DEVELOPMENT OF A REAL-TIME VISION SYSTEM

FOR AN AUTONOMOUS MODEL AIRPLANE

Danko Antolovic

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Master of Science
in the Department of Computer Science
Indiana University

October, 2001

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Science.

_____

Prof. Steven D. Johnson, Ph.D.

_____

Thesis committee

Prof. Florin Cutzu, Ph.D.

_____

Prof. Michael E. Gasser, Ph.D.

September 21, 2001.

ACKNOWLEDGMENTS

This thesis is a description of a hardware/software system constructed in a series of Y790 independent study courses in the Department of Computer Science at Indiana University. The work was done under the supervision of Professor Steven D. Johnson, to whom I am grateful for his support, his interest in the progress of the project, and for his insightful and critical comments. Professor Johnson constructed the camera gimbal currently in use.

I am happy to have had the help of Mr. Bryce Himebaugh, engineer and pilot extraordinaire. Beside constructing the A/D converter and the servo circuit, Bryce has shared his knowledge and skill through many helpful and enjoyable discussions.

I also wish to thank Professor Robert DeVoe of the IU School of Optometry. His expertise on animal vision has helped me establish a broader context for some of the problems encountered in robotic perception.

Finally, my thanks go to Laurie, my spouse, for her patience during this, the latest of my academic stints.

ABSTRACT

Danko Antolovic

DEVELOPMENT OF A REAL-TIME VISION SYSTEM

FOR AN AUTONOMOUS MODEL AIRPLANE

This thesis describes a real-time embedded vision system capable of tracking two-dimensional objects in a relatively simple (uncluttered) scene, in live video. This vision system is intended as a component of a robotic flight system, used to keep a model airplane in a holding pattern above an object on the ground. The system uses a two-pronged approach to object tracking, taking into account the motion of the scene and the graphic "signature" of the object. The vision system consists of these main components: a motion-detection and filtering ASIC, implemented on FPGAs, a scene-analysis program running on a Motorola ColdFire processor, a dual-port RAM holding the image data, and a digital camera on a motorized gimbal.

_____

_____

_____

CONTENTS

1. INTRODUCTION TO THE SKEYEBALL VISION PROJECT

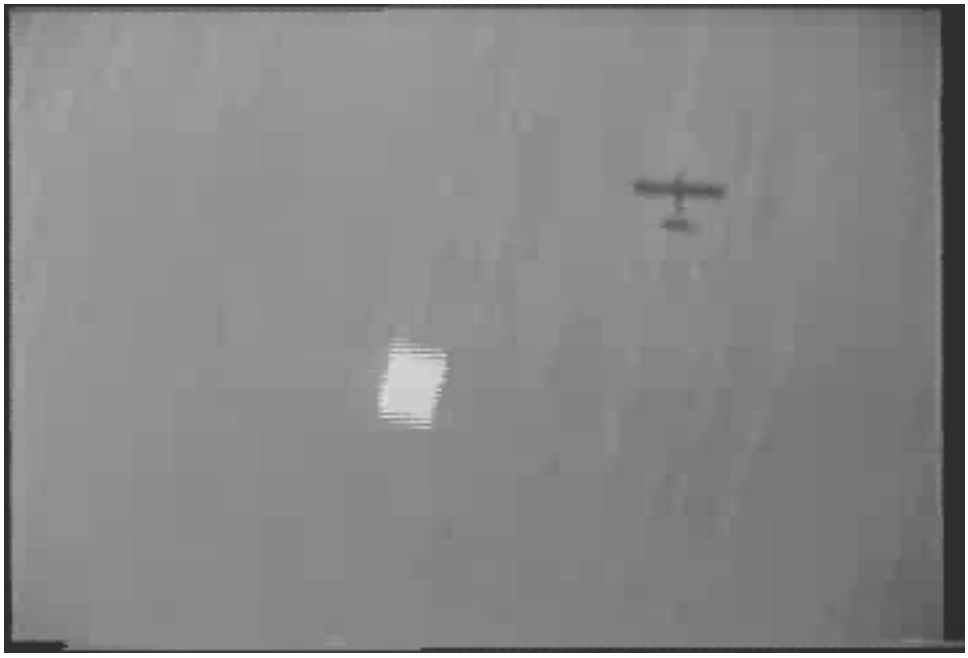
Skeyeball is an ongoing project in the Department of Computer Science at Indiana University. It is centered around a radio-controlled model airplane, which is being converted into a semi-autonomous vehicle. Its primary perception is a computer vision system, and it will also be equipped with attitude sensors, digital video and telemetry downlink, and digital command uplink.

The objective is to give the airplane the autonomy to fly beyond the line of sight, navigate, and find objects of interest by their visual appearance rather than by location.

The objective of the work described here was to build a vision system that follows an object in a relatively simple (uncluttered) scene, in live video. This vision system will be integrated into a larger robotic navigation system used to steer the airplane into holding pattern above a selected feature on the ground.

1.1 History of the vision system

The Skeyeball vision was first envisioned as a subsystem implemented on a microcontroller chip. Soon it became obvious that a fast (and not too costly) implementation of the early processing stages was needed: vision became an ASIC-cum-microprocessor system, and it is still such a system today.



Picture 1: Aerial view of a target overflight

The development has gone through two distinct phases. The first phase yielded a strictly laboratory prototype: the hardware was built from proto boards, and the processors were a Xilinx XC4010 FPGA and a Motorola MC68332. Data were shared through an SRAM on the common bus. This architecture required considerable data copying, and the 25 MHz MC68332 processor was rather too slow for the task. Nevertheless, the system was capable of (slow) object tracking, moving a camera on a simple gimbal. Pictures 2 and 3 show the gimbal and the circuitry of the first phase.

We then obtained some realistic footage by flying the airplane with the immobile camera. The laboratory prototype was capable of detecting target features in overflight sequences, but tracking an object reliably at flight speeds was very problematic. Picture 1 shows a

typical aerial view: the plane casts its shadow next to the bright square target (a brightly colored blanket on the grass).

This first phase gave us a fairly good insight into the minimal requirements of such a system. The second (current) phase is described in the rest of the thesis. Two major architectural improvements are a faster microprocessor (90 MHz Motorola ColdFire) and a dual-port RAM for shared data. Elimination of one very cumbersome proto board has also made the ASIC (application-specific integrated circuit) implementation much easier.

The fundamental vision algorithm has not seen much change over time, except for the addition of the threshold calculation in the second phase - the improvement has been the increased speed. Much greater modifications had to be made to the data flow procedures, to take advantage of the dual-port memory and better bus architecture.

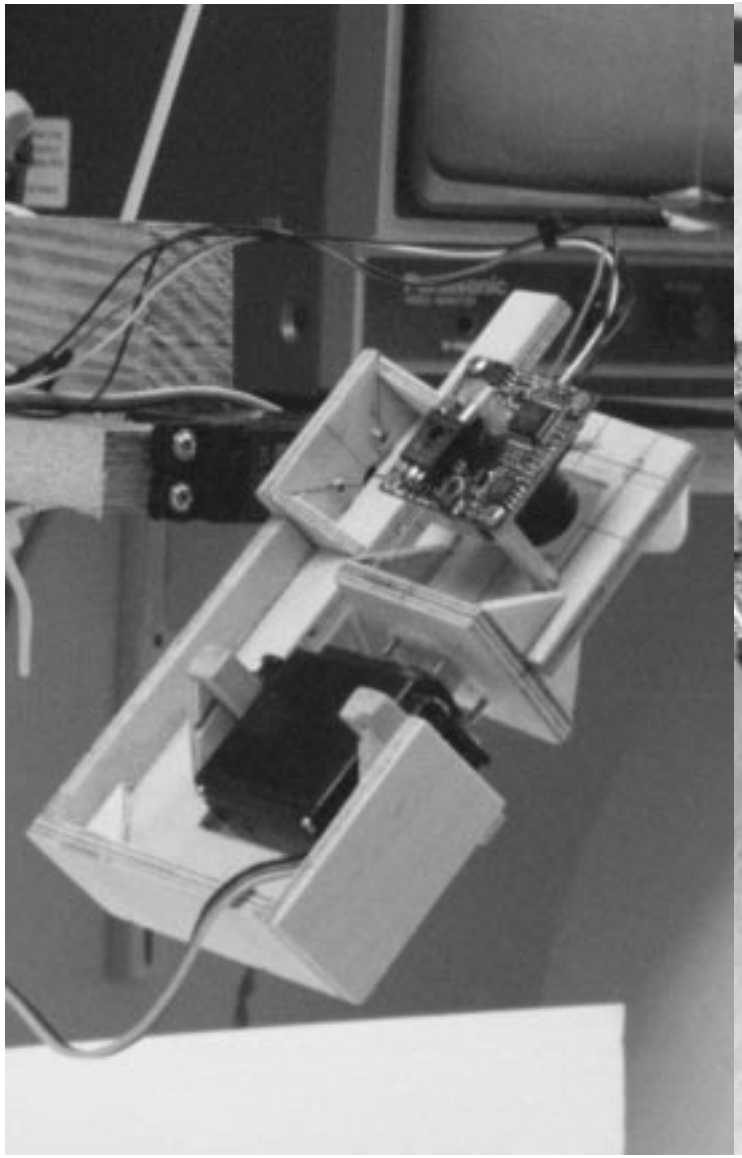
Finally, in the second phase, the system was given the proper startup procedure and radio controls, and the entire circuitry was built so as to be suitable for mounting inside the airplane. Pictures 4-7 show the equipment built in the second phase: three circuit boards, the new camera gimbal and the radio-controlled power switch. Picture 7 also shows the radio and TV links connected to the vision system.

1.2 Structure of this document

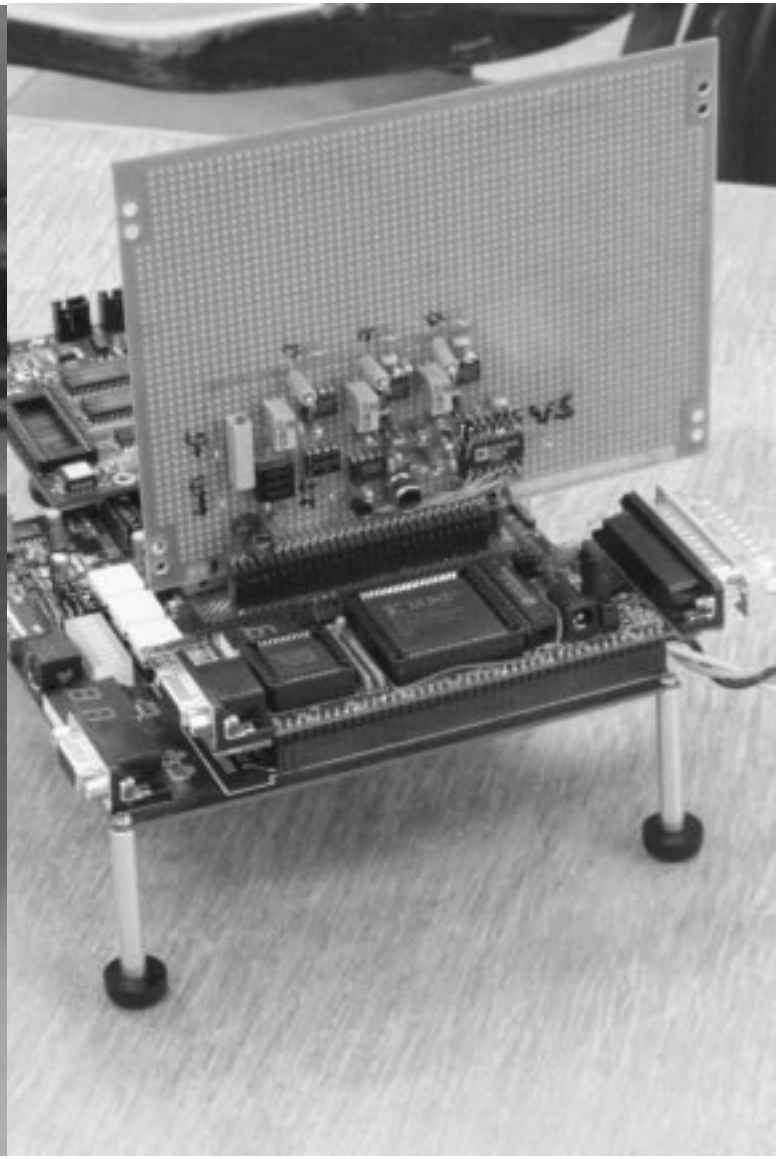
This document serves a dual purpose: it describes the constructed system as a solution to an engineering/computational problem in broad terms; it also describes it at the level of detail needed for modification and further development.

The system divides itself naturally into several subsystems. The first five sections of this document provide an overview, and the remaining sections describe the subsystems separately, with the level of detail increasing within each subsystem description. We have tried to make it clear where the broad description ends and the detailed one begins: typically, detailed descriptions are grouped into specialized subsections.

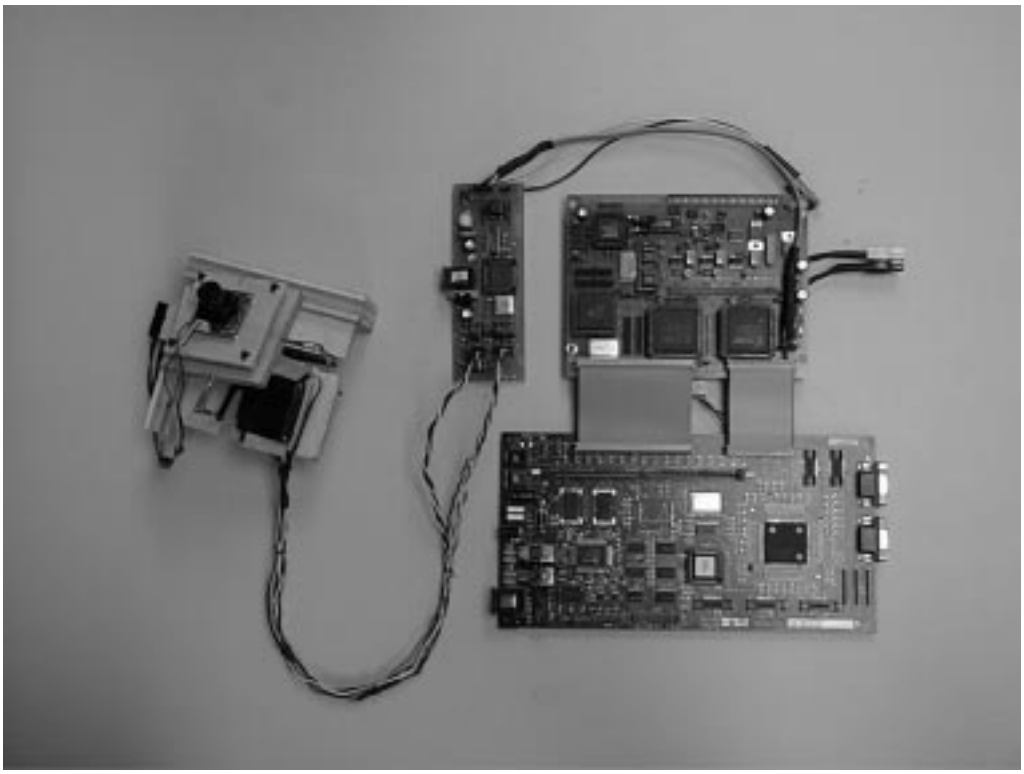
The ultimate level of detail - schematics, pinout lists and the source code - has been relegated to electronic form. This document contains summary descriptions of those materials, as well as passages referring directly to source details. Interested reader should become familiar with the circuit schematics and the source code.

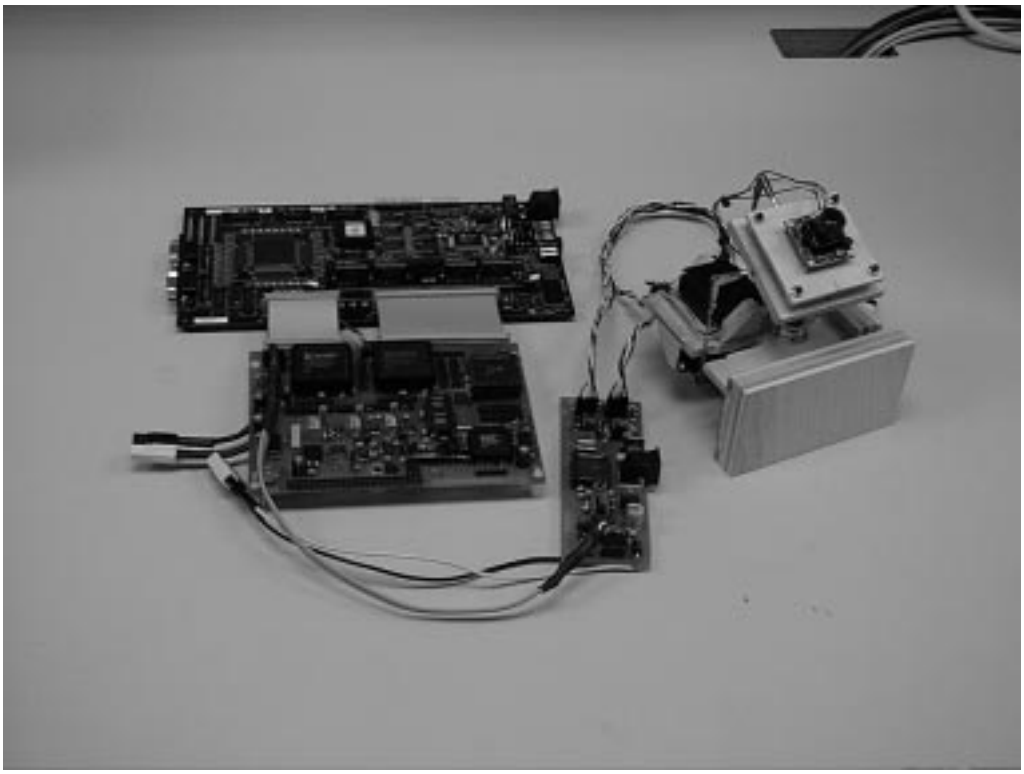


Picture 2: Gimbal, phase 1

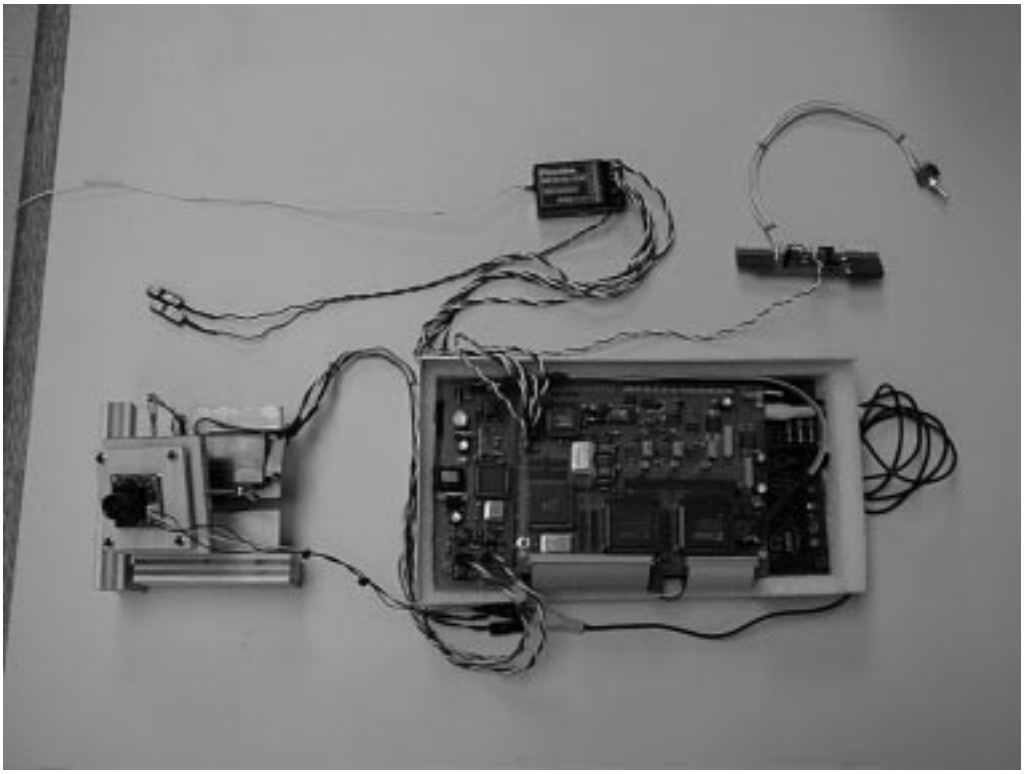


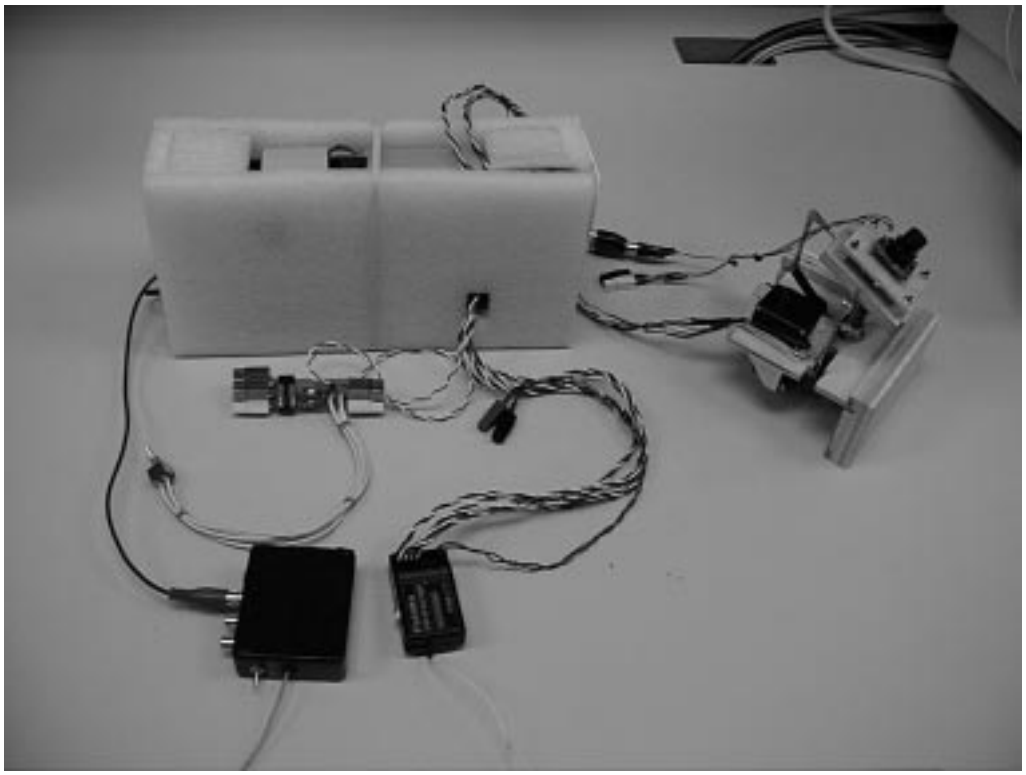
Picture 3: Circuit boards, phase 1





Pictures 4,5: Circuit boards and gimbal, phase 2





Pictures 6,7: Vision system, radio and TV links, power switch

2. FUNCTIONAL OVERVIEW OF THE VISION SYSTEM

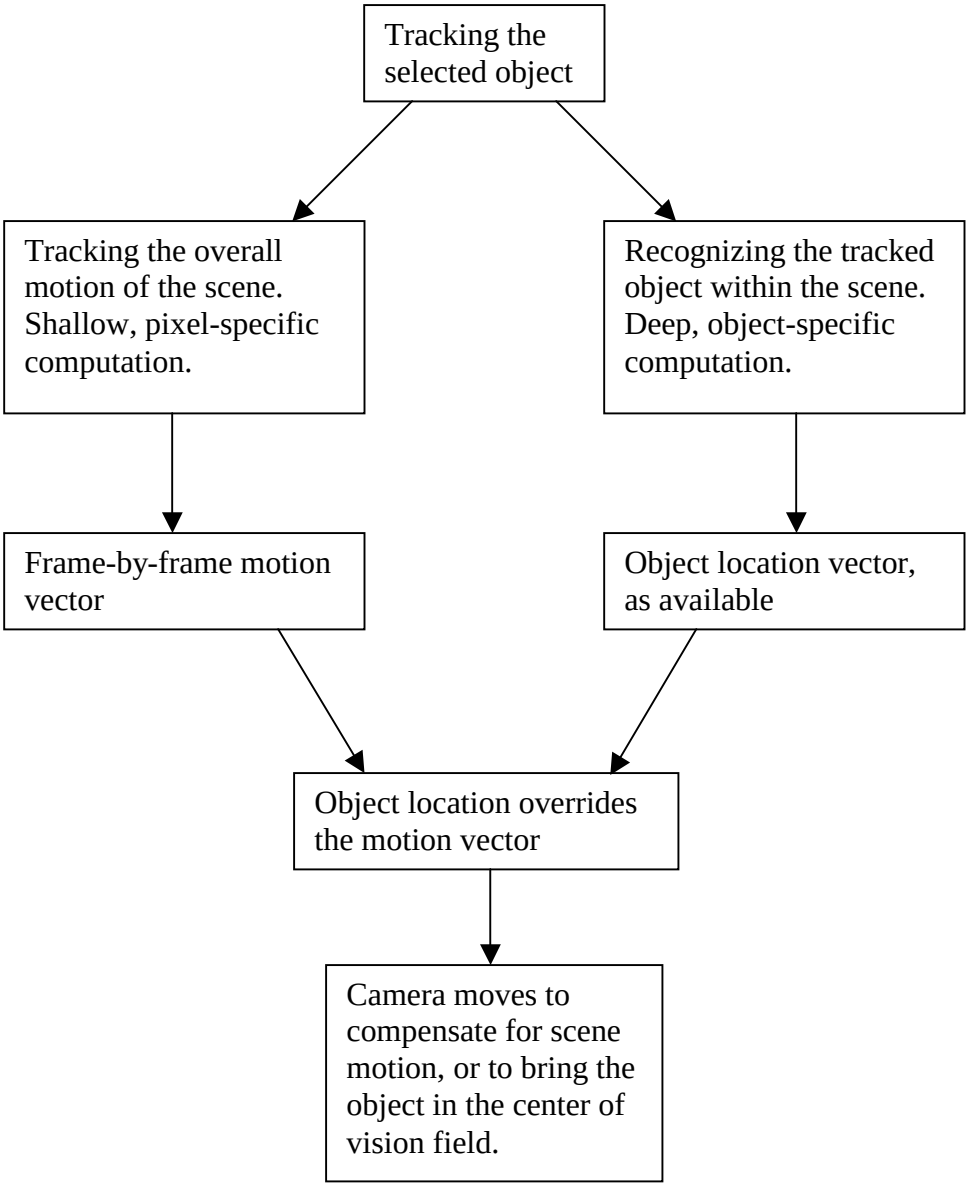
2.1 Vision methodology

As we stated in the introduction, the objective of this work was to build a vision system that will follow an object in a relatively simple scene, in live video. We have used a two-pronged approach to object tracking, taking into account the motion of the scene and the graphic "signature" of the object.

