock, should be 3.125 MHz (note: 1/16 of clk_i)
x_loc_o	Out - 10 bits	This is the X or horizontal location of the “pointer” on the screen. This value will range from 0 (furthest left) to 1023 (furthest right).
y_loc_o	Out - 10 bits	This is the Y or vertical location of the “pointer” on the screen. This value will range from 0 (top) to 1023 (bottom).
rightbut_o	Out - 1 bit	The active high state of the right button (when '1' means right button is being pushed)

Name	*Direction*	*Description*
leftbut_o	Out - 1 bit	The active high state of the left button (when '1' means left button is being pushed)
mouseok_o	Out – 1 bit	This output will go high whenever a mouse is attached and detected. Note that if the mouse is idle this output will oscillate at approximately 0.5 Hz.
ps2_data_io	Inout – 1 bit	This is the data line for the mouse. It is open collector (will never be driven high), so must be connected to the data pin of the mouse and also to a pullup resistor (around 10 Kohm)
ps2_clk_io	Inout – 1 bit	This is the clock line for the mouse. It is open collector (will never be driven high), so must be connected to the clock pin of the mouse and also to a pullup resistor (around 10 Kohm)

Table 4 - mouse control interface

The pin-out for the mouse connector is shown in figure 26. The pins of interest are:
1 : Data lines
3 : Ground
4 : +5 volts
5: Clock

Figure 26 - PS/2 Pinout (View of mouse cable)

8.0 Conclusion

This project's goal is to implement an entire object tracker without using very much memory or a fast microprocessor. The project has been successful so far, as the core functions are tested to work exceedingly well. The project is capable of selecting the most common colour, and locating areas where this colour occurs the most again. All this is done without every storing a frame of video in memory, and this is done in true real-time.