This approach was motivated by the fact that object recognition is computationally intensive, and impossible to accomplish on a frame-by-frame basis with the available hardware. Nevertheless, the vision system must operate fast enough not to allow the object to drift out of the field of vision. Picture 9 illustrates this point.

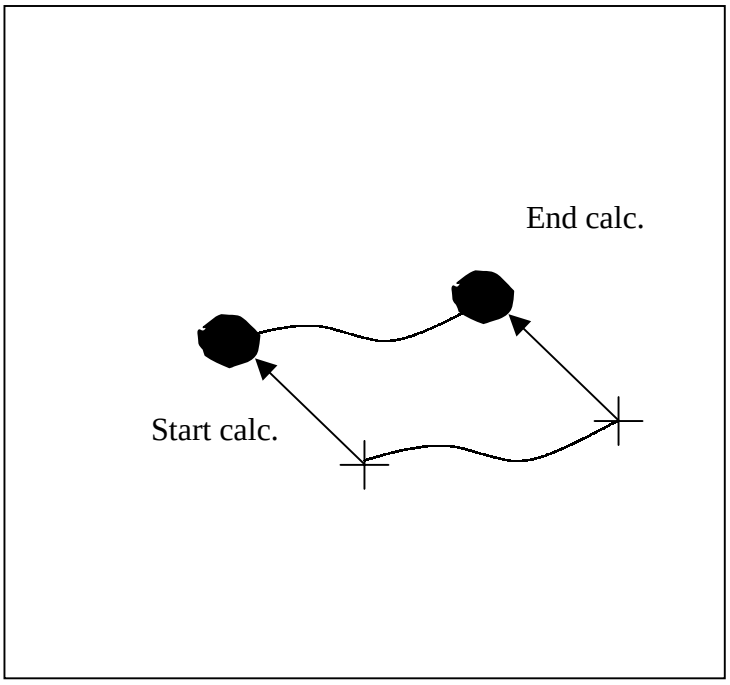
The objective is to recognize the small dark object as the target, obtain its offset from the center of the vision field, and move the camera to bring the object to the center. Obviously, the vision system must take a still frame and base its calculation on it. In the meantime, the object will change location, perhaps even drift out of the field. By the time it is calculated, the displacement vector may well be irrelevant.

To avoid this, the motion of the scene is tracked frame by frame, and the camera moves to compensate for it. The center of the field moves along with the target, and the displacement vector, when available, will still be meaningful. It should be said, however, that the importance of the drift compensation decreased as we used a much faster processor in the second phase of the project (see Section 1.1).



Picture 8: Method overview

Object recognition requires several stages by which redundant and non-essential visual information is parsed out, until we are left with a selection of well-defined objects, also referred to as the features of the scene. In our case, the stages are: thresholding, edge detection, segmentation into connected components, clean-up and signature/location calculation.



Picture 9: Drift compensation

Picture 10 gives a functional overview of the vision system, where the progressively thinner arrows signify the reduction in bulk of the visual information. This is the typical "funnel" of the vision problem, leading from simple computations on large volume of data to complex ones on a small volume, yielding a number or two as the result.*

* At 30 frames per second, 483 lines per frame, and 644 byte-sized pixels per line, raw camera output amounts to 9.33 Mbytes /sec. The 644 samples per line of continuous signal conform to the 4:3 aspect ratio prescribed by NTSC, and one byte per pixel is a realistic choice of color depth.

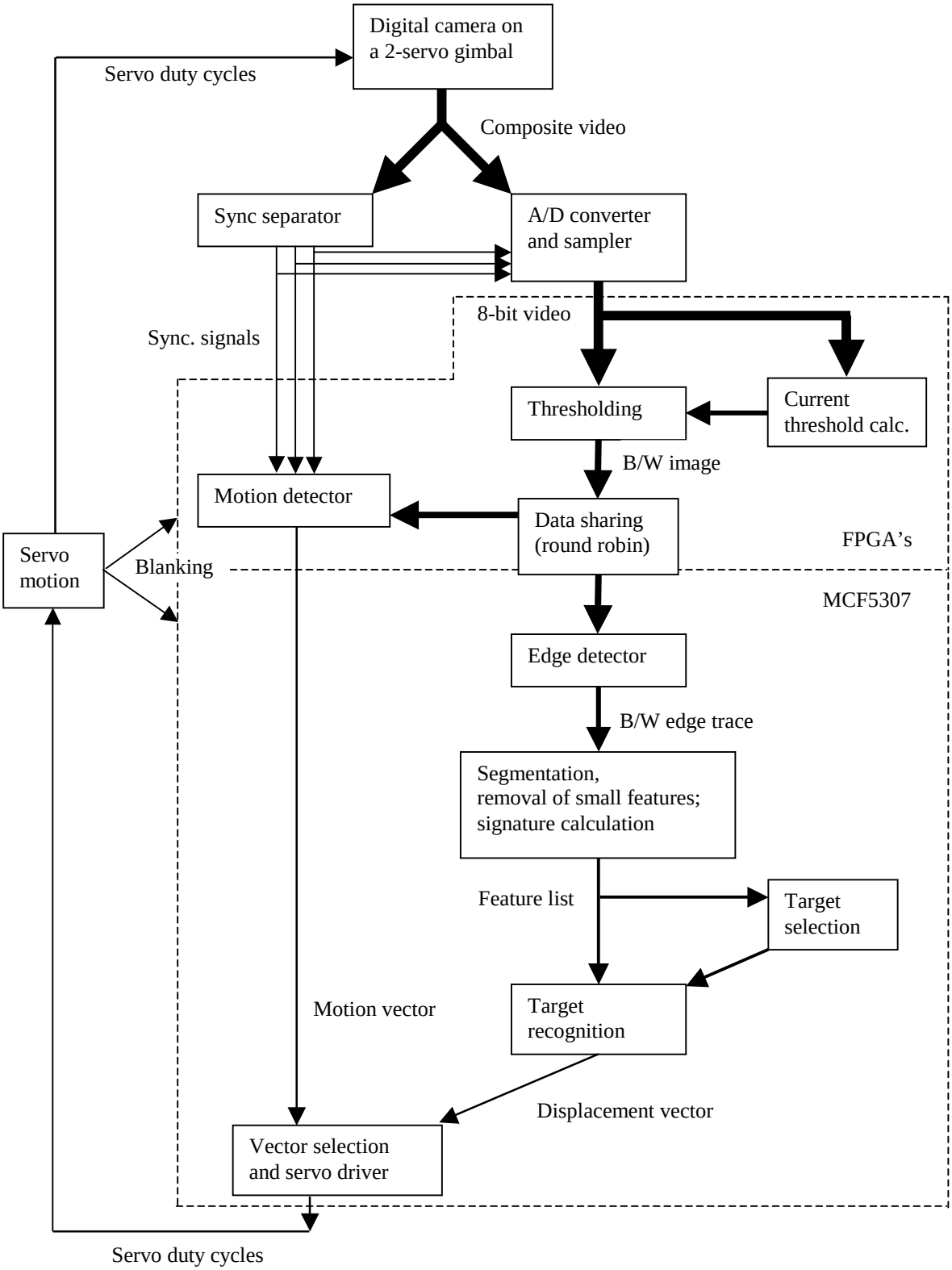
To identify a target object within the scene, we used the second moments about the object's principal axes of inertia as the "signature." Second moments are invariant under rotations and translations, fairly easy to calculate, and work well in simple scenes.

2.2 Biological parallels

It is not entirely surprising that certain functional analogies should develop between robotic perception, such as this real-time vision system, and perception in animals. While such analogies must not be taken too literally, they provide a glimpse at useful generalizations to be made about perception problems, and we will sketch them out as appropriate.

For example, camera motion based on feature recognition bears a similarity to the (involuntary) saccadic motion of the eyes, the one-shot movement that brings a detail of interest into the center of the vision field.[1] Eyes' saccades are fast, have large amplitude (as large as the displacement of the object of interest), and they are ballistic movements, i.e. they are not corrected along the way. Such a motion is compatible with a need for speed over precision: if the object identification is computationally intensive, use the result to full extent, and as fast as possible.

At the end of the process, the vision produces a few dozen bytes every couple of frames – actual rate depends on the contents of the image.



Picture 10: Functional overview

3. PROJECT STATUS

3.1 Capabilities and limitations

We have tested the lab vision system by manually moving irregular shapes in a vision field fairly free of random clutter, and also by means of a rotating table (Section 3.2 below).

The recognition system tracks its target reliably under translation and rotation, and in the presence of several shapes introduced as interference. Excessive skew breaks off the tracking, since the vision algorithm makes no provision for it, but an oblique view of ca. 20 degrees is still acceptable (Section 3.2). Likewise, occlusion is interpreted as a change of shape, and the target is lost when partially occluded (e.g. by drifting beyond the edge of the vision field). These limitations are obvious consequences of the vision methodology described in Section 2.1 and Picture 10.

The tracking and motion detection work equally well with the zoom engaged. As expected, zoom makes the system more reliable in tracking small targets, at the expense of limiting the field of vision to the central one-ninth.

3.2 Measurements of the tracking speed

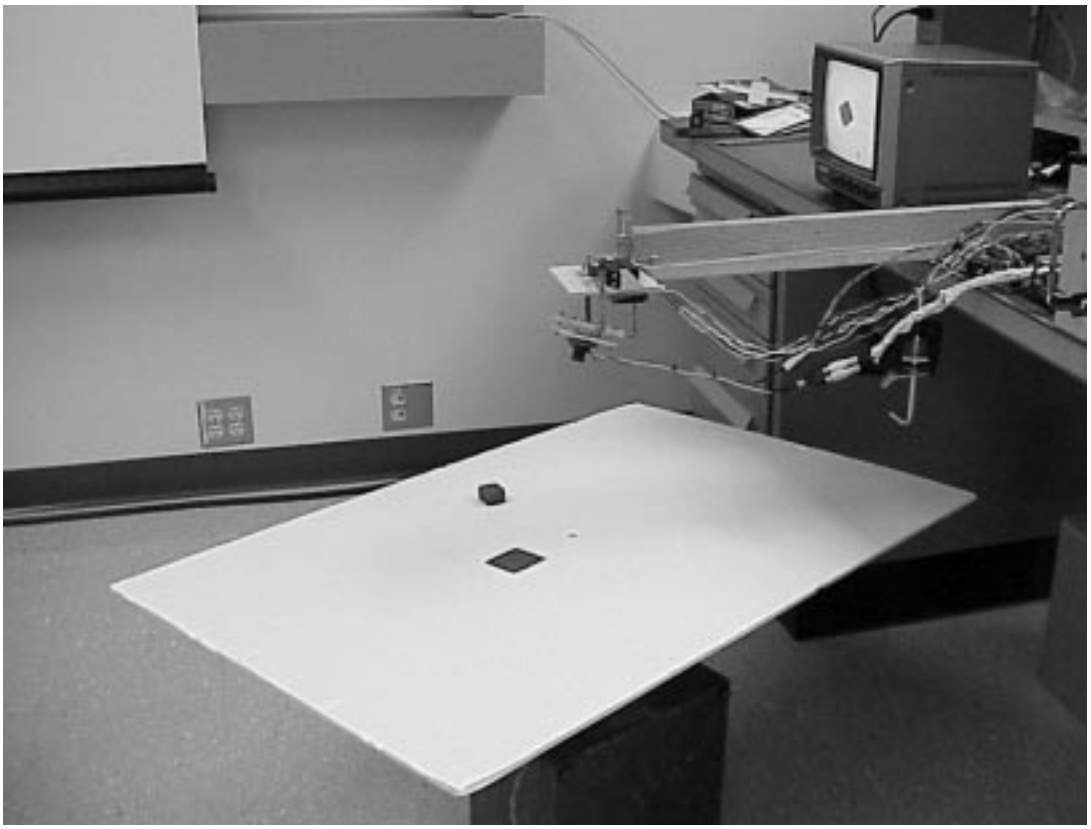
The Skeyeball airplane flies within certain ranges of speed and altitude; our fixed-camera flights have ranged from 18 to 60 mph, with a typical speed of ca. 35 mph. Likewise, target-overflight altitudes have been from 60 to 320 ft. Consequently, the line of sight to the target feature changes direction relative to the body of the airplane, with

certain angular velocity, and the vision system must be able to keep up with it. In the test flights, the target passed through the vision field of the fixed camera in time intervals ranging from 0.8 to 4.2 seconds, depending on the altitude and velocity of the airplane.

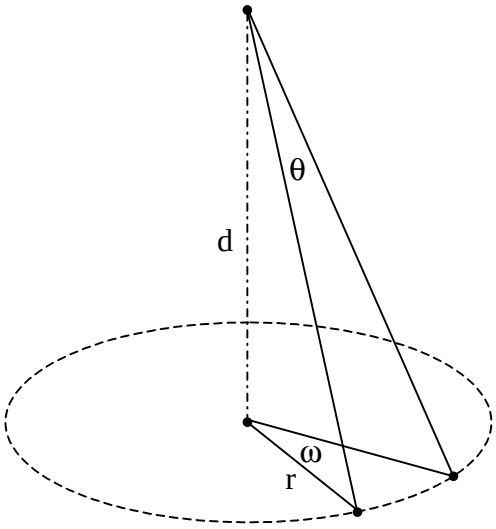
In order to obtain some quantitative measure of the vision's tracking abilities, we have constructed a test rig - a rotating table with features to track. The vision system locks successfully onto the (largest) presented feature and the camera turns following the rotation of the table. The rotation speed is gradually increased, until the tracking breaks off or target acquisition becomes impossible. Picture 11 shows the experimental setup: the camera gimbal, the rotating table with two features, and the TV screen showing the camera's view.

Picture 12 shows the geometry of the setup. The speed of the table's motor was regulated by applying variable voltage, and the angular speed of the table was measured as the time needed for ten turns. A simple formula relates the table's angular speed ω to the camera's sweeping speed, θ:

$$\theta = \frac{\omega}{\sqrt{1 + \left(\frac{d}{r}\right)^2}}$$



Picture 11: Laboratory set-up for the tracking-speed measurements



Picture 12: Table speed vs. the camera's sweep

d is the elevation of the camera above the table, and *r* is the distance of the target feature from the center of the table.

At d = 26.5 cm, and r = 10 cm, we found that the tracking was still reliable with the camera sweeping an arc at the maximum speed of:

$\theta(\max) = 45$ degrees/second.

The camera's field of vision is ca. 47 degrees high and ca. 60 degrees wide (see Section 20), which puts this vision system within the range of speeds required to keep up with the overflight speeds that were quoted above.

Of course, tracking speed depends on the complexity of the scene. These measurements were performed with two or three shapes, plus an intermittently visible edge of the table. The scene observed in a real flight is richer in features, but at least for grassland and trees, features tend to have low contrast and disappear below the threshold, which in turn is set to single out high-contrast targets.

3.3 Summary remarks on the perception problem

Real-time perception can be envisioned as a funnel in which the data volume is reduced, but the algorithmic complexity increases. Typically, there will be several stages with fairly different breadth/depth ratio.

This is intrinsically not a problem amenable to single-architecture processing. Of course, a speed tradeoff is in principle always possible, but engineering considerations such as

power consumption and heat dissipation place a very real limit on that approach. We believe that it is better to use several processor architectures, each suitable for a different stage of the perception process. Appearance on the scene of configurable microchips makes this goal both realistic and appealing.

Robotic perception is also a problem in embedded computing. Requirements imposed by the small model airplane are a bit on the stringent side, and one can envision a much more relaxed design for an assembly line or security system. However, the need for perception is naturally the greatest in mobile robots. In such applications the vision system will always have to be compact and autonomous, because it bestows autonomy on a mobile device whose primary function is something other than carrying a vision system around.

Architecture should follow function, starting at a fairly low level. For example, data collection in this system is done with a digital camera which serializes the (initially) parallel image input. This choice was dictated by good practical reasons, but the system lost a great deal of processing power because of that serialization. Image input should have been done in parallel, which in turn would have required a specialized device and a much broader data path in the initial processing stage.

The segmentation stage is better suited for implementation on general-purpose processors because of the smaller data volume and more "sequential" algorithms. An architectural alternative may be possible here: segmentation could be attempted on a highly connected

neural net circuit, trading off an exact algorithm for an approximate, but parallelizable, search procedure. Neural net searches, on the other hand, are usually slow to converge and may not improve the overall speed.

Animal vision cannot be separated from cognitive functions and motor coordination, and this must be true for robotic vision as well. How much "intelligence" is built into high-level processing of visual information depends on the ultimate objectives of Skeyeball: for example, searching for a 3D shape in a complex panorama is a problem different from that of hovering above a prominent feature on the ground.

In terms of steering and motor coordination, biological parallels are relevant. It is known that inertial motion sensors play a large role in the gaze control of mobile animals [1]. Since the input from the motion sensors is simpler, and the processing presumably faster, this sensory pathway provides the supporting motion information much faster than can be obtained by visual processing. The airplane may very well benefit from an eventual integration of its vision and attitude/motion sensors.

Visual perception is an ill-posed problem, and examples of functioning compromises may be more valuable than exact results. Throughout this document, we point out similarities with biological systems which strike us as interesting, although we do not pursue them in depth for lack of expertise on the subject. A few pitfalls notwithstanding, we believe that a synthesis of computational, physiological and engineering knowledge will be necessary for the eventual development of reliable and versatile perception systems.

4. HARDWARE ARCHITECTURE

4.1 Architectural components

Picture 13 is an overview of the architecture of the vision system. Picture 14 shows all the signals pertaining to the flow of data from the camera, through the processors and back to servo motors, but it omits some peripheral details.

Camera - The “eye” of the system is a small digital camera, producing grayscale (non-color) NTSC video signal; its other characteristics are largely unknown. The camera is mounted on a gimbal driven by two servo motors, with a 50-degree range of motion in each direction, and is permanently focused on infinity.

Sync separator – the NTSC grayscale video signal contains three synchronization signals. These are extracted by means of video sync separator LM1881 by National Semiconductor, mounted on a prototyping board along with supporting circuitry.

A/D converter – we use Analog Devices’ AD876, which is a pipelined 10-bit converter. It is mounted on the same proto board, with supporting circuitry for its reference voltages.

Sampling control, thresholding and threshold calculation, motion detection and zoom are implemented as digital designs on a synchronous pair of Xilinx XC4010 FPGA's, running at 33.3 MHz. Start-up configuration is done with two Atmel’s AT17LV config ROMs.

The entire object recognition is implemented as code, running on a 90 MHz Motorola MCF5307 ColdFire integrated microprocessor. We use a commercial evaluation board, SBC5307, with 8 megabytes of DRAM, start-up flash ROM, expansion bus and communication ports.

The two processors share image data through a 32K dual-port SRAM, CY7C007AV by Cypress Semiconductor. Data access is implemented as a round-robin procedure, with the objective of speeding up high-volume data transfer in the early stages of the vision process.

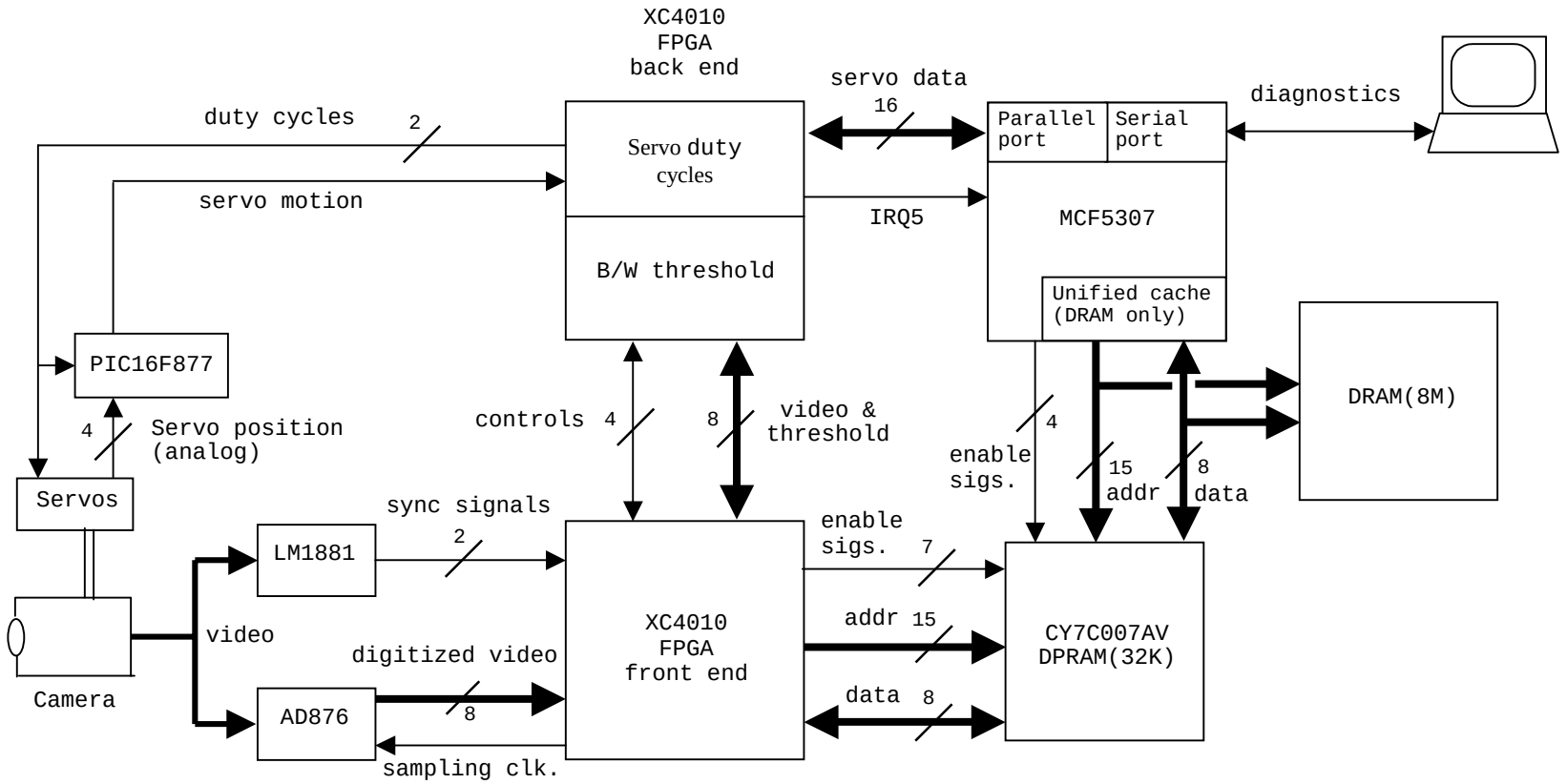
The driver for the servo motors that move the camera is implemented on one of the two FPGA's. Motion feedback from the servos is generated by a PIC16F877 microprocessor, on the basis of servos' analog position signals.

4.2 Biomorphic approach to architecture

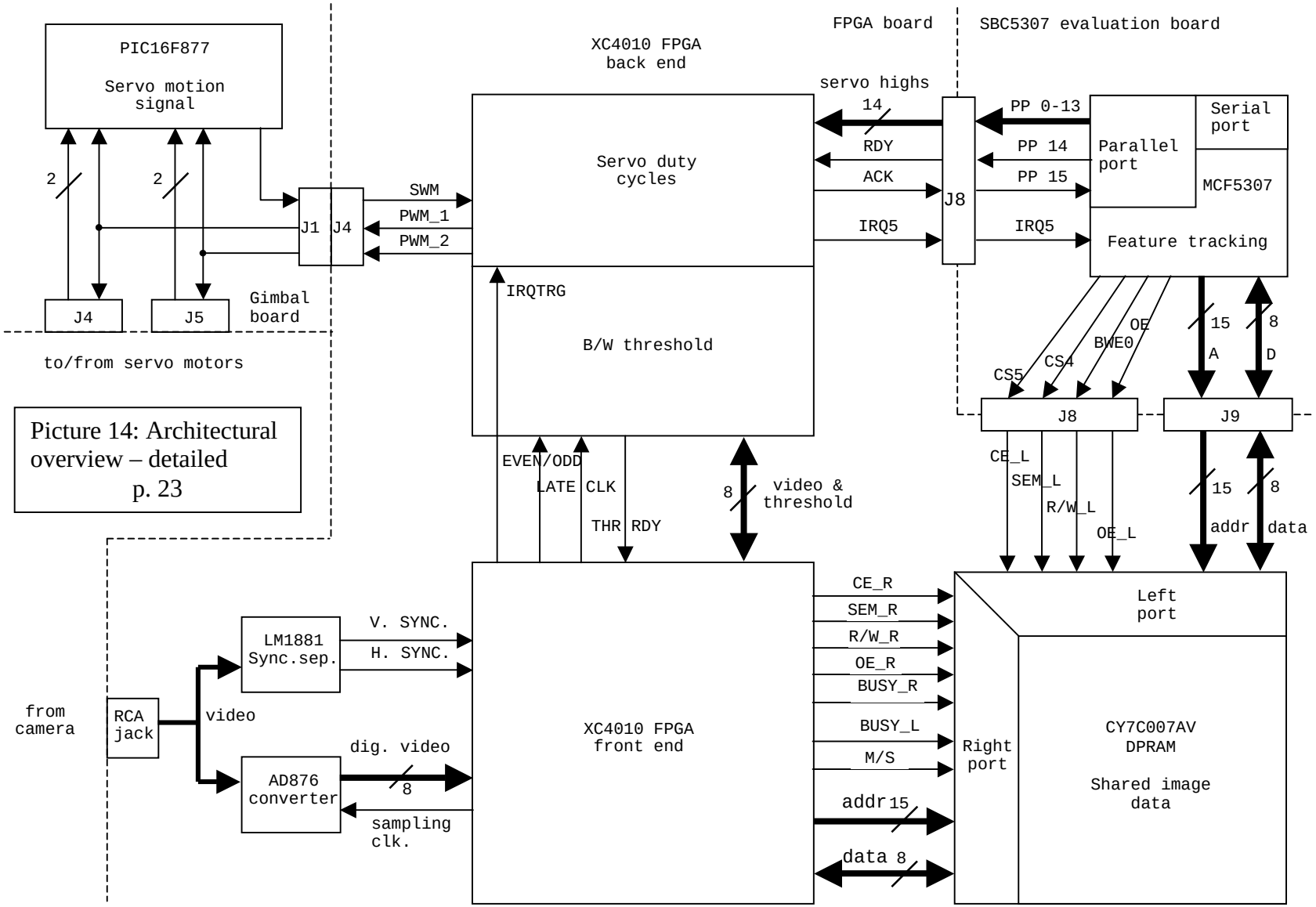
Nervous systems of animals utilize specialized hardware almost by definition. There is much evidence that biological architecture follows function: for example, the retina, with its layers of many specialized types of cells, is apparently a structure which has evolved to deal with the initial stages of the vision funnel, from the cellular level up.

Architecture of this robotic system follows the same "biomorphic" principle as much as possible. In order to increase overall speed and throughput, we have opted for ASICs and dedicated data paths, even at the cost of under-utilizing some components. Multitasking

and time-multiplexing are systematically avoided. Biological systems follow this principle because of evolutionary constraints, but they solve the real-time perception problem well, and the trade-offs they make appear to be the right ones.



Picture 13: Architectural overview


PIC16F877
Servo motion signal
2
2
J1
J4
SWM
PWM_1
PWM_2
J4
J5
Gimbal board
to/from servo motors
XC4010 FPGA back end
Servo duty cycles
IRQTRG
B/W threshold
FPGA board
SBC5307 evaluation board
servo highs
14
RDY
ACK
IRQ5
J8
PP 0-13
PP 14
PP 15
IRQ5
Parallel port
Serial port
MCF5307
Feature tracking
15
A
8
D
OE
BWE0
CS4
CS5
J8
J9
CE_L
SEM_L
R/W_L
OE_L
15
addr
8
data
EVEN/ODD
LATE
CLK
THR
RDY
8
video & threshold
Picture 14: Architectural overview – detailed
p. 23
from camera
RCA jack
video
LM1881 Sync.sep.
V. SYNC.
H. SYNC.
AD876 converter
dig. video
8
sampling clk.
XC4010 FPGA front end
CE_R
SEM_R
R/W_R
OE_R
BUSY_R
BUSY_L
M/S
addr 15
data 8
Right port
Left port
CY7C007AV DPRAM
Shared image data

5. DESIGN SUMMARY

This section provides a top-level summary of the schematics and code modules which comprise the functional configuration of the hardware described in Section 4.

5.1 XC4010 digital design

Configuration of the two FPGA processors is implemented with the Xilinx Foundation development tool, either as schematics or as Abel HDL code.[2] This list gives an overview of the functionalities contained in the highest-level modules. Design components and signals are described throughout the text and in the schematics themselves.

5.1.1 Front-end FPGA

This design consists of seven top-level schematics and a number of macros. It utilizes about 60% of the logical blocks (CLBs) of the XC4010 FPGA.

VISION_IN – contains the entry point for the video sync signals, vertical and horizontal image framing and sampling, and the zoom.

ANALOG_IN – input from the A/D converter, normalization to black reference level, black-and-white thresholding.

FIELD_END – placeholder schematic, invoking macros for vector output, run-time parameters and the round robin.

VISION_OUT – input/output to the DPRAM.

RR_ADDRESS – address counters for image buffers, round-robin address multiplexer.

MOTION_VECT – invokes the pixel read/write cycle, calculation of the motion vector.

CLOCKS – entry point for the external clock and reset signal, generation of internal clocks.

5.1.2 Back-end FPGA

The design consists of five top-level schematics and a number of macros. It utilizes 96% of the CLBs of the XC4010 FPGA.

DUTY_CYC_PP – IRQ5/parallel port communication, duty cycle generator, servo-motion signal.

HIST_DP – data path for the threshold search in the histogram.

HIST_IO_RAM – histogram storage and management.

HIST_MINMAX – comparison logic for the threshold search, control ASM.

HIST_INTEGRAL – calculation of the histogram and features area.

5.2 MCF5307 (ColdFire) code

Programs running on the MCF5307 ColdFire processor are written in C and ColdFire assembler,[3] and compiled/assembled with GNU "gcc" and "as." This list groups the code modules by system function; further description is provided in the text and in the source comments.

main.c	- initialization and main loop for feature recognition

Configuration and startup:

cache.s	- cache initialization
ConfigRegs.s	- MCF5307 configuration, running from flash
ConfigRegs2.s	- MCF5307 configuration, running from DRAM
crt0.s	- setup for the C language
globals.c	- init. of global variables for functional code
glue.c	- heap setup and other book-keeping
start.s	- processor startup sequence
vector.s	- vector table

Feature recognition:

ConnectedComponents.c	- segmentation algorithm
Diagnostics.c	- vision system's error reporting
FeatureDetector.c	- feature "signatures"
FeaturePoints.c	- maintenance of heap data structures
Features.c	- driver modules for acquisition and tracking

GraphDFS.c	- depth-first graph traversal
SimpleEdge.c	- edge detector

Inter-process communication:

CyclesPP.s	- communication through the parallel port
IRQHandler.s	- handler for the Interrupt 5 (Process 1)
roundRobin.s	- round-robin DPRAM access

Servo motion:

servo.s	- translating displacements to servo duty cycles

Auxiliary:

Datalog.c	- interface library for the diagnostic data log
DataOutput.c	- diagnostic data output
serial.c	- communication library for the serial port(s)
SerialHandler.s	- UART interrupt handler
IRQ7Handler.s	- handler for the Interrupt 7 (soft restart)
TermInput.c	- stub for the terminal command input

6. NTSC AND THE EVENTS SYNCHRONOUS WITH THE VIDEO SIGNAL

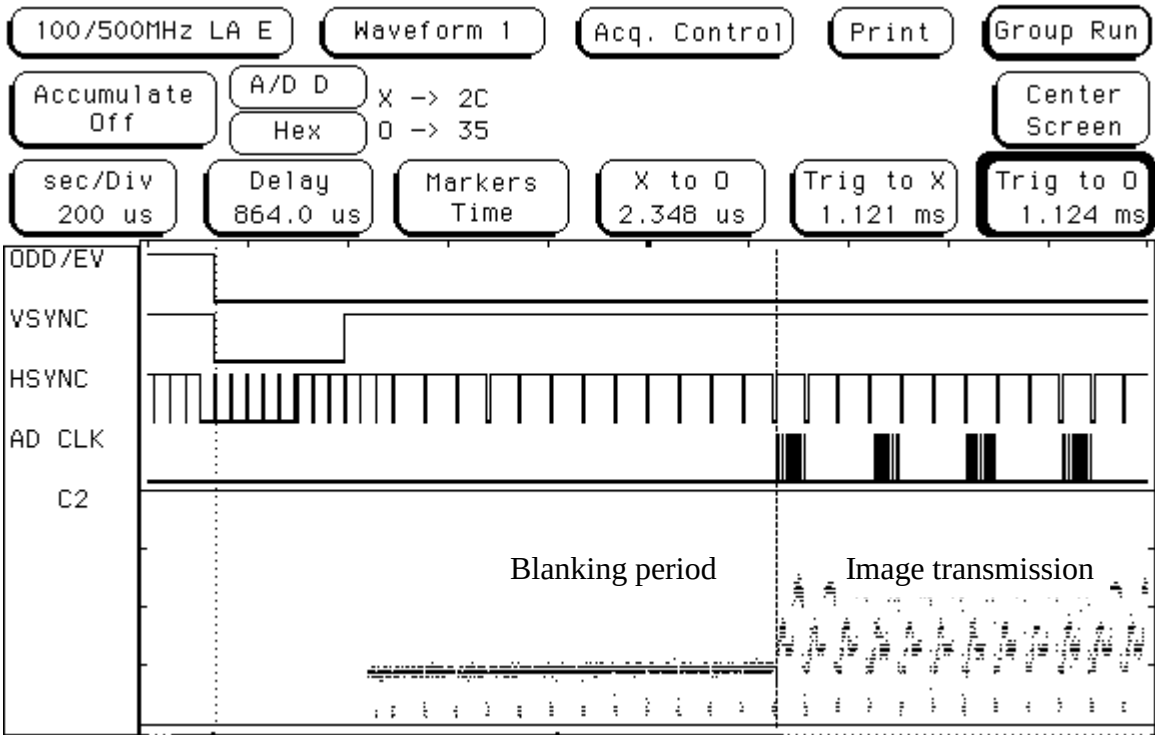
Frames of the NTSC television signal[4] consist of two interleaved fields, marked by an even/odd synchronization signal. At the beginning of each field there is a period of time when the beam retraces back to the top of the image (at low intensity), a period marked by the vertical synchronization signal. Also, horizontal retracing between video lines is marked by the horizontal sync.[5] Pictures 15 to 18 show some details of the operations synchronous with the video signal.[†]

We sample the content of the video signal during the even field of each frame, and at variable resolution – every third line in the absence of zooming, every line when the zoom is engaged. Motion detection, which compares adjacent video frames, is performed simultaneously with the sampling, between the pixels, as it were. An early fraction of the odd field is used for communication between processes and for the threshold calculation.

6.1 Even field

Picture 15 shows the beginning of the even field, the synchronization signals, the A/D sampling clock, and the composite analog video signal. Notice the long vertical blanking period before the beginning of the actual image transmission.

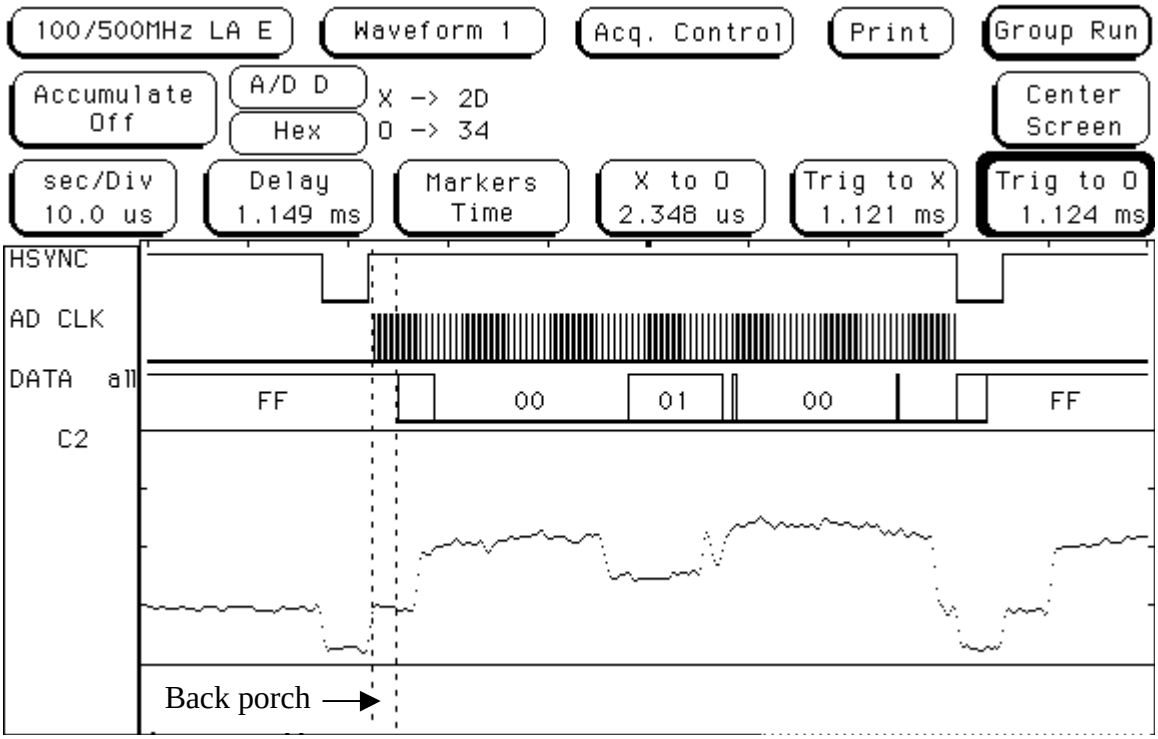
[†] Pictures 15-18, 27, 28, 32, 35, 38 and 42 are screen shots from a Hewlett-Packard 16500B logic analyzer.



Picture 15: The even field

The sampling clock operates in bursts, during the sampled portion of the even field (see Pictures 15, 16, 19 and 26). First sample of each line is taken during the so-called back porch of the horizontal sync, at the black reference intensity. We use the first sample as the zero reference for the remaining grayscale values in that line: there is quite a bit of intensity wobble in the camera signal, and this referencing makes the image steadier.

Picture 16 shows the video content of one line, the sampling clock and the thresholded digital data obtained from the video signal. The values of one (black pixels) in the middle of the line correspond to the dip in the video signal, which was in turn caused by a dark object at the top of camera's vision field.

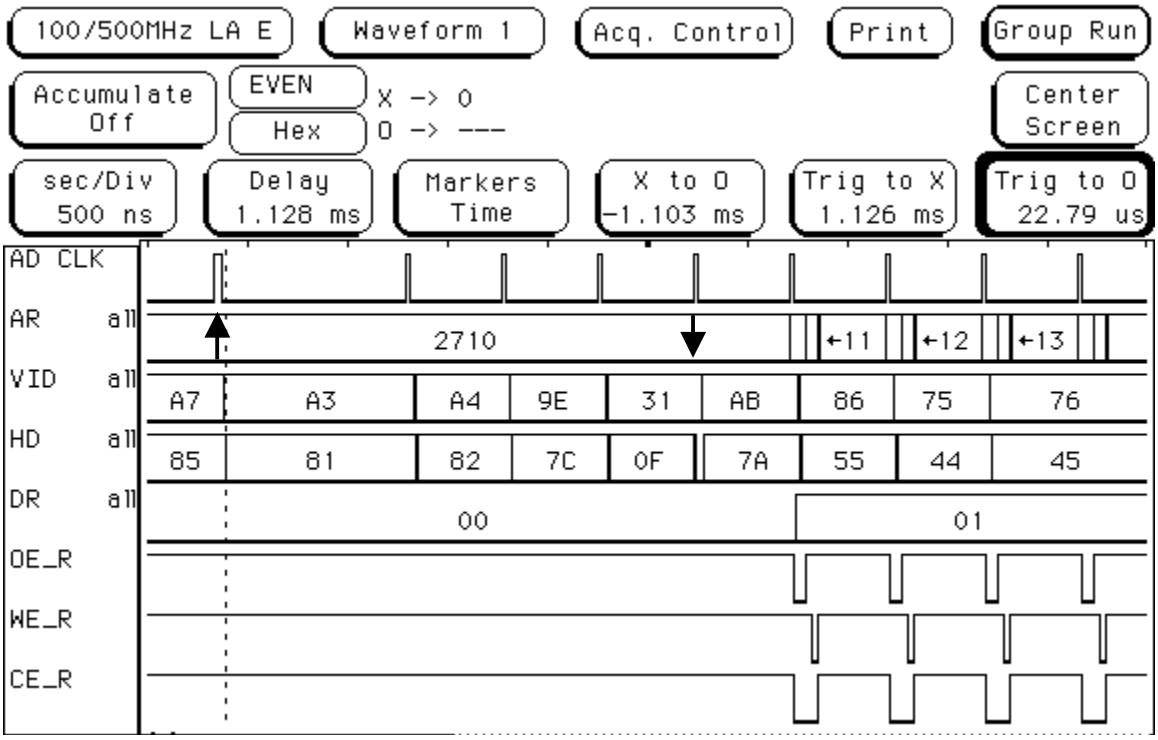


Picture 16: Video signal carrying one line

Picture 17, at 1.7 MHz sampling rate, shows the delay between the sampling clock and the return of digital data. The A/D converter is pipelined,[6] introducing a delay of four clock periods, which we account for by delaying the beginning of pixel read/write cycles. The I/O at the end of the line extends past the sampling clock by the same amount.

The field AR is the pixel's address in the DPRAM (right port), VID is the converter's output. First tick of the AD clock occurs at the end of back porch, and the resulting black reference value is latched four ticks later, as 0x31 in the VID signal. HD is the normalized grayscale value: notice that VID – HD = 0x31 past the black reference.

DR is the thresholded signal, showing in this case a black object in the first line. OE_R, WE_R and CE_R are the memory control signals of the DPRAMs right port.



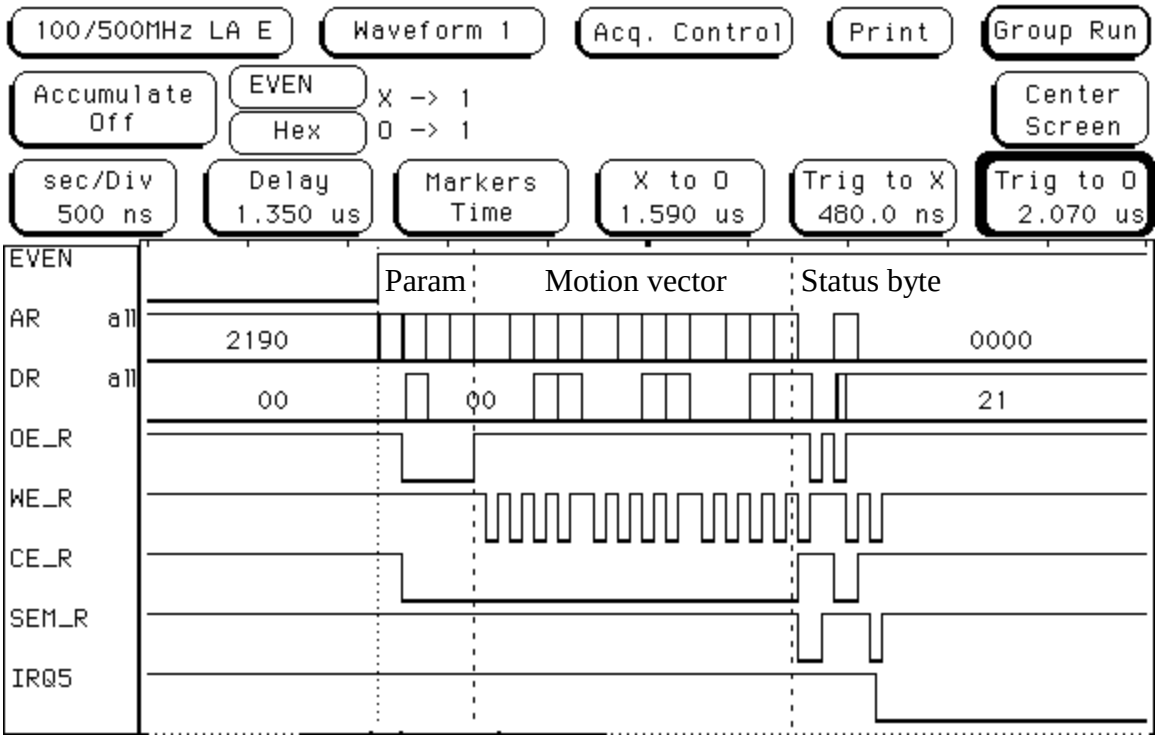
Picture 17: A/D pipeline delay

Motion vector calculations and storage of the digital image are described in Section 12, dealing with the pixel read/write cycle (p. 64).

6.2 Odd field

Picture 18 shows the beginning of the odd field and the DPRAM I/O associated with it. At this time, parameters are updated, the motion vector has been calculated and is written

out (notice the twelve dips in the WE signal), and Process 1 updates the status byte (notice that the semaphore operation SEM_R brackets the status byte read and write). IRQ5 is asserted, triggering the handler on MCF5307 and starting the parallel-port communication (see Section 16, on IRQ/PP).



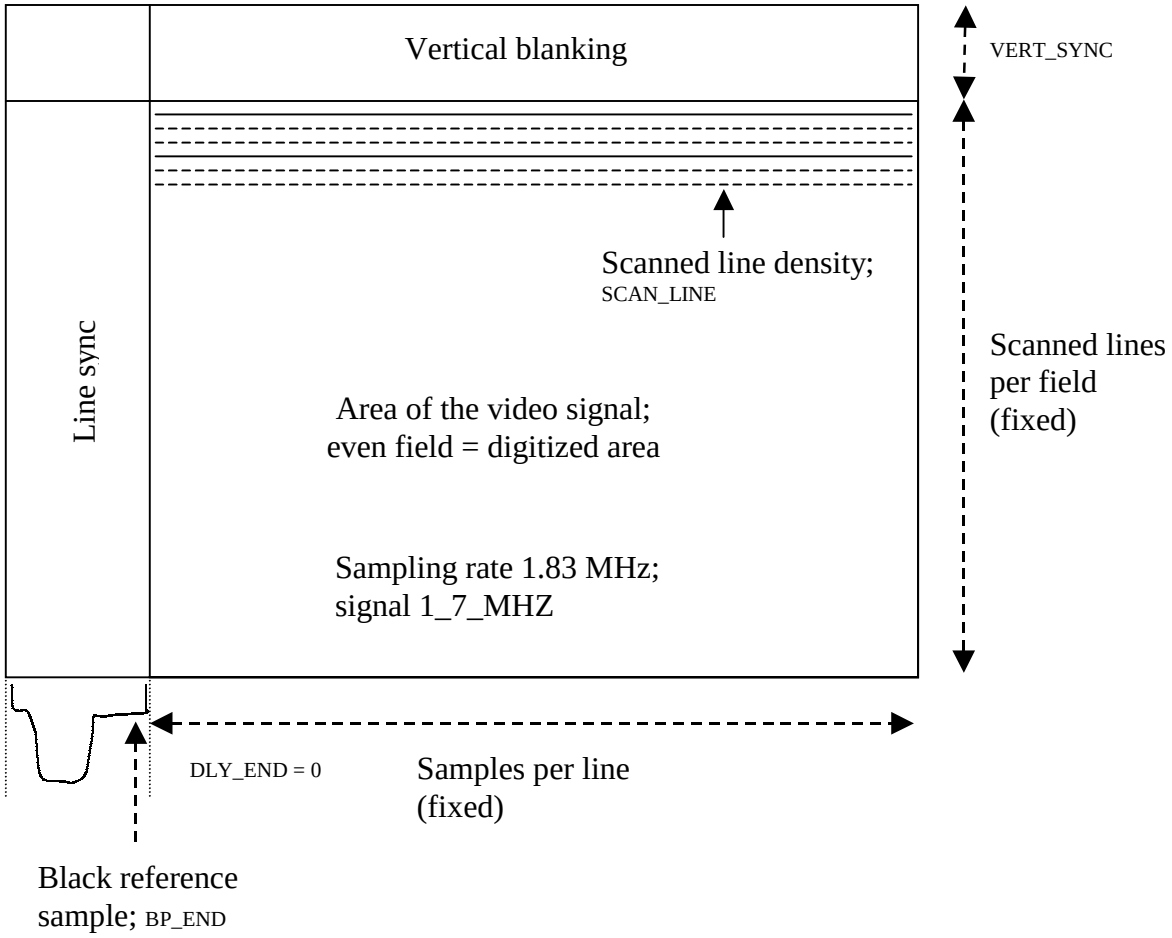
Picture 18: The odd field

7. FORMATTING THE IMAGE SCAN

The design receives the sync signals already separated from the total video signal. It uses the syncs to control the digitization and framing, to assign coordinates to pixels, and to count total numbers of pixels and black pixels in the image. For convenience in analyzing the VCR video signal, which does not have the even/odd sync, this sync signal is being generated internally.

Deciding at which points to sample the video signal, i.e. generating a sampling clock for the A/D converter, is the main functionality derived from the synchronization signals. Vertical formatting means the selection of video lines, while the horizontal formatting means a selection of discrete sampling points on the continuous video signal. The two formats differ in details, and are made somewhat more complex by the presence of the zoom (see Section 10, on the digital zoom). Also, in this system, sampling is limited to the even field of the frame.

Picture 19 shows the formatting geometry and the signals involved, in the absence of zooming. Zoom geometry is shown in Picture 26.



Picture 19: Formatting and sampling

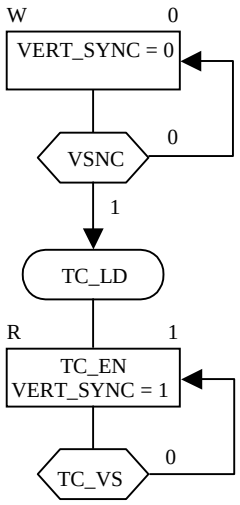
7.1 Vertical formatting

Briefly, the vertical formatting circuit (in the schematic VISION_IN) determines three things:

- at which line in the even field to start sampling
- at what line density to scan (how many lines to skip between scans, if any)
- how many lines to scan (i.e. when to stop)

7.1.1 Starting line

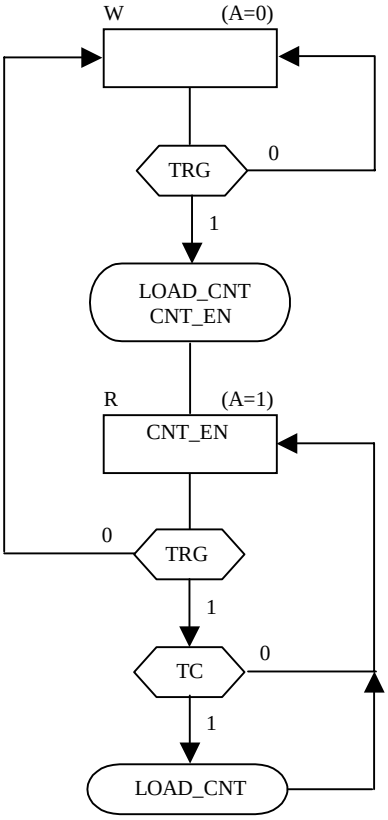
Obviously, sampling must be suppressed during the vertical retrace (vertical blanking), and when the 3X zoom is engaged, over the top third of the image as well. This is accomplished by extending the duration of the vertical sync to the first scanned line, counting the requisite number of lines. A loadable counter and a small ASM, triggered by the signal VSNC and clocked by LINE_SYNC, produce the signal VERT_SYNC, which extends to the first scanned line.



Picture 20: VERT_SYNC ASM

7.1.2 Line density

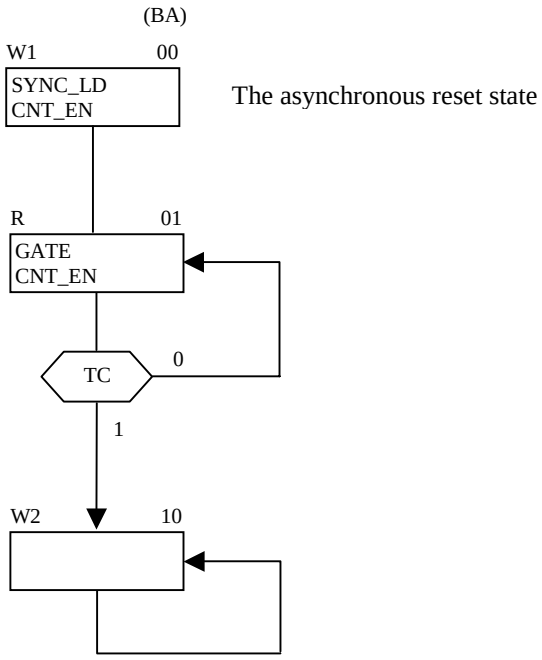
Component LOAD_CNT2 is a counter which reloads an external value D_IN whenever it runs out, then continues running to that value. It has a provision for the zero count, and two term count signals, full period and half period. It runs while its TRG input is high. This component is used to set the line density, by clocking it with LINE_SYNC.



Picture 21: LOAD_CNT2 ASM

7.1.3 Line count

Component SCAN_LPF is a stopping counter which is reset asynchronously on the rising edge of its TRG input. Its output GATE remains high for the duration of the count; afterwards the counter sleeps until the next reset. This component is used to count the scanned lines in the even field.



Picture 22: SCAN_LPF ASM

7.2 Horizontal formatting

The horizontal formatting circuit (also in VISION_IN) does these four things:

- decides at what pixel density to sample
- produces the sampling signal for the black reference at a fixed position in line
- decides where in the scanned line to start sampling pixels
- and how many pixels to sample per line (i.e. when to stop)

7.2.1 Pixel density

An ASM and a loadable counter, identical to those in LOAD_CNT2, generate a clock signal at the pixel sampling frequency, by reducing the system clock by a factor dependent on the zoom.

7.2.2 Black reference sampling

Component LOAD_CNS is a counter which synchronously loads an external value D_IN, runs to that value and stops. It has a provision for the zero count, and two term count signals, full period and half period. It starts when its SYNC_LOAD input goes high. This component is used to generate the black ref. sampling signal (BPE), by counting off the length of the back porch in the intervals of the sampling clock.

7.2.3 Starting pixel

A second LOAD_CNS counts off the delay from BPE to the first pixel (zero, or one third of the line for the 3X zoom). It raises the signal EN_SAMP, during which pixel sampling is enabled.

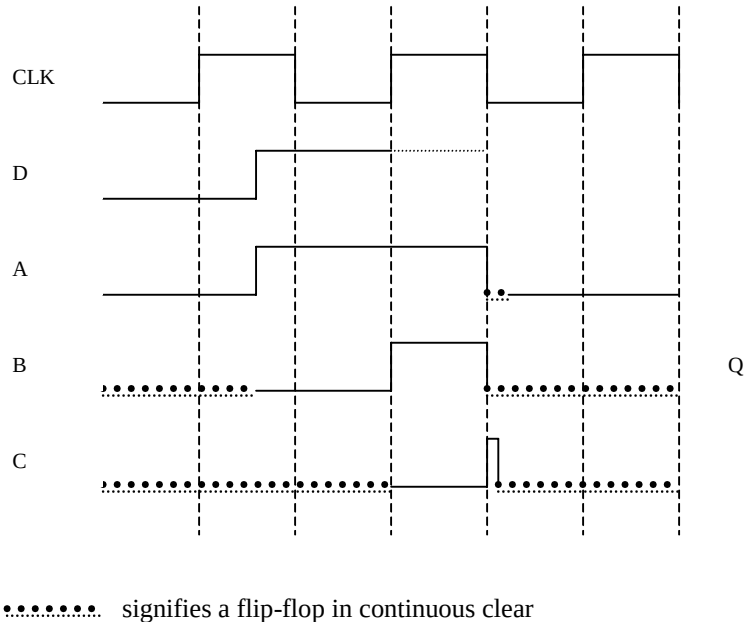
7.2.4 Pixel count

An ordinary counter counts the sampled pixels in the line, and lowers the EN_SAMP when the full number of pixels is reached.

7.3 Auxiliary components

CLK_DELAY – this component creates bursts of its input clock CLK, for the duration of the input CTRL, only delayed by a fixed number of clock periods. CLK is assumed to be a continuous clock This component is used to create a clock burst delayed by four clock periods, which is needed to latch the output from the A/D converter's pipeline.

HCP (half-clock pulse) – passes to Q the first high half-period of CLK, following the rising edge of its input D, and only that. It is used to convert the term-count signals (which last a full clock period) into half-period pulses. The component is asynchronous, and uses three FF's (A,B and C) which mutually clear each other, according to this timing diagram:



Picture 23: HCP timing diagram

7.4 Signals

Sync signals are active low, inverted and used as active high through the design. In order to process still-frame output from VCRs, which lacks the odd/even signal, this signal is generated internally.

LINE_SYNC - active high delimiter between video lines, typically 4.7 microseconds. Suppressed during the odd field.

VSNC - active high delimiter between fields, typically 230 microseconds. Suppressed during the odd field.

VERT_SYNC - derivative of VSNC. Extends from the beginning of VSNC to the first sampled line in the field, covering vertical retrace and vertical zooming delay. Covers up unused synchronization interval in LINE_SYNC.

SCAN_LINE – active high during each scanned line; reflects the vertical sampling density (every line or every third line, set by the zoom level). Its derivative SCAN_L1 is low during horizontal syncs. Both signals are active only during even fields.

EN_SAMP - when this signal is high, pixel sampling of the video line is permitted. This signal is high in every n-th video line, as set by SCAN_LINE, and covers either the entire line or the middle third of it, as set by the zoom level.

1_7_MHZ - clock which controls the density of pixel sampling of the video lines. At 108 pixels per line, this clock runs at 1.83 MHz or 5.49 MHz, depending on the zoom level (see clock-reducing counter).

BP_END - the single pulse indicating the end of back porch. Its delayed derivative, LATE_BPE, is used to latch the black reference level. Unlike the rest of the sampling signals, these are not affected by the zoom.

AD_CLK - triggering signal sent to the A/D converter. It comprises BP_END and the 1_7_MHZ line sampling burst covered by EN_SAMP.

LATE_CLK – line sampling burst, delayed by several periods (4) of the sampling clock, to allow for pipeline delay in the A/D converter. This signal clocks the utilization of the digitized signal.

DLY_END – single pulse indicating the end of horizontal zooming delay.

8. DIGITIZING AND THRESHOLDING (schematic ANALOG_IN)

Video signal from the camera is digitized with the AD876 converter chip. The circuit generates the sampling trigger for the converter, AD_CLK, and receives 8-bit grayscale signal, VIDEO, in return.

Video signal is corrected for the intensity fluctuations by subtracting the black reference value from it. The corrected video is thresholded to a strictly black and white signal, C_BIT, which is both stored in the RAM and passed to frame comparators.

The corrected video is also passed to the back-end FPGA for histogram/threshold calculation, during the even field. In the odd field, the signal THR_RDY from back-end latches the calculated threshold into a data register, to be used in the next frame.

Presently, the sampling is done on 81 lines of every even field of the video signal, at 108 pixels per line. This yields a 108x81 b/w digitized image frame, at the correct NTSC width-to-height ratio of 4:3.

8.1 Black-and-white inversion

Which side of the threshold is considered active, or a feature, is a matter of convention, and can be set by a run-time parameter. The choice does not affect the edge detection, although it affects the sensitivity of the motion detector somewhat.

9. SETTING THE THRESHOLD AUTOMATICALLY

9.1 Heuristic procedure

This vision system operates on the assumption that the scene consists of relatively luminous target feature(s) and relatively dark uninteresting background (or the reverse). An early and important step is to set a black-and-white threshold that will separate the features from the background, greatly reducing the complexity of the scene.

Setting this threshold manually is a delicate task. The selection is guided by the apparent simplicity of the b/w image, and once set, the threshold usually works well for a range of similar images. It would be difficult for the navigator to adjust the threshold in flight, and small errors in the threshold can alter the result dramatically. Automatic thresholding was implemented to make the vision more robust.

Finding the threshold follows this heuristic procedure:

a) Construct the grayscale histogram of the image. This is a straightforward pixel count, accumulated in an array of 256 grayscale levels.

b) Find the highest maximum in the histogram and assume that it is located in the middle of a large "hump" representing the background.

c) Find the lowest minimum on one side of the highest maximum (in this case, the brighter side). Set the threshold to that grayscale level: the area opposite (brighter than) the background hump represents features.

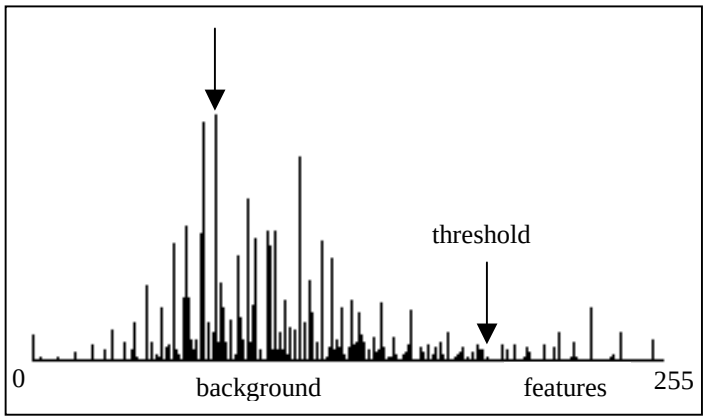
d) Disregard threshold choices which define a very small feature area in the histogram, since they typically have no visual significance.

e) If the search for a meaningful minimum fails, reverse the grayscale and look for features on the opposite end of the histogram in the next video frame

f) Clear the histogram.

The assumption here is that the lowest minimum gives best separation of the histogram into background and features of interest, and the success ultimately depends on the grayscale separability of the image. Picture 24 gives an illustration of the procedure.

The grayscale version of the image is not currently used in the later vision stages; neither is the value of the threshold. For that reason, the histogram/threshold process is implemented in hardware, on the back-end XC4010 connected to the front-end vision processor via a dedicated data bus. While the histogram/threshold algorithm is well suited for implementation in code, running it on the ColdFire processor would have complicated the data flow and slowed down the feature recognition.



Picture 24: Sample histogram

9.2 Description of the threshold calculation on the back-end FPGA

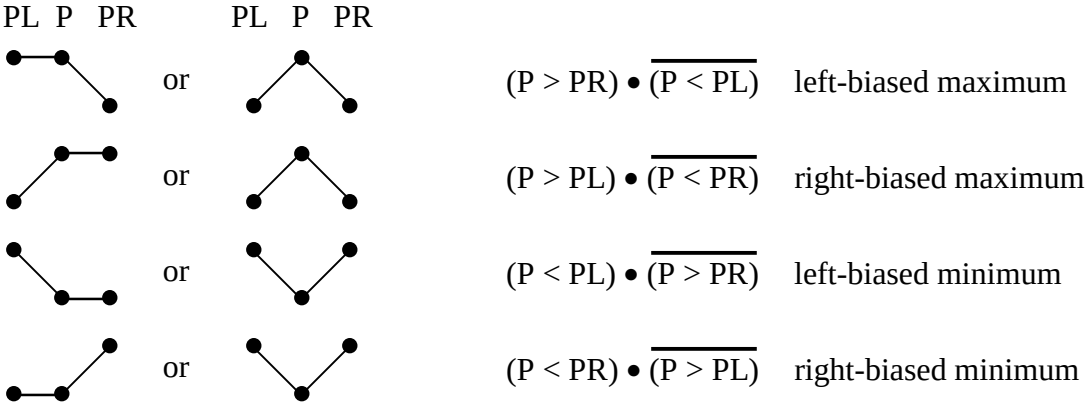
9.2.1 Data path

The histogram is stored in a synchronous RAM component, SYNC_RAM, which is contained in the schematic HIST_IO_RAM, along with elements which build (and clear) the histogram. There are 256 word-sized locations in SYNC_RAM; the histogram is maintained by presenting the grayscale value to the RAM as the address, and incrementing the corresponding location by one (or setting it to zero).

During the histogram build (even field), histogram addresses are the video data coming from the front-end FPGA through the bus HIST_IO. Depending on the nature of the image, these grayscale values may be inverted by subtracting them from 255. During the threshold calculation (odd field), HIST_ADDR is generated internally by a counter, and histogram values appear on the bus HIST_OUT.

The threshold calculation sweeps the histogram twice, by convention in downward direction (255 to 0, white to black). The sweep of histogram addresses is generated by the counter C(P); specific count values are latched in MIN_LOC and MAX_LOC registers, as the locations of histogram extrema (schematic HIST_DP).

Minima and maxima are detected by comparing three adjacent histogram values, which flow through the comparison registers PL, P and PR during the sweep. These are the relevant comparisons, with black dots representing relative heights of the adjacent bars of the histogram:



We use the left-biased comparisons. Current extremes are latched into registers CUR_MIN and CUR_MAX. Since we are interested in the global extremes, locally found extremal values must be compared with current largest/smallest values. All the comparison logic is contained in the schematic HIST_MINMAX.

Step d) from previous section is implemented in the schematic HIST_INTEGRAL. Histogram integral and the features integral are accumulated in registered adders, and the features integral is compared with an appropriate fraction of the total histogram integral.

Since the search for a meaningful minimum can fail, the success/failure is recorded in the flip-flop MIN_FOUND and passed to the control ASM.

9.2.2 Control

Control ASM is implemented in the Abel code component HIST_ASM, shown in Picture 25 (two pages). The algorithm is by its nature sequential, and can be roughly divided into these six steps: initialize the address and the comp registers, search for the maximum, re-initialize, search for the minimum, notify front end or invert the grayscale, clear the histogram. Individual states and logic are explained in the ASM chart.

9.2.3 Signals

CP_SYNC_LD – synchronously load the value 255 into the address counter C(P).

CLR_REG – synchronously clear registers PR, CUR_MAX, MAX_LOC.

PL_LD, P_LD, PR_LD – enable loading of comp registers PL, P and PR.

CUR_MAX_LD, MAX_LOC_LD, CUR_MIN_LD, MIN_LOC_LD – enable loading of maximum/minimum registers.

PR_ASYNC_LD – asynchronously set registers PR and CUR_MIN to “infinite” value.

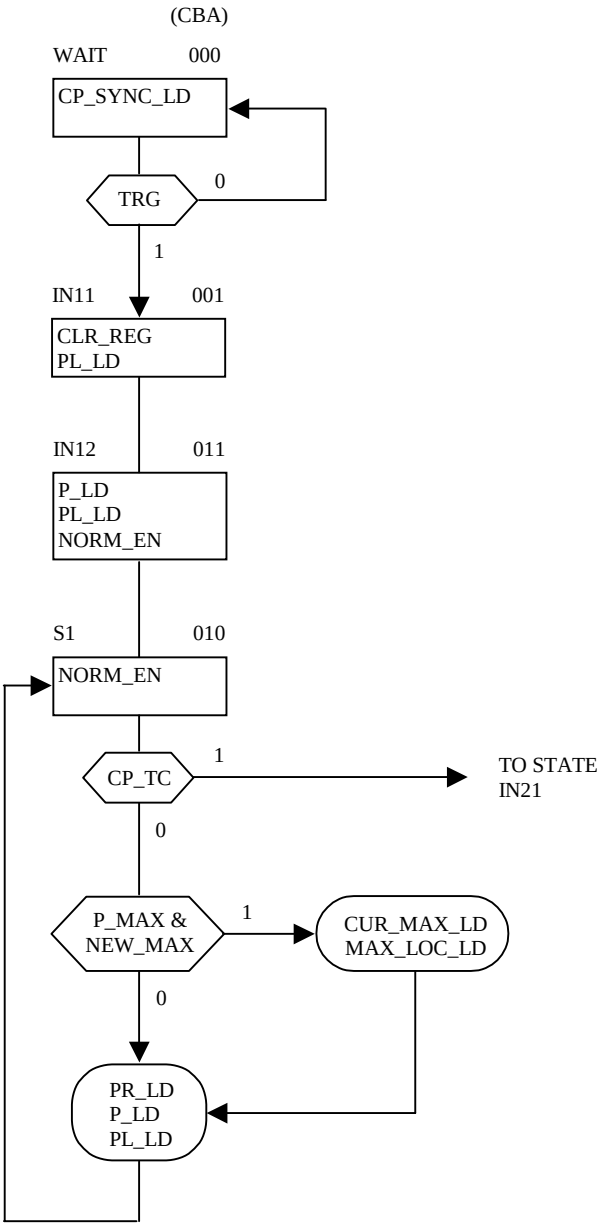
THR_RDY – notify the front end that the threshold is ready.

INV_GRAY – set the grayscale inversion for the next frame.

CLR_WE – enable writing zeros into the histogram.

NORM_EN, FT_EN – enable the accumulation of histogram and feature integrals.

FFR – feature integral is meaningful relative to the entire histogram area.



Keep the histogram location at 255 [C(P) = 255]

Start the threshold calculation

Clear registers CUR_MAX, MAX_LOC, PR.
Load histogram data at location 255

PL -> P; DATA(255) -> PL
Completed initialization of PL, P, PR for max. search.
Calculate the area of the histogram.

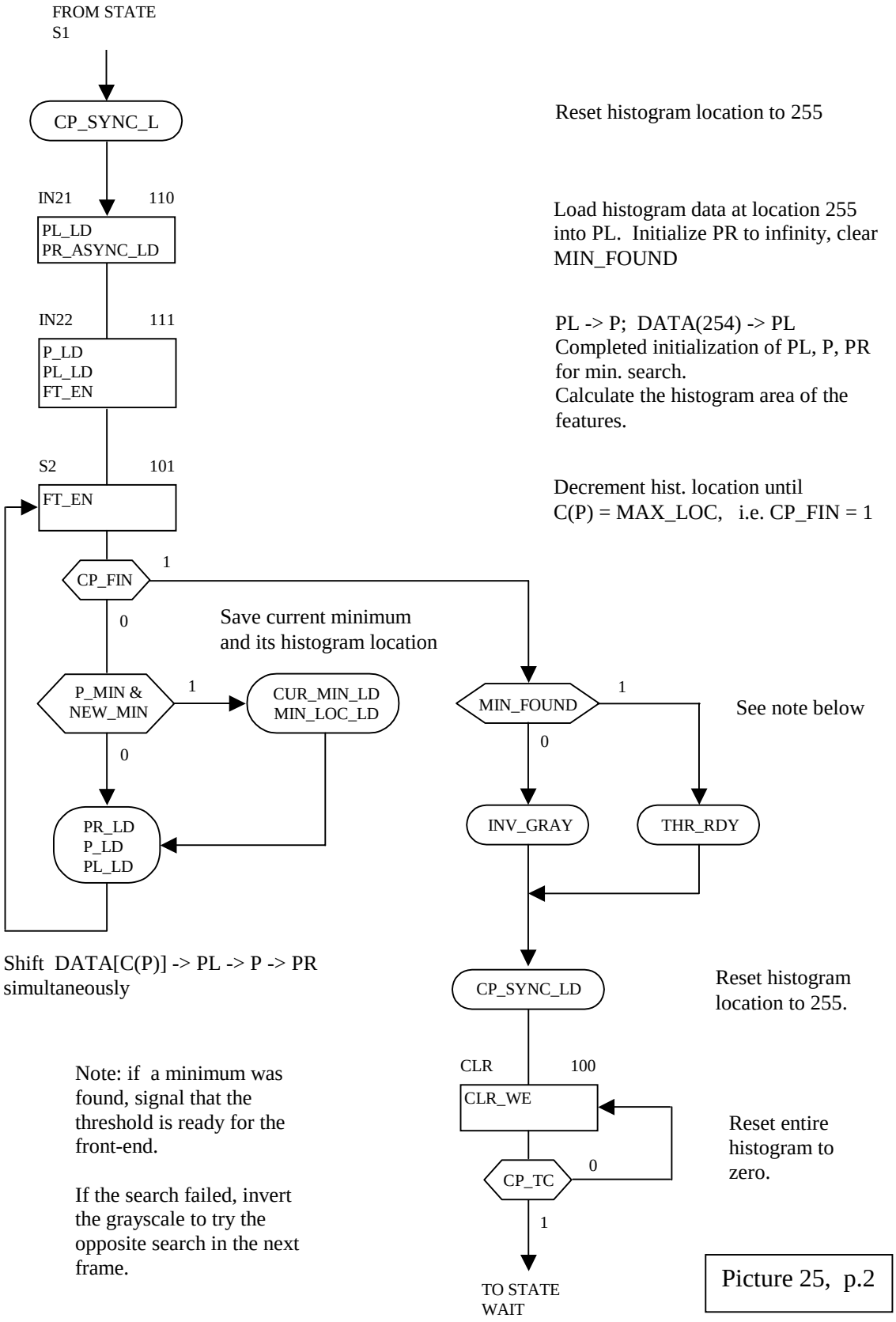
Decrement hist. location until C(P) = 0, i.e. count down until CP_TC = 1

Search for the maximum finished; start the minimum search.

Check if current value is a maximum (P_MAX true), and if it is larger than earlier maxima (NEW_MAX true). Save current maximum value and its histogram location.

Shift DATA[C(P)] -> PL -> P -> PR simultaneously

Picture 25: Control ASM for the histogram/threshold calculation, p.1



Picture 25, p.2

10. DIGITAL ZOOM

It became very obvious during test flights that the human navigator and the vision steering system need two different perspectives on the ground scene. At altitudes of 200-300 ft, view through the wide-angle camera was adequate for general orientation, but the target was impractically small for the vision system to handle. Using a narrow-field (f=16 mm) lens, or equivalently, flying close to the ground, produced good target images if and when the target was ever located. The camera was fixed to the body of the plane, but even with an independently movable camera the navigator would have difficulties spotting areas of interest through the narrow field.

A camera with the zoom lens, or even two different cameras on the same gimbal, would solve this difficulty at the cost of additional mechanical equipment. However, since the vision system originally used only one sixth of the total image information (sampling every third line of the even field), there was room for electronic, instead of optical, zooming. Only the vision system sees the effect of the zoom, since the navigator currently receives no digitized image feedback.

The digital zoom, as implemented, is a 3X zoom. It amounts to using the full line density of the even field, sampling pixels at three times the "no zoom" rate. In order to maintain the same data volume, only the central one-ninth of the video image is actually digitized and passed to the vision system. A 6X zoom could be implemented by using the lines in the odd field also, but some care would have to be exercised regarding the end-of-field communication between processes.

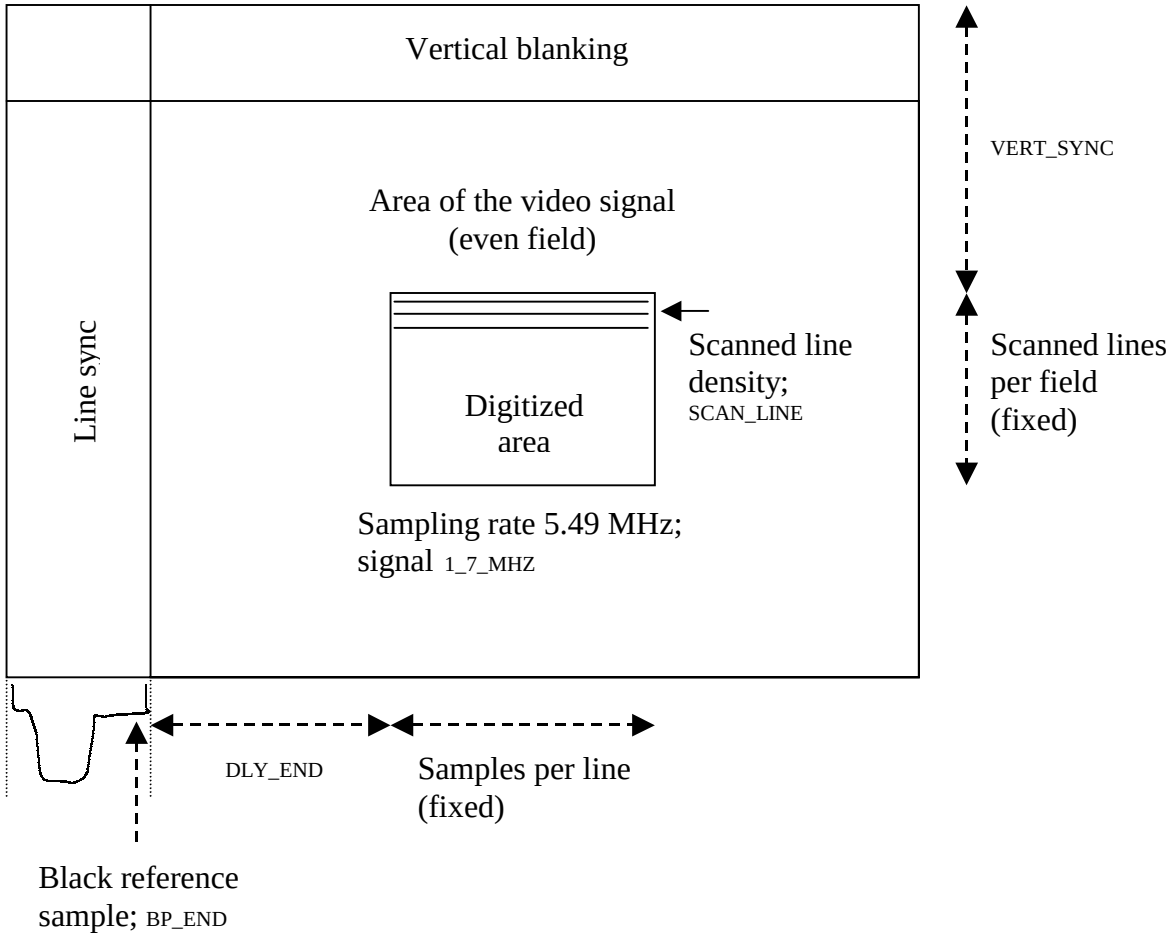
10.1 Zoom implementation on the FPGA

The zoom has been implemented on the front-end FPGA, as part of the overall digitizing circuit. The zoom level is passed to the FPGA as a run-time parameter, and a selection is made between two sets of five constants. These five constants define the resolution and framing of the image.

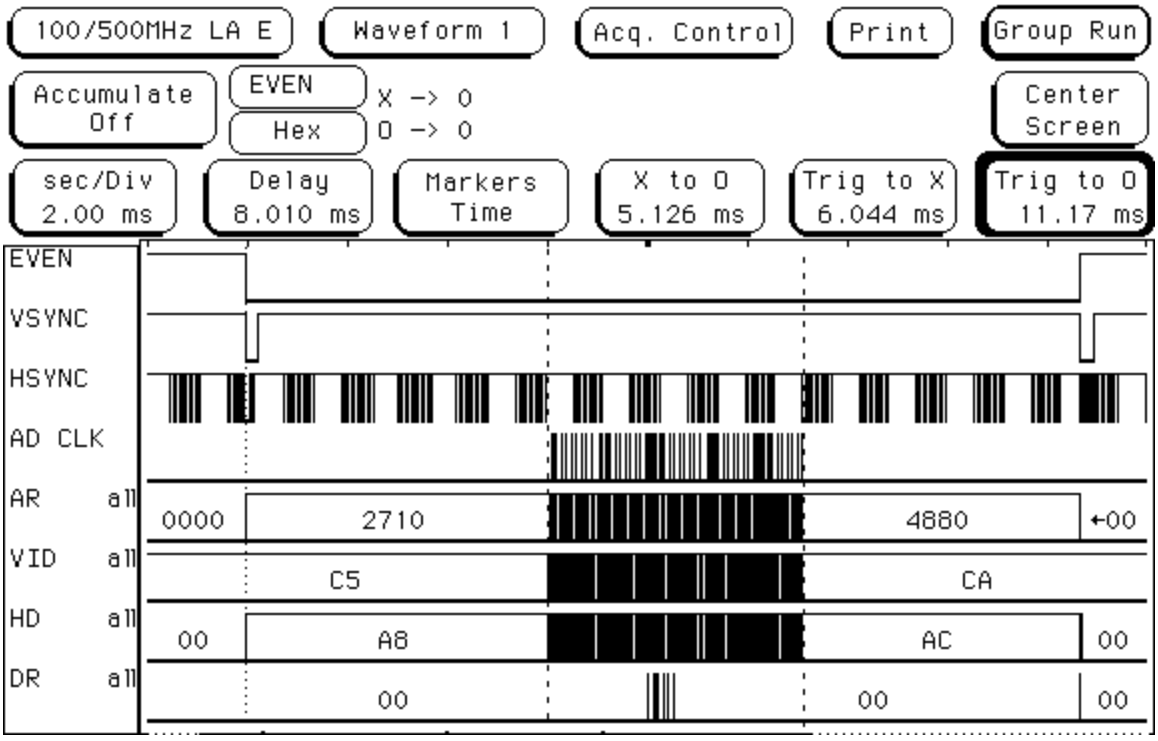
In order to cope with high sampling frequency, the FPGA's clock rate is set to 33.3 MHz, and the pixel read/write cycle was made as short and pipelined as feasible (see description in Section 12). Picture 26 shows the zoom's geometry and the signals involved. Refer back to Section 7, Formatting the image scan, for details (p. 33).

Pictures 27 and 28 give an overview of the formatting, modified by the zoom. Picture 27 shows the sampling clock active in the central one-third of the lines of the even field. Greater magnification shows the sampling clock also limited to the central one-third of one line, with the black reference sample following the horizontal sync (Picture 28). The non-zero data signal (DR) is due to a dark object in the camera's field of vision.

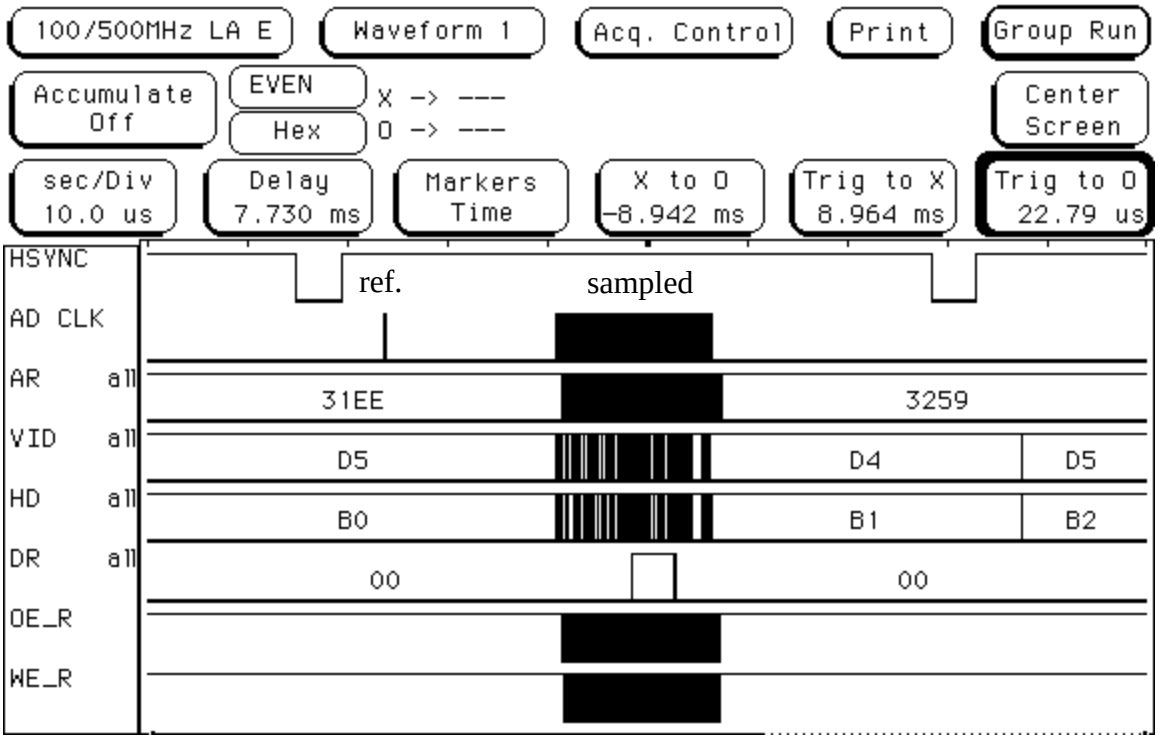
When the zoom is engaged, rotations of the camera produce larger displacements in the image. Therefore, the procedure that calculates servo duty cycles must also take the zoom into account and turn the camera by smaller angles (see the program module Servo.s).



Picture 26: Formatting and zoom



Picture 27: Vertical formatting (zoom)



Picture 28: Horizontal formatting (zoom)

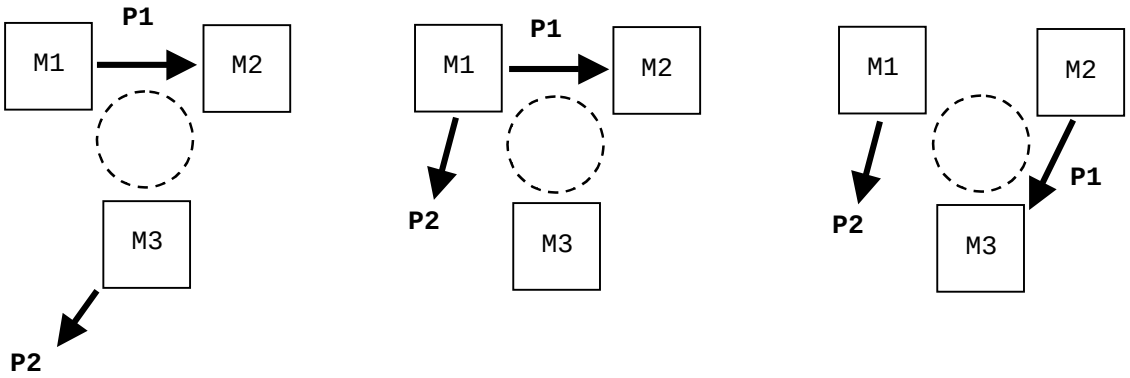
11. ROUND ROBIN PROCEDURE FOR DATA SHARING

Presence of two processes (motion detection and feature recognition), running on separate processors, makes heavy demands on the memory containing the image data. In this system, access conflicts and bus logjams are avoided by using a dual-port SRAM chip and a round-robin data access procedure.

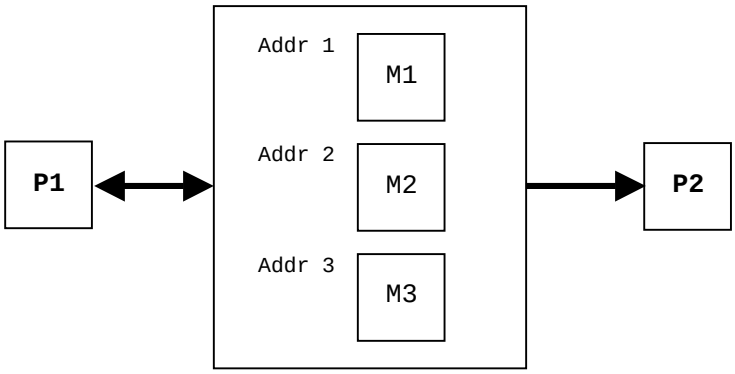
The RAM chip used is CY7C007AV,[7] an asynchronous 32K x 8 part by Cypress Semiconductor. It has two address/data ports, which can read simultaneously from the same memory location. The part arbitrates read/write access conflicts in hardware, although that feature is not used here. The chip also has a bank of hardware semaphores, with their own chip-select signals and arbitration logic. This feature was essential in implementing the round robin procedure.[8]

In this scheme, the motion detection (process P1) reads from one memory area, say M1, and writes into another (M2). It swaps these areas on each new video frame.

Feature recognition (P2) takes a few frames' time to complete one calculation. When P2 needs an update, it reads the P1's read frame, say M1 (P2 never writes). On the next video frame, P1 reads M2 and writes to M3, then swaps M2 and M3 until P2 claims whichever of these is P1's read frame at the moment (see Picture 29).



Picture 29: Round robin



Picture 30: DPRAM's memory buffers

Buffers M1-M3 are implemented as distinct memory areas on the dual-port memory chip, and waiting is eliminated completely. P2 can start reading the P1's read frame through its own bus, and the round-robin motion is performed by switching the starting addresses of M1-M3 (see Picture 30). Read/write conflicts cannot occur on buffer access, only double reads, which are permitted by the DPRAM.

At any moment, each buffer is assigned to one of these three states: P1 writes, P1 reads, P2 reads; any buffer can be in any of them, and no two buffers are ever the same. The record of the current state is maintained in a dedicated location, the status byte, which is updated by P1 and P2 on every turn of the round robin.

Since P1 and P2 are mutually asynchronous, genuine access conflicts will occur on the status byte. These are avoided by protecting the status byte with the semaphore: only one port can hold the semaphore (this is arbitrated by the memory chip), and that port updates the status before releasing the semaphore. The other process stays in a polling loop until access is granted, but the duration of the busy wait is no more than a one-byte I/O operation, which is insignificant on either processor.

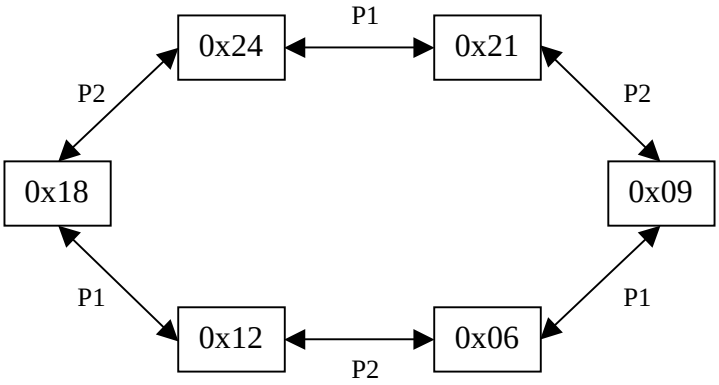
11.1 Status byte

The status byte, at the DPRAM address 0x08, contains three two-bit fields corresponding to the buffer states P1W, P1R and P2R, and the value in each field is the number of the buffer assigned to that state.

Maintenance of the status byte is very simple. On reset, its value is set to 0b00100100 (0x24), which means that:

- buffer zero is P1's write buffer
- buffer one is P1's read buffer
- buffer two is P2's read buffer

Process 1 swaps the contents of fields P1W and P1R (interchanges its working buffers). Process 2 swaps the contents of fields P1R and P2R (releases the buffer it just read and takes up the reading buffer of the Process 1). Between updates, each process maintains a private copy of the status information; otherwise, its working buffer(s) could change in mid-cycle, with disagreeable results. Picture 31 shows the allowed swaps of the status byte values.



Picture 31: Status byte values

11.2 Round Robin on the front-end FPGA

The procedure by which the two processes share buffers of image data in the DPRAM has already been described earlier. This section deals with the implementation of the round robin on the FPGA side, as the component ROUND_ROBIN.

The private copy of the status byte resides in the register SBYTE_REG, which is read from and written to the DPRAM address 0x08. Notice the peculiar ordering of input bus leads, which accomplishes the swapping of fields P1R and P1W.

Current addresses for the three image buffers reside in three counters driven by the LATE_CLK (see schematic RR_ADDRESS). Fields P1R and P1W are used to operate the multiplexer which selects the current buffer for read or write operations.

The ASM (see picture 33) is straightforward: it polls the semaphore for access, reads and writes the status byte, then releases the semaphore. Picture 34 shows the timing diagram.

SEM_IN – read value of the semaphore.

SEM_OUT – written value of the semaphore.

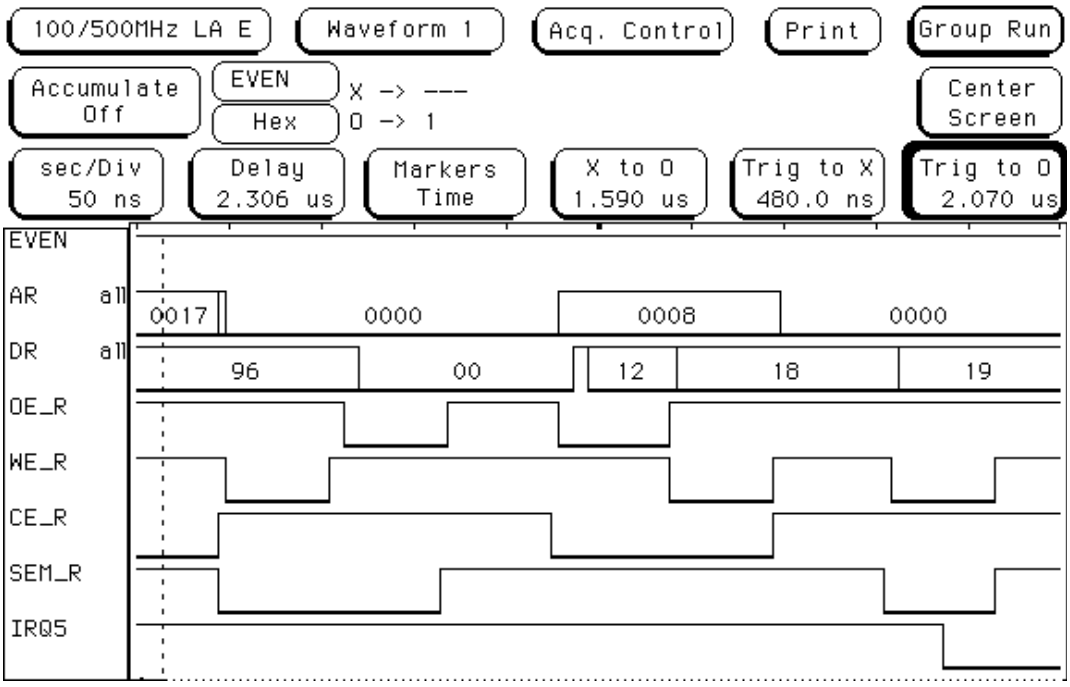
RR_SE – semaphore enable (the semaphore's chip select).

SEM_WE, SEM_OE, SEM_D_TSB – control signals for the semaphore I/O.

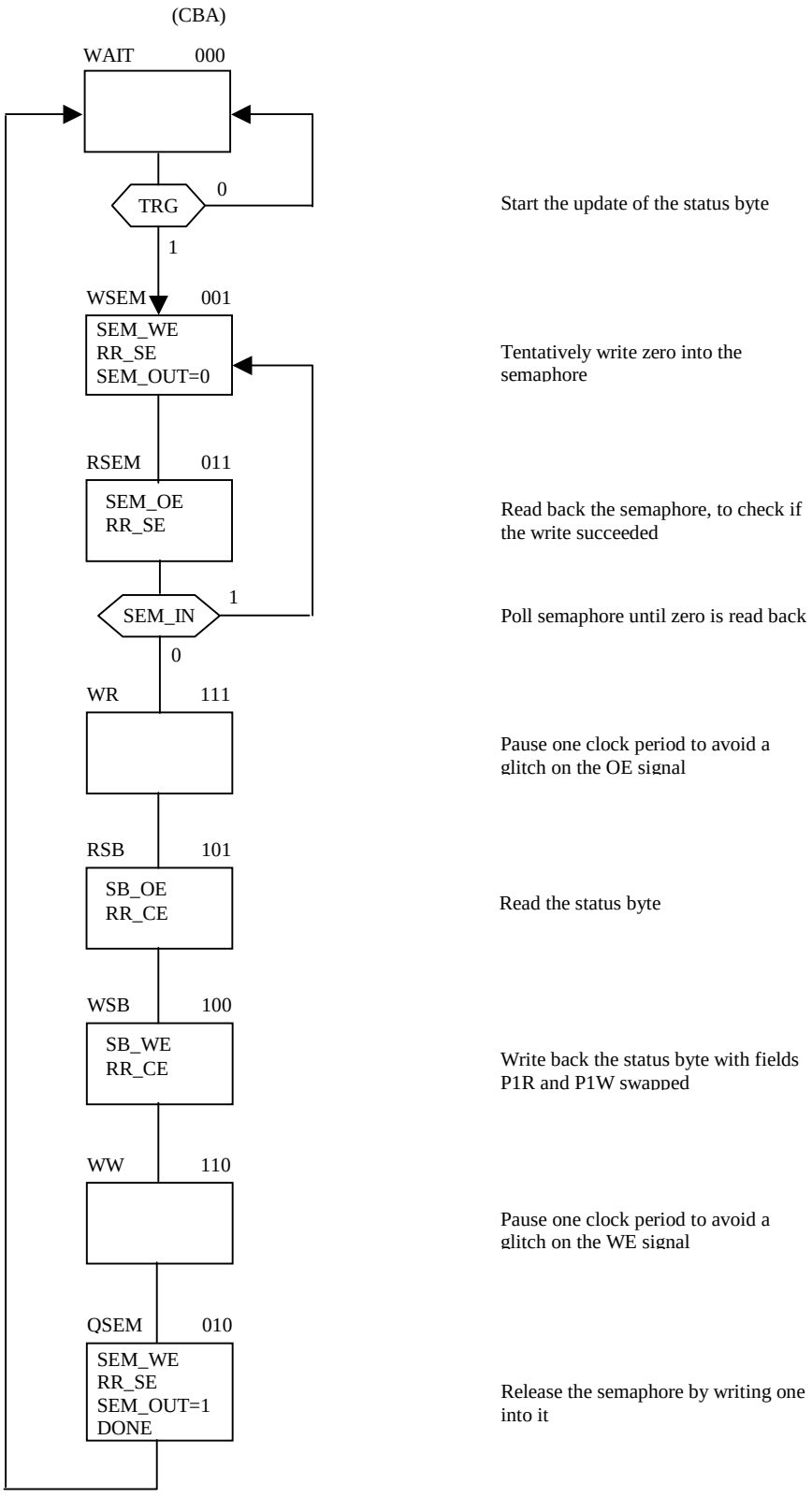
RR_CE – chip enable for the regular RAM area.

SB_WE, SB_OE - control signals for the RAM I/O.

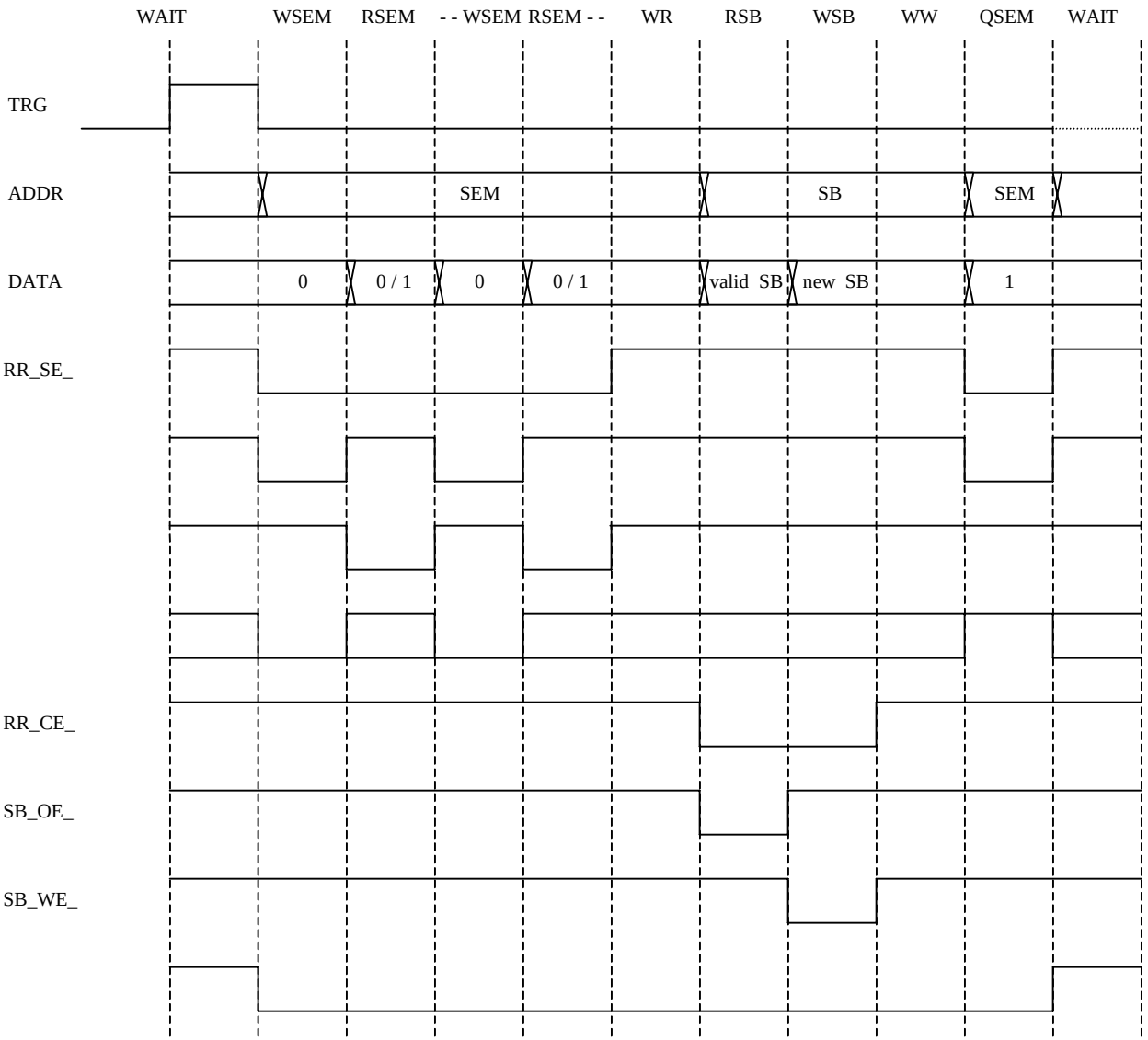
STAT_A_TSB – address TSB control for the entire round robin sequence.



Picture 32: Semaphore-protected update of the status byte



Picture 33: Round-robin ASM



Picture 34: Round-robin timing

p. 62

11.3 Round Robin on the MCF5307 processor

Implementation of the round robin procedure in code is straightforward: the flowchart is almost identical to the ASM chart in Picture 33. The procedure is invoked once in each pass of the feature recognition loop (see module main.c). It polls the semaphore for access and reads the status byte when the access is granted; swaps the fields P1R and P2R, updates the status byte and releases the semaphore. The new P2R (in the local copy of the status byte!) is used to select the starting address of the read buffer, and that address is made available to the vision code in the global variable IMAGE_FRAME.

The subroutine round_robin is contained in the program module RoundRobin.s, along with the subroutine config_cs, which configures the left port (ColdFire side) of the DPRAM.

Note on the DPRAM addresses on MCF5307: The hardware is configured so that the ColdFire processor uses its chip select 4 for the semaphore bank of the DPRAM, and the chip select 5 for the regular storage area. ColdFire chip selects are assigned blocks of address space 2 MB in size;[9] consequently, the base addresses for the semaphores and the regular RAM become 0xFF800000 and 0xFFA00000 respectively, even though they are contiguous in the DPRAM's address space (see Picture 48 on p. 98). ColdFire generates the proper address in the lower 15 bits, and the high bits serve only to activate the right chip select.

12. THE PIXEL READ/WRITE CYCLE

The pixel read/write cycle is central to the early vision processing: it stores the digitized B/W image and performs the motion detector's frame comparison. This section describes the cycle suitable for the 33.3 MHz system clock and the 5.4 MHz pixel sampling rate.

Events within the cycle are sequenced by the state machine PIXEL_CYCLE, which is clocked by the system clock and runs one full sequence per period of the sampling clock.

The pixel cycle accesses two memory buffers, P1R and P1W; corresponding addresses of the pixel in these two buffers are determined by the round robin algorithm. Pixel calculation is triggered by the delayed sampling clock, LATE_CLK, which also increments the buffer addresses.

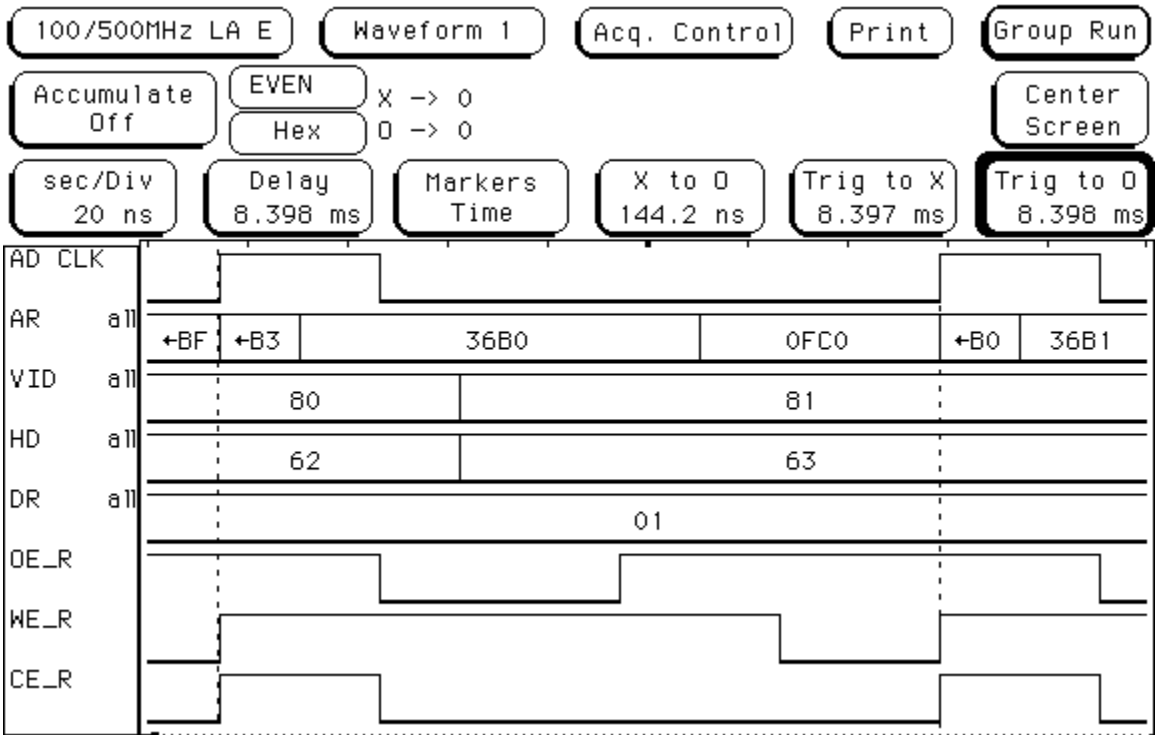
The pixel's read address is calculated during the high time of the sampling clock, and the value is stable on the signal M_BIT one clock period later; this is the pixel's value from the previous frame. The thresholded signal, C_BIT, becomes available around the same time. Both bits are presented to the comparator/accumulator (see Section 13.3, the description of the FPGA design, p.70, for details) and the incremented value of the motion vector is clocked in one period later.

The address is now switched to the pixel's write address, and on the rising edge of WE, one clock after the address switch, the new pixel value is written to the write buffer. The

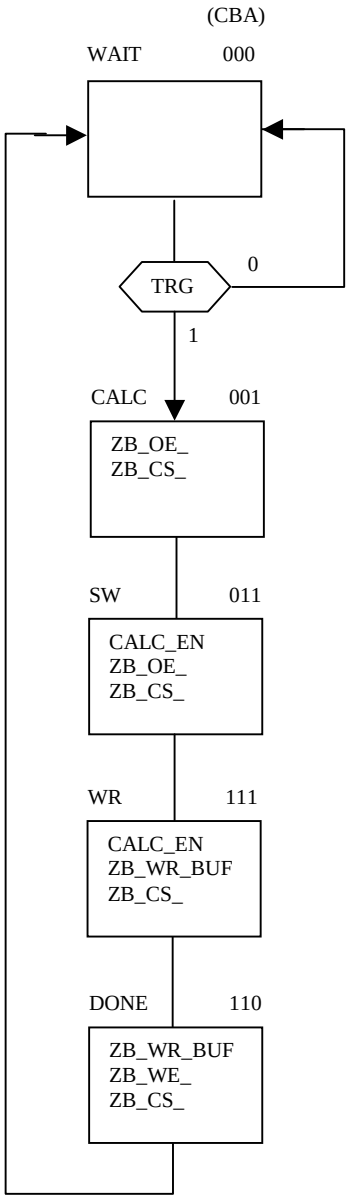
entire read/write cycle takes five clock cycles: at 33.3 MHz, this is sufficiently fast to complete all pixel processing at the sampling rate.

In addition, there are RAM-controlling signals in the cycle: output enable, write enable and the three-state output buffer on the zero bit of the data line. All of these are active low. Picture 37 shows the timing diagram of the read/write cycle, one column per state.

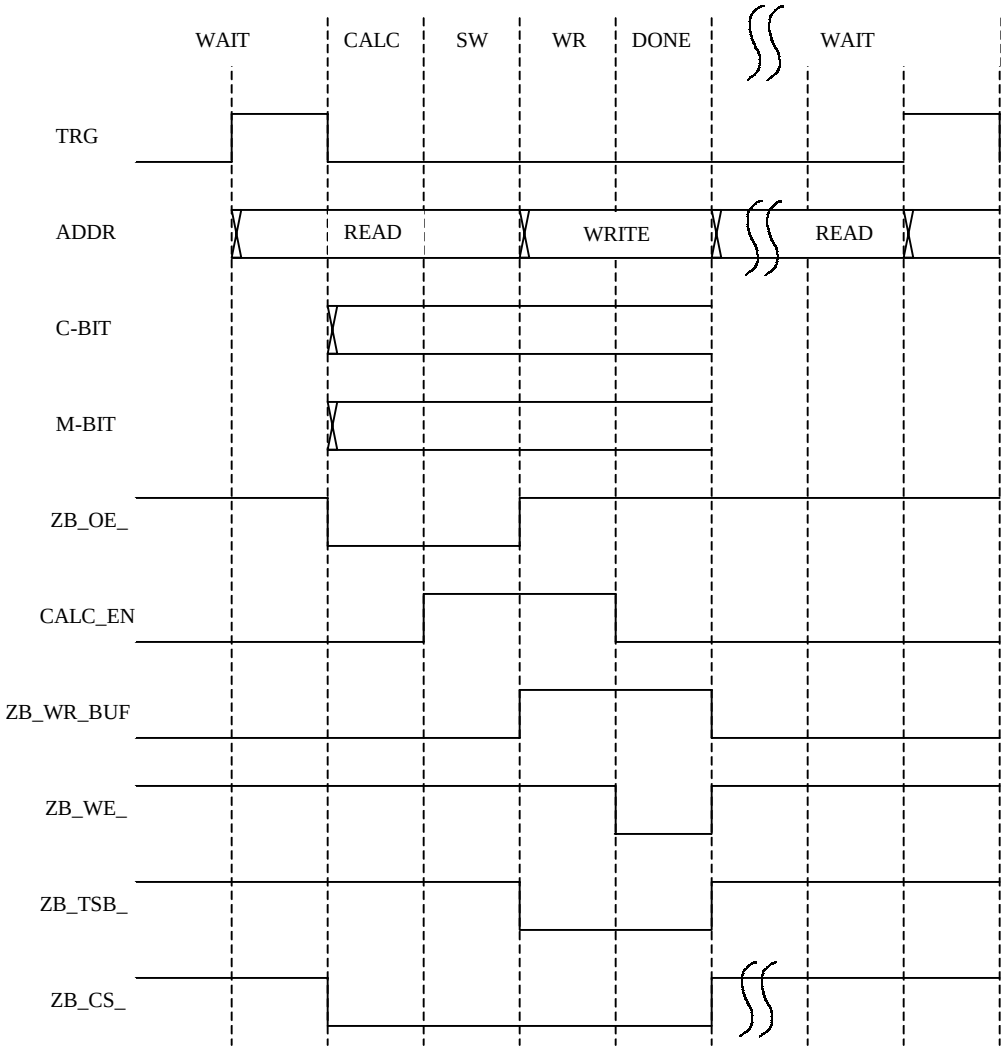
Picture 35 shows the read/write cycle driven by the 33.3 MHz clock, and running at the full sampling rate of the 3X zoom.



Picture 35: Pixel cycle



Picture 36: PIXEL_CYCLE ASM



Picture 37: Pixel cycle timing

Pull the ASM out of wait

Data bit from video stream

Data bit from memory

OE to read M-BIT

Enable the bit comparison

0-access P1R;1-access P1W

WE to write C-BIT

Open TSB to write C-BIT

CS for the pixel cycle I/O

13. FRAME COMPARISON AND THE MOTION VECTOR

13.1 Methodology

Motion detection in this system is limited to linear displacements of the entire scene, since we are interested only in detecting changes due to the movements of the plane (ego-motion). Drift detection would perhaps be a more accurate term.

Formula used to calculate the motion vector is as follows:

$$\Delta = \frac{\sum x_i (\Delta pix)_i}{\sum pix_j}$$

This is essentially the formula for the dipole moment of the displacement, with the previous frame's pixels counting as the negative charge, and the current frame as positive. Summations are over the entire image, pix is the pixel value (zero or one), Δpix is the difference between consecutive frames, and x_i is the i-th pixel's position. The normalization constant in the denominator is simply the number of black pixels in the image.

Motion detection in the vision of insects with composite eyes utilizes the principle of consecutive activation/deactivation of receptors; direction of motion is determined by the pattern of neural wiring between adjacent receptors (eyelets in the composite eye).[10] [11] Our detector has no pixel adjacency information, and cannot detect local motion within the image. Instead, it obtains an integrated value of spatial distances between activated/deactivated pixels. For simple drift motion, this is an adequate motion vector,

with the caveat that the detector is sensitive to appearance of new objects in the periphery, which it interprets as motion.

13.2 Computation

Pixels are processed in real time, and each pixel cycle contains the following steps:

- corresponding pixel from previous frame is read from the DPRAM buffer P1R;
- current and previous pixel are presented to two comparators, which calculate the components of the motion vector;
- current pixel is stored in the DPRAM buffer P1W.

Each b/w pixel is stored in the zero-th bit of a byte, which makes addressing simpler and faster. Higher bits of these bytes are not used.

13.3 Design components

line-in-frame counter - counts scanned video lines within one frame. Used as the vertical (Y) coordinate of the current pixel.

pixel-in-line counter - counts the sampling clock (LATE_CLK), starting at the beginning of each video line. Used as the horizontal (X) coordinate of the current pixel.

pixel-in-frame counter - counts the sampling clock (LATE_CLK), starting at the beginning of a frame. It resets to the starting address of the frame in the SRAM, and its value is used as the address of the SRAM byte that contains the current pixel.

COMP_ACCUM - comparator/accumulator; this component adds/subtracts the value on the input bus NUM[31:0] to/from the current value in its internal register. The current value is always available on the output bus SUM[31:0]. The sign of the operation depends on the values of CBIT and MEM, as follows:

CBIT	MEM	operation
0	0	none
1	0	add
0	1	sub
1	1	none

This operation is designed to capture the differences in pixels of adjacent frames. It is enabled by the EN signal. ASYNC_CTRL resets the register value to zero asynchronously, and no operations take place while ASYNC_CTRL is high.

black pixel counter - counts the sampling clock (LATE_CLK), starting at the beginning of a frame, only if C_BIT is high on the rising edge of the clock. Count of black pixels in one frame.

13.4 Signals

C_BIT – single-bit output of the digitizing/thresholding circuit ANALOG_IN. This is the current pixel of the current frame.

M_BIT - current pixel of the previous frame, retrieved from DPRAM and compared with the C_BIT to detect motion.

CALC_EN – enable signal for the comparison; output of the PIXEL_CYCLE sequencing ASM.

14. WRITING THE MOTION VECTOR TO DPRAM

At the beginning of the odd field, vector components and the normalization constant are written in DPRAM, at the address 0x0C, as three longwords in the big endian order. Component VECT_OUT handles that procedure.

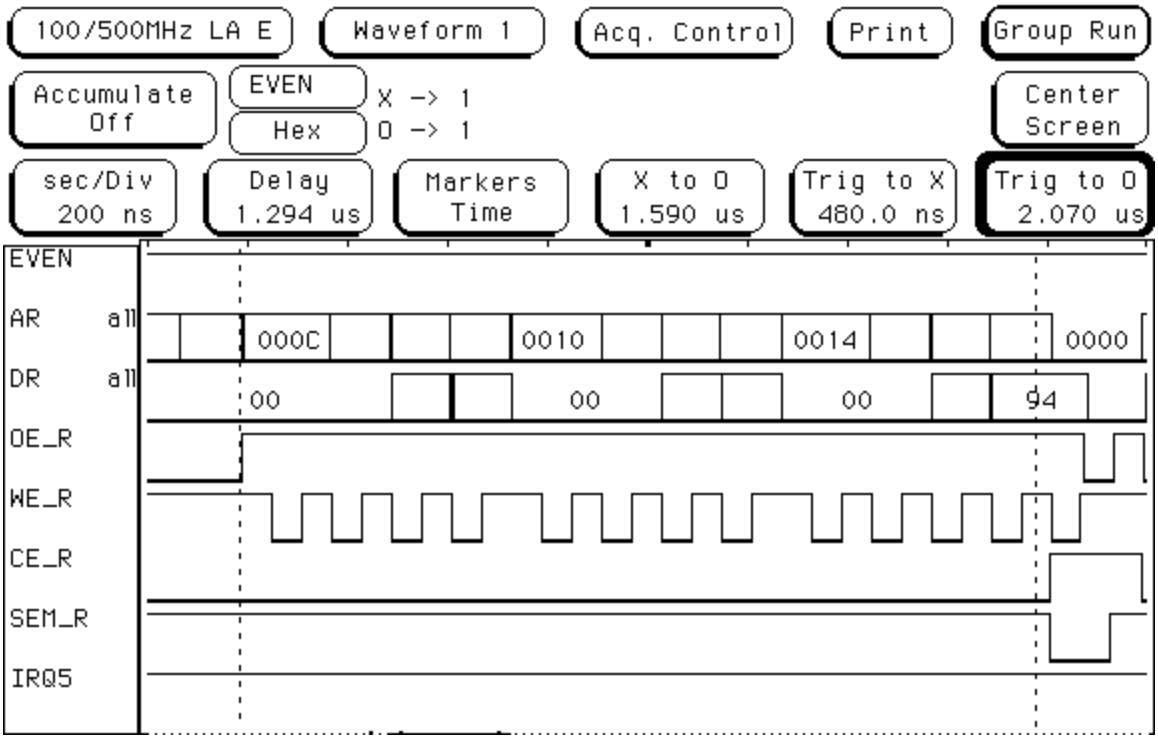
VECT_OUT has an address counter, a word counter and a byte-in-word counter. The latter two counters operate the multiplexers which select the proper byte for output, and the whole procedure consists of a straightforward double loop, corresponding to three words and four bytes per word. The ASM chart is shown in Picture 39.

TRG – the trigger signal.

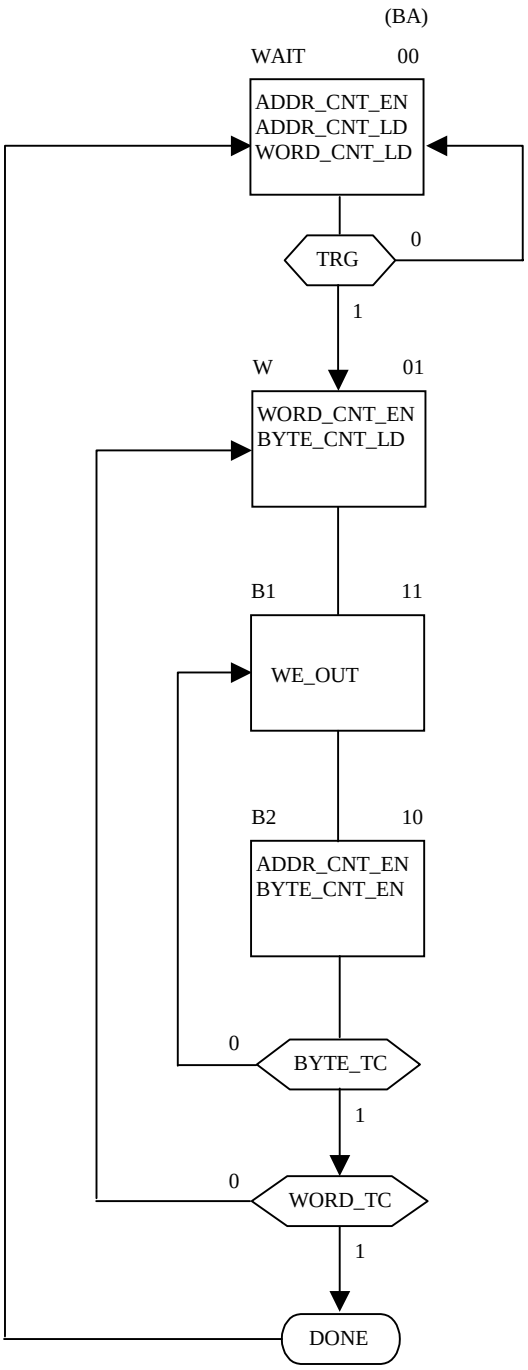
…_CNT_EN, …_CNT_LD – control signals for the counters

WE_OUT, TSB - DPRAM control signals

DONE – ending signal; signal to the next stage to proceed.



Picture 38: Vector output



Initialize the address and word counters

Start the vector output

Count the word, restart the byte count

Put out a byte

Increment the byte's address, count the byte in the word

Finished with one word?

Finished with all the words?

Picture 39: VECT_OUT ASM

15. PARAMETERS OF THE FRONT-END FPGA

The component PAR_IO1 handles the parameter logistic. Currently, there are three byte-size parameters allocated to the front-end FPGA, starting at the DPRAM address 0x09:

- b/w threshold, (0 – 255)
- one byte of bitwise parameters:
    - bit 0 - b/w inversion; zero stands for dark features, one for bright features
    - bit 1 - zoom level: zero for no zoom, one for 3X zoom
- mailbox, written to DPRAM on each frame:
    - bit 0 – zoom level indicator: zero for no zoom, one for 3X zoom

Parameters can be added as needed, by a fairly straightforward extension of this component.

Default values of the parameters are contained in the circuit, as byte-size constants. On the first high TRG after power-up, that is on the first odd field, default parameters are written to DPRAM at consecutive addresses. On subsequent TRGs, each parameter's address is presented on the bus ADDR_OUT, and the corresponding parameter selection signal goes high, for the duration of two clocks. This allows the current parameter values to be read from DPRAM and latched into parameter registers in the circuit. The mailbox parameter is written to DPRAM on each frame, to be read by the microprocessor.

The ColdFire program updates the parameters in the DPRAM during the IRQ5 handler, and reads the mailbox. This mechanism allows for changes in the parameters to be made

at run time, e.g. by operator commands, as well as for the FPGA to send signals to the ColdFire. Currently, the mailbox is used to notify the ColdFire when the zoom command has been radioed to the front-end FPGA.

Sequencer ASM for PAR_IO1 is shown in Picture 40. Its subcomponent, INIT_BOX, raises PAR_INIT once after the power-up, and its ASM is shown in Picture 41.

TRG – the trigger signal; high once per frame, at the start of odd field

TRG_MACH – derivative trigger, produced by the initialization component INIT_BOX.

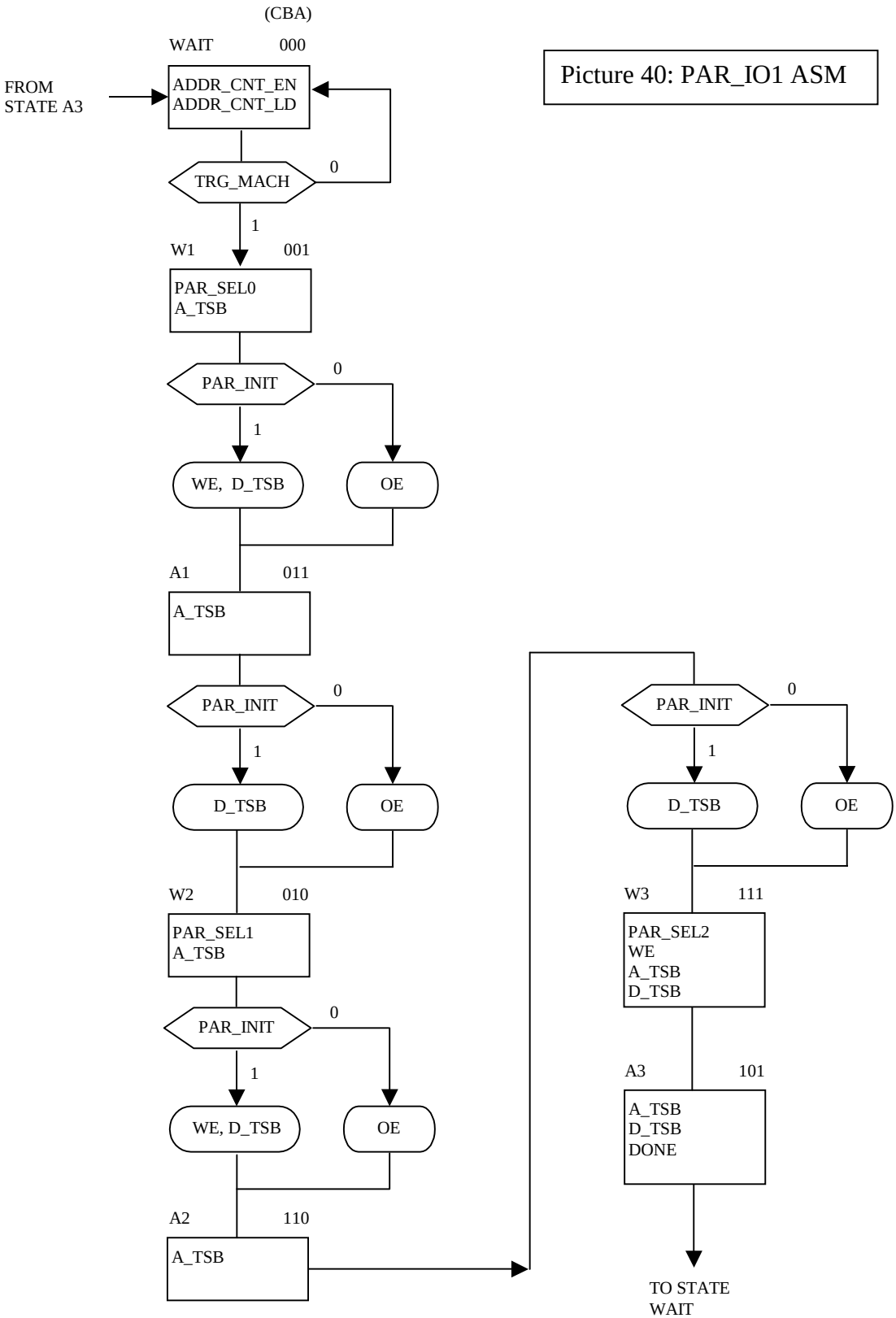
ADDR_CNT_EN, ADDR_CNT_LD – signals that control the address counter

PAR_INIT – high on first occurrence of the TRG; passes the default parameter values onto the PAR_IO bus.

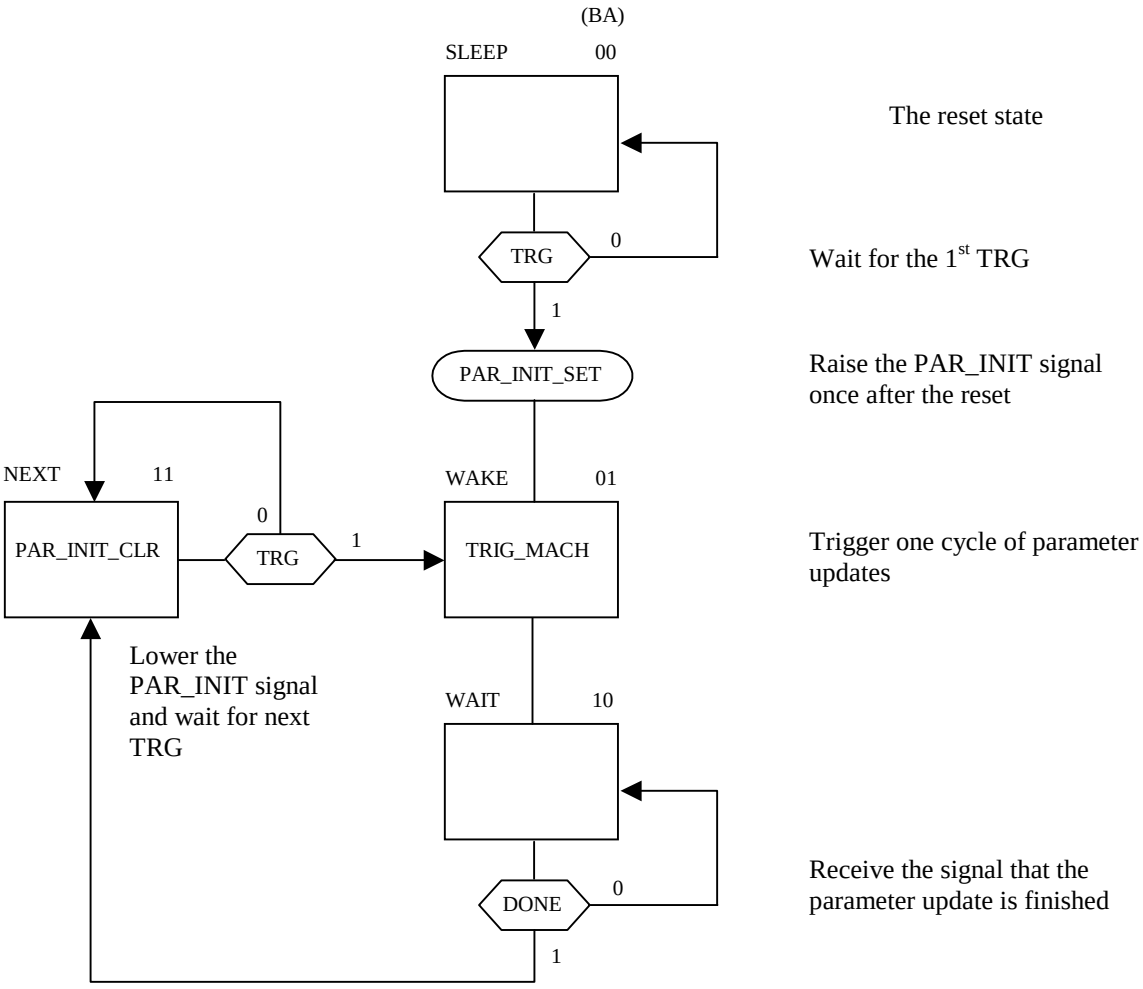
A_TSB, D_TSB, OE_OUT, WE_OUT – RAM control signals involved in parameter I/O to and from the DPRAM.

PAR_SEL0, PAR_SEL1, PAR_SEL2 – parameter selectors; enable latching of the corresponding parameter in the appropriate data register in the circuit.

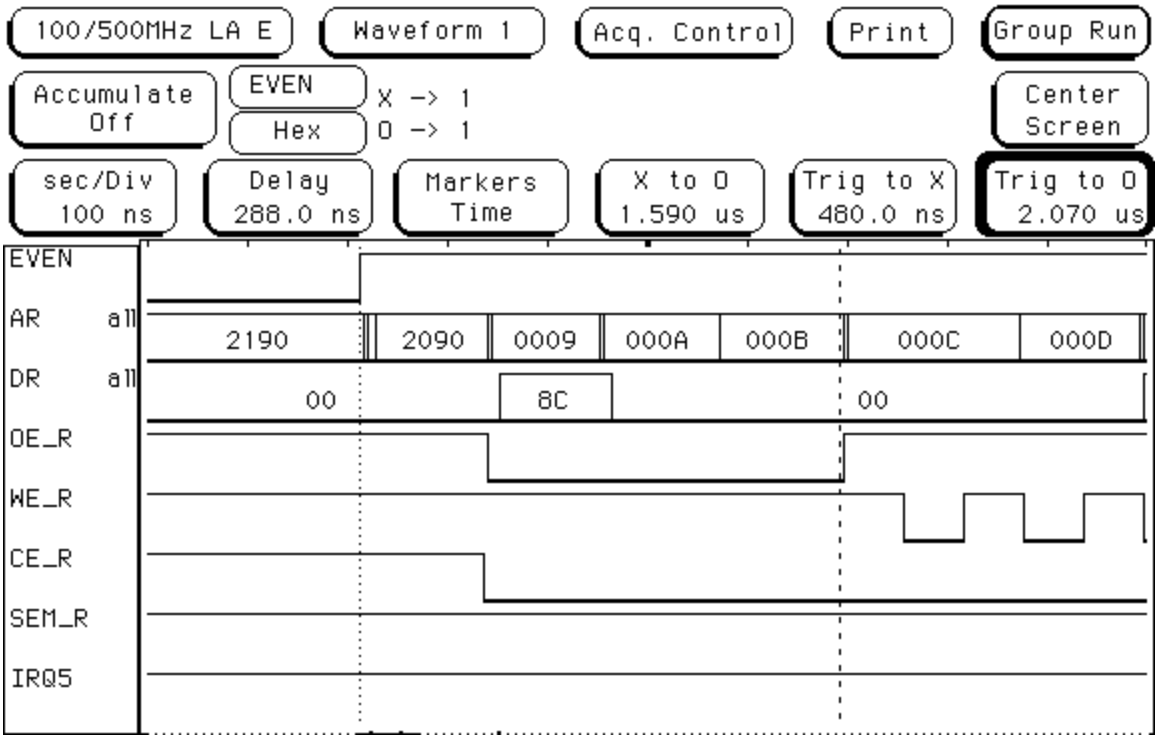
PAR_IO – data bus which carries the parameters.



Picture 40: PAR_IO1 ASM



Picture 41: INIT_BOX ASM



Picture 42: FPGA parameters

16. THE IRQ5/PARALLEL PORT COMPLEX

A tightly coupled hardware/software subsystem, centered around the Interrupt 5 and the parallel port of the MCF5307, coordinates the back-end data flow in the vision system. Here we describe that subsystem.

16.1 IRQ5 handler

Early in the odd field, after the motion vector has been written to the DPRAM, back-end FPGA generates an IRQ5, as a hardware signal to the MCF5307. When the processor enters the IRQ5 handler, the interrupt is acknowledged by a handshake on the parallel port, via two signals, ACK and RDY (bits 14 and 15).

The motion vector is read from DPRAM (it was written before the IRQ5, so there is no access conflict), and it is normalized by dividing by the black pixel count. This operation is performed here and not in the front-end FPGA, where the vector is generated, because of long integer divisions.

Subject to some size restrictions, the vector is translated into increments in the servo cycle's pulse widths, and these increments are used to update the current pulse widths. Notice that the desired position of the camera is always known to the vision system (in the form of calculated pulse widths), but that the actual position may not be known in real time.

16.2 Duty-cycle generator

The generator of the pulse modulated servo signals resides on the back-end FPGA. It receives the pulse widths from the IRQ5 handler, and produces the corresponding waveforms. Pulse widths are passed as 14-bit numbers on the parallel port, in a protocol synchronized by the ACK/RDY signals.

16.3 Servo motion feedback

When the pulse width changes, the servos start moving into the new position. The servo circuit, built around a PIC16F877 microprocessor, detects the pulse change and begins to monitor the angle of the servo shaft, as an analog signal. It asserts the servo-move signal, which remains high until the servos have stopped.

In this fashion, the instantaneous information about the camera position is decoupled from the vision system. The vision merely issues the desired position, and receives confirmation when that position is reached.

While the servo motors are moving to their new position, the IRQ5 is not being generated, since the motion detector would counteract the displacement motion, leading to unsteady movement of the camera.

16.4 Displacement vector

When it is not communicating with the IRQ5 handler, the duty-cycle generator waits for the displacement vector transfer, initiated by the MCF5307. The displacement vector is

being calculated by the Process 2, and when ready, it is transferred to the back-end FPGA in the same way as the motion vector.

In this transfer sequence, however, the servo-motion signal (the response to the new displacement vector) is passed back to the MCF5307, forcing the Process 2 into a busy wait until the servos have stopped moving. This might appear wasteful at first, but it is easy to see that Process 2 really must pause during the servo motion. The snapshot for feature recognition must not be taken until the camera has moved to the new location. Otherwise, the change in the image would not be registered, and the next cycle of feature recognition would end up working with stale data.

16.5 Saccadic blanking

On the face of it, this suppression of image sampling during camera motion resembles the phenomenon of saccadic blanking in human/animal vision.[12] It is well known that the sensitivity of the optic nerve is suppressed while a saccade (a rapid eye movement) is in progress, and it is plausible that the purpose of this suppression is to prevent visual confusion in biological systems as well.

Interestingly, there is some question whether the suppression of the optic nerve signal is triggered by the blurring of retinal image or by a signal from tension sensors in the eye muscles.[13] In a robotic system it is much easier to detect servo motion than image blur, and the choice of mechanism is obvious.

16.6 Description of the IRQ/PP circuit on the back-end FPGA

The circuit design is contained in the schematic DUTY_CYC_PP. Control of the process is carried out by the ASM component IRQ_PP1, described in the previous section (see also ASM chart, Picture 43, and timing diagrams, Pictures 44 and 45).

The data path leads from the parallel port to two 14-bit registers, which hold the current pulse-width values. These values are in turn available to the square-pulse generators.

The pulse generator component, CYCLE_GEN, contains a fixed-value counter, which measures the 20 ms period of the servo's duty cycle, and a loadable counter, which measures the current pulse width. A simple two-state ASM switches between high and low signal levels.

The servo-move signal passes through the component DIP_FILTER which eliminates the short (less than a CLK cycle) dips in the signal (noise).

ACK, RDY – handshake signals on the parallel port:

ACK - pin 14, output

RDY - pin 15, input

IRQTRG – signal from the front-end FPGA to start the IRQ5 communication at the start of odd field (see Picture 18).

IRQ – hardware request for Interrupt 5 on ColdFire.

SERVO_MOVE – the cleaned-up servo motion signal; also sent out to pin 0 on the parallel port.

PIN0_TSB – signal which reverses the sense of pin 0: output when high, input when low. Pins 1-13 are all input pins

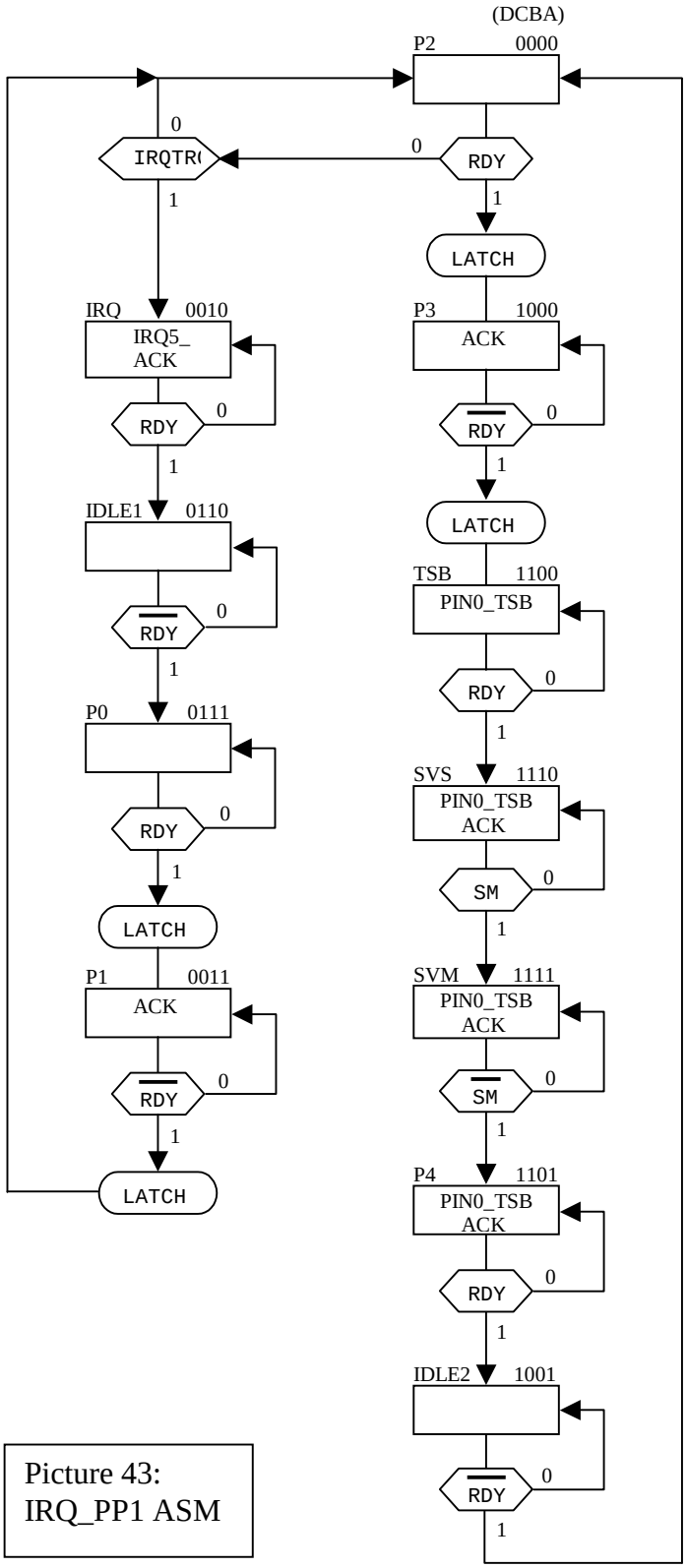
LATCH1, LATCH2 – register-enable signals to capture the pulse width values.

PWM1, PWM2 – generated servo duty waveforms.

Comm. with handler:

Acknowledge handler and terminate IRQ5 signal.

Latch pulse widths into data registers.

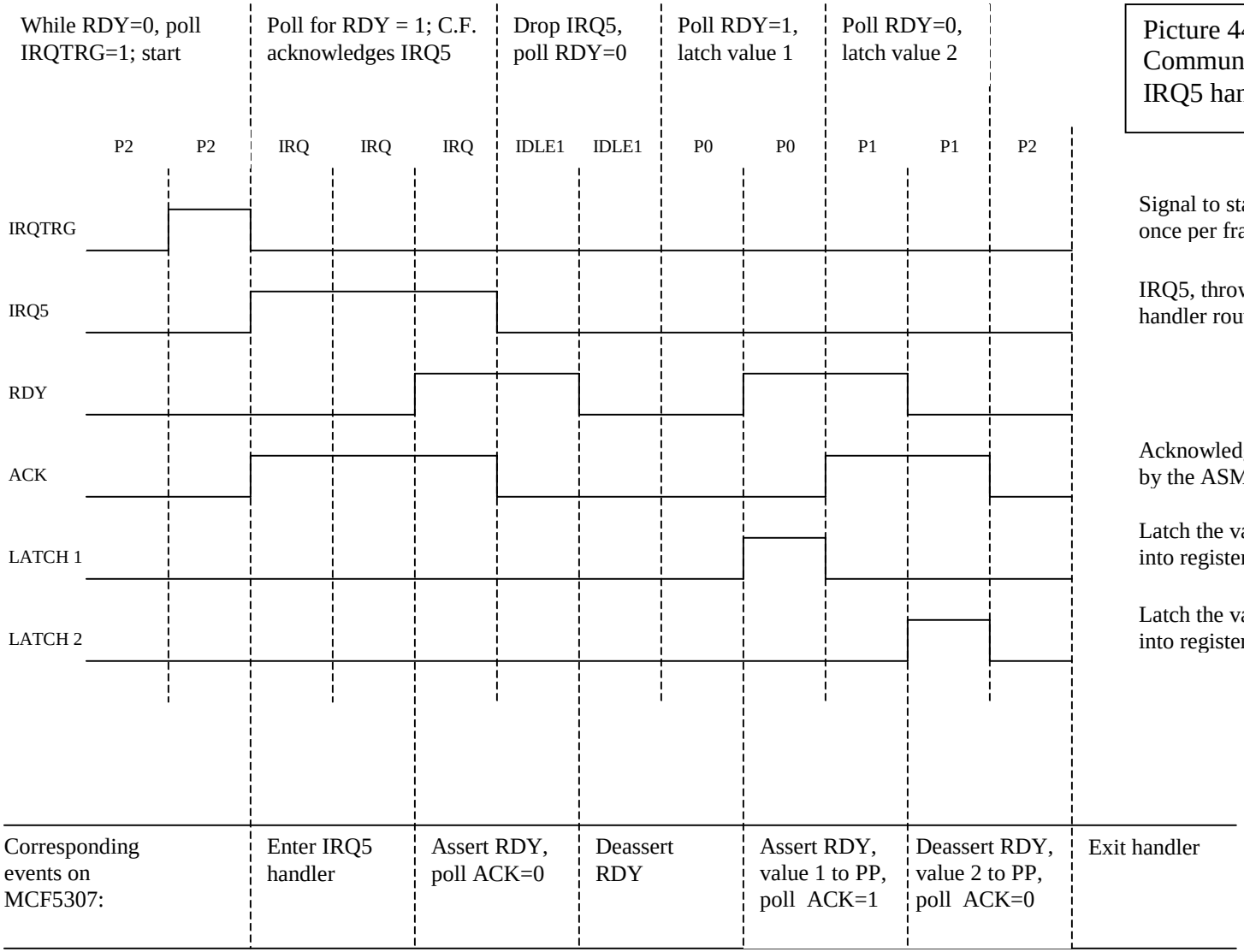


Comm with Proc.2:

Latch pulse widths into data registers.

Poll the servo motion signal until it goes high, then low again. Idle during that time.

Picture 43:
IRQ_PP1 ASM



Picture 44: Communication with the IRQ5 handler

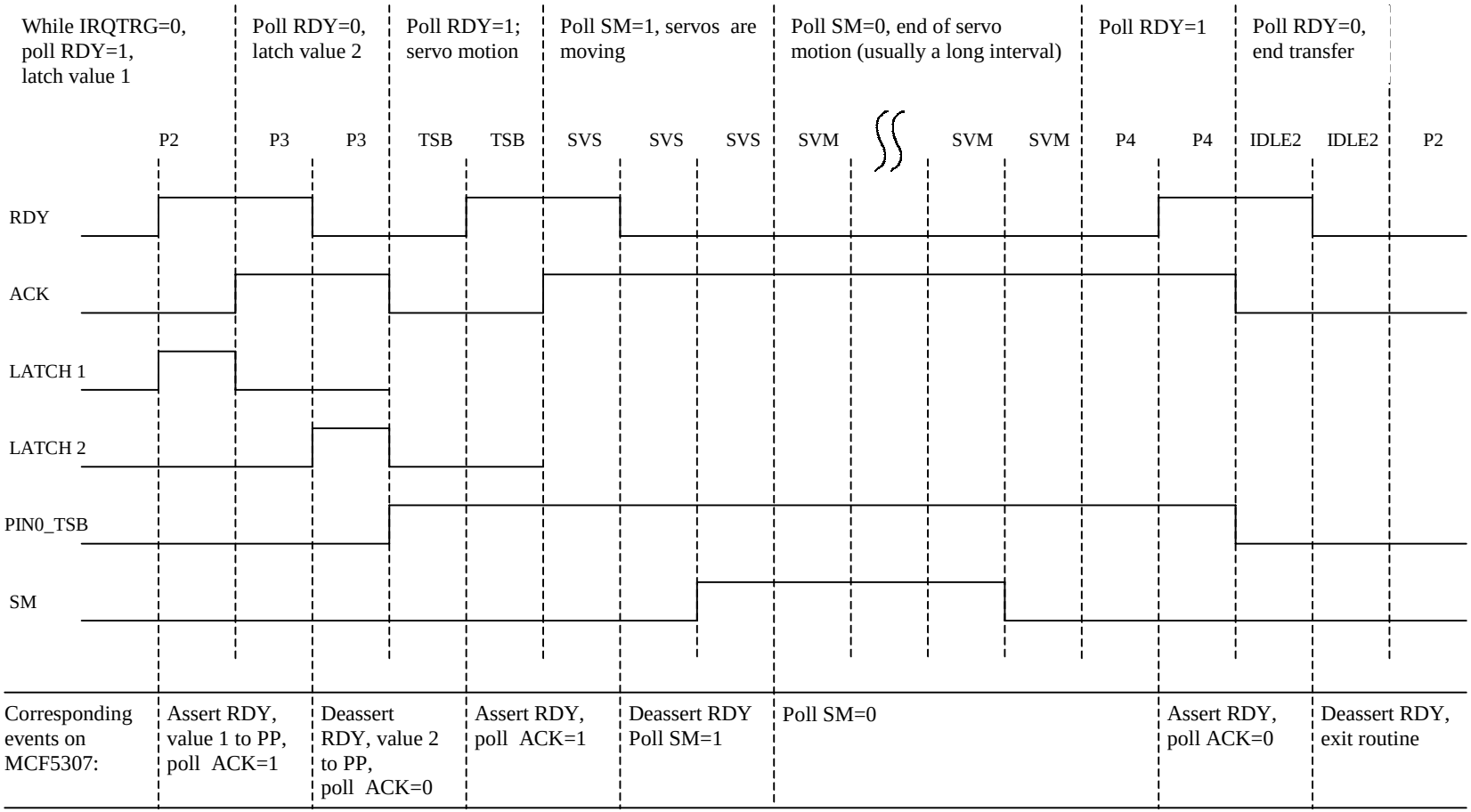
Signal to start the IRQ5 exchange, once per frame

IRQ5, throws ColdFire into the handler routine

Acknowledgment signal generated by the ASM

Latch the value on parallel port into register 1

Latch the value on parallel port into register 2



Notes:

1) PIN0_TSB reverses the sense of pin 0 on the PP. When low, pin 0 is the input bit zero. When high, pin 0 outputs the SM (servo move) signal to C.F.

2) SM is the filtered and synchronized input from the camera servo board. When high, at least one servo motor is in motion.

Picture 45: Communication with Process 2

17. AUXILIARY FEATURES

The system has several auxiliary features, which do not pertain to its main function, but which are necessary for the deployment on a flying airplane.

17.1 Serial communication with the MCF5307

The ColdFire processor has two built-in UARTs, and the SBC5307 board has two serial ports. One port is configured and active, and the other can be easily made so as well. There is a small I/O utility library, which allows transmission of byte strings, and transmission/reception of individual bytes.

17.2 Diagnostic data logging

Lower bank of the SBC5307's DRAM is organized as a four-megabyte circular buffer, to be used for logging data during the system's operation. This buffer is initialized upon board reset, so the system should not be casually reset (or turned off!) before the diagnostic data are retrieved. A utility library allows for recording of bytes and strings of bytes, and for dumping the buffer's contents to the serial port. Data should be captured to a file using a terminal software. Note: press the “D” key to start the data dump to the terminal.

17.3 Soft restart of the vision program

The non-maskable Interrupt 7 is used to implement a soft restart of the vision program on the MCF5307. When IRQ7 is received (as a hardware signal), the corresponding handler cleans up the cache and the parallel port, releases all heap allocations, resets the stack and

returns to the main entry point. In this way, the vision algorithm can be pulled out of some confused state without resetting the entire ColdFire board.

17.4 Radio controls

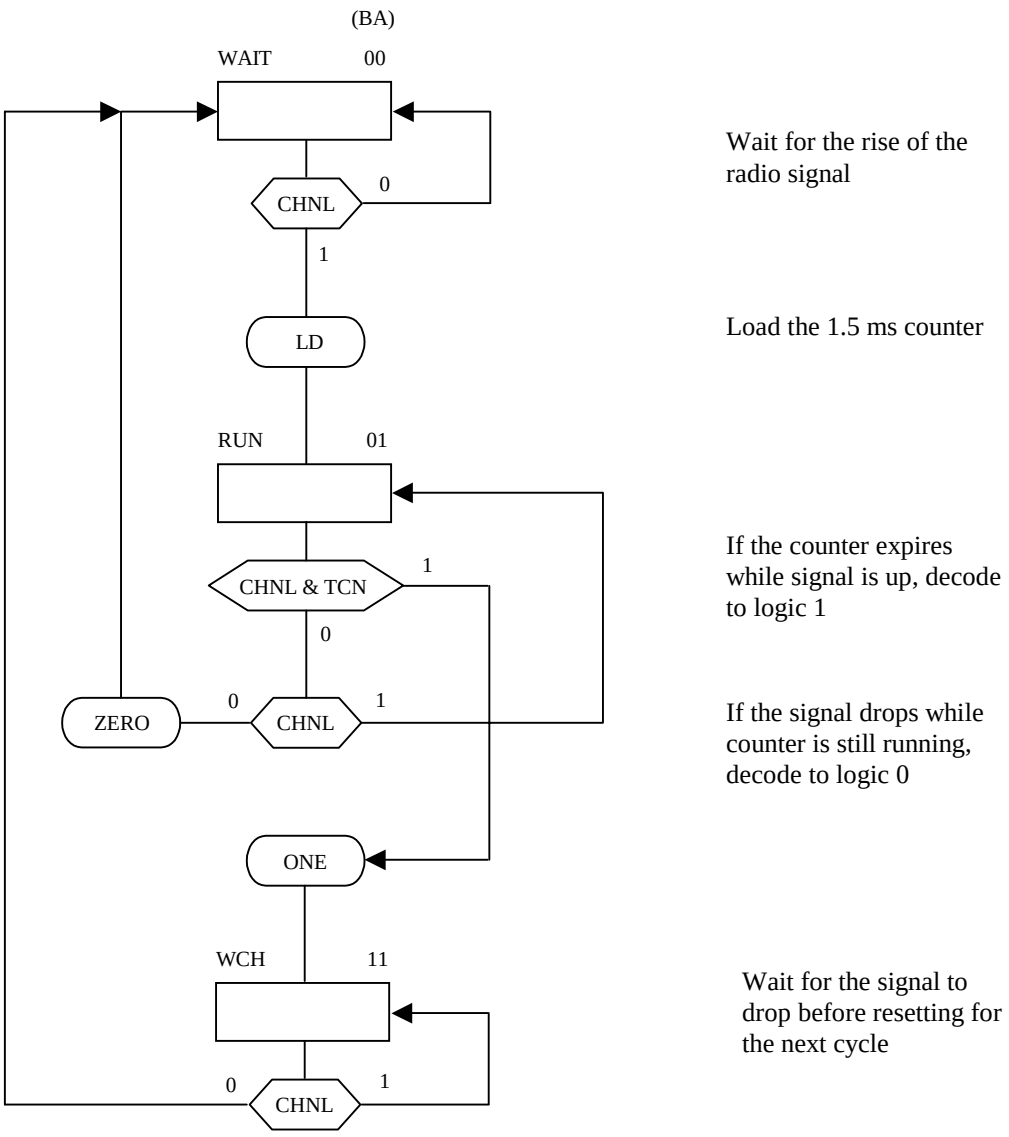
The system contains a decoder (implemented on an XC9572 CPLD) which accepts four PWM signals from a radio receiver, and converts them into four logic signals. The decoder uses the flight control's power supply and receives no input from the rest of the system. Its purpose is to implement radio control over the following functions:

- power on/off: circuitry unrelated to manual controls of the plane can be shut off from the ground in an emergency. An electronic switch is attached to the battery pack for that purpose.

- reset: vision boards (and possibly other components, in the future) can be reset to a clean state in case of irreparable malfunction.

- soft restart: the vision system can be brought back to its clean state without reset and the consequent loss of diagnostic data. The FPGA part of the system is reset without harm. The vision can also be held in restart continuously, in which case the camera's gaze is fixed, the tracking is stopped and no data are logged. This is useful to keep the camera from turning around aimlessly during takeoff and landing.

- zoom: the zoom of the vision system can be turned on or off by the operator.



Picture 46: Radio decoder ASM (module CHANNEL)

17.4.1 Radio decoder's signals in the schematic CHANNEL

CHNL – pulse-width modulated signal from the radio receiver.

LD – synchronous load signal for the 1.5 ms counter.

TCN – the counter's expiration signal (term count).

ZERO/ONE – clear/set the logic output.

18. FEATURE RECOGNITION ON THE MCF5307 PROCESSOR

Functioning of the programs running on the ColdFire processor is best understood in terms of the time periods involved. One component, centered around the motion vector, runs in response to IRQ5, which is issued by the FPGA once per video frame. It has already been described earlier, in Section 16 on the IRQ5/parallel port subsystem.

The other component, feature recognition, runs in the background relative to the interrupts. It has no time requirements imposed on it by the input signal, only the general consideration that faster processing leads to better tracking. Here we describe that procedure.

The feature recognition consists of the following computations:

- the thresholded black-and-white image is scanned for edges. A pixel is defined as an edge point if it is black in color, and has between 2 and 7 black neighbors. The output of edge detection is another b/w image: edges are traced in black, and the redundant interior of the features has been removed.

- segmentation: edge points are logically grouped into connected threads or loops, and each connected component is assumed to represent a distinct feature (object) in the scene. The output is a collection of arrays, each containing the edge coordinates of one feature. We use a two-pass algorithm described by Lumia et al.[14] (Algorithm 3 in the reference). For each line, adjacent points in that and the previous line are

labeled as belonging to the same component, and the labeling equivalences are resolved by means of a depth-first graph search. Here is a simple example:

Original image	Labeled image
0 0 1 0 0 0 1	0 0 a 0 0 0 b
0 0 1 1 0 0 1	0 0 a a 0 0 b
0 0 1 1 1 0 1	0 0 a a a 0 b
0 0 1 1 1 1 1	0 0 a a a a X

At the pixel X, the equivalence of labels a and b will be recognized, and changes in labeling carried out in the second (bottom up) pass. The procedure works well for simple scenes, but the connectedness problem is combinatorial in nature, and the equivalence search will eventually get bogged down in complex images.

- insignificantly small features are removed, and an invariant signature is calculated, which is used for actual recognition of the object in the scene. We use the second moments about the principal axes of inertia, which are invariant under rotation and translation and relatively easy to obtain. The calculation involves one square root per feature, but requires no adjacency information about the feature points. As a preliminary, this step also calculates the feature's center of mass.

- target recognition: signatures of the objects in the scene are compared to those of a selected target feature. If the target is found, the displacement vector is the difference between its center of mass and the centerpoint of the image. If the target is lost, an error is returned.

18.1 Main data structures in the MCF5307 code

Digitized Image:

IMAGE_FRAME	starting address of the read buffer P2R on DPRAM
FRAME_WIDTH	image dimensions in pixels
FRAME_HEIGHT	

Motion/Displacement Vectors:

XS_VECT_ADDR	DPRAM address of the vector components
X_MVECT, Y_MVECT	local copy of the motion vector
NORM_MVECT	normalization constant of the motion vector
X_DVECT, Y_DVECT	the displacement vector

Semaphores:

These variables carry messages between (and within) the two main processes on the MCF5307. They are integers, set to one and cleared to zero.

DISP_VECT_AVAIL	feature recognition announces that a new displacement vector is available
TARGET_AVAIL	main process announces that a target feature has been selected for tracking

Post-Segmentation Description of Image Features:

Each individual connected edge is stored in a structure of the type Fpoint. The structure contains the point count, arrays of coordinate values, center of mass and the second

moments. These structures are allocated dynamically as the features emerge from image analysis.

compResult - a structure of the type ComponentList*, holds the overall result of the image segmentation. It contains the feature count, and an array of pointers to structures of type Fpoint, which contain the details of each feature.

targetFeature - the feature that is being tracked is stored in this Fpoint structure.

Fpoint and ComponentList data types are specified in the header file FeaturePoints.h

Parameters of the Servo Duty Cycles:

See the data section of the program module Servo.s for description and current values.

19. INITIALIZATION OF THE SBC5307 BOARD

Pre-initialization conditions:

The entire code (init and functional, ca. 45K in size) resides in flash ROM, and the flash ROM is associated with the global chip select. Vector table and initialization code are linked to, and reside at, zero address; vector base register (VBR) points to zero. Functional code is linked to the upper bank of DRAM, and resides in the flash, just above the init code. See the linker script sbc5307vis.ld for the details of the linking procedure.

Initialization sequence:

- on reset exception, initial PC and SP are loaded from ROM, as longwords at addresses zero and four. PC points to start-up code in ROM.
- ColdFire processor is configured: cache is disabled and turned off; SIM and upper bank of DRAM are configured; chip selects are configured and the system is pulled out of the global chip select
- contents of the flash ROM are copied to upper DRAM, VBR is set to the base of upper DRAM
- program control jumps to DRAM starting point
- chip select zero is reconfigured to the top of address space (to get the flash memory out of the way)
- lower bank of DRAM is configured
- control jumps to main and starts running functional code

20. CHARACTERISTICS OF THE CAMERA/SERVO SYSTEM (GIMBAL)

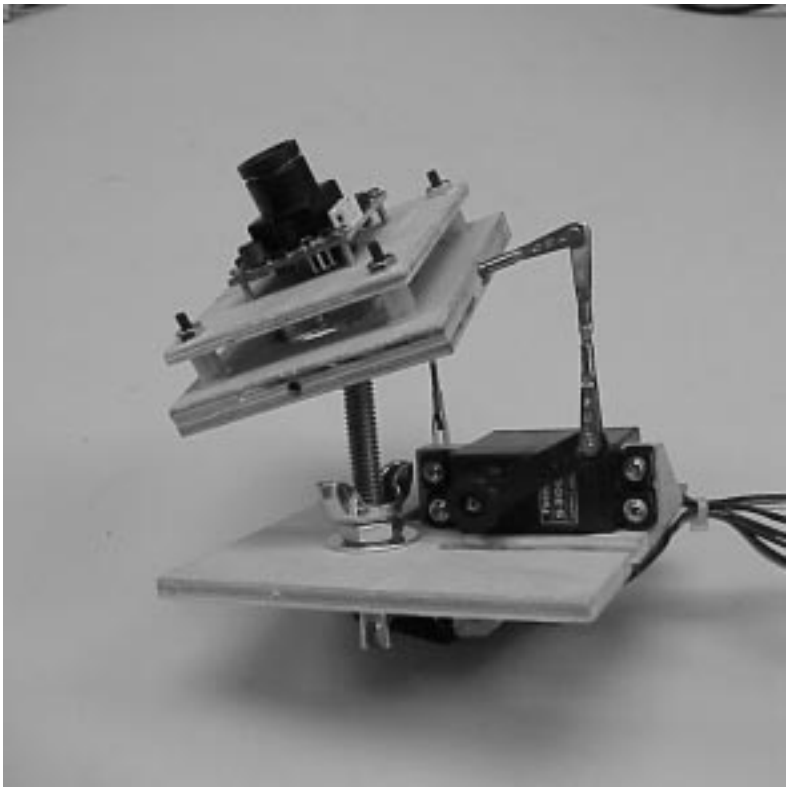
A technical description of the camera is not available, but by measuring the screen image of an object of known size, we have determined the following:

- the angle corresponding to a one-pixel displacement (the camera ratio) is 0.62 degrees/pixel, at our given image resolution,
- the camera's field is ca. 60 degrees wide and ca. 47 degrees high.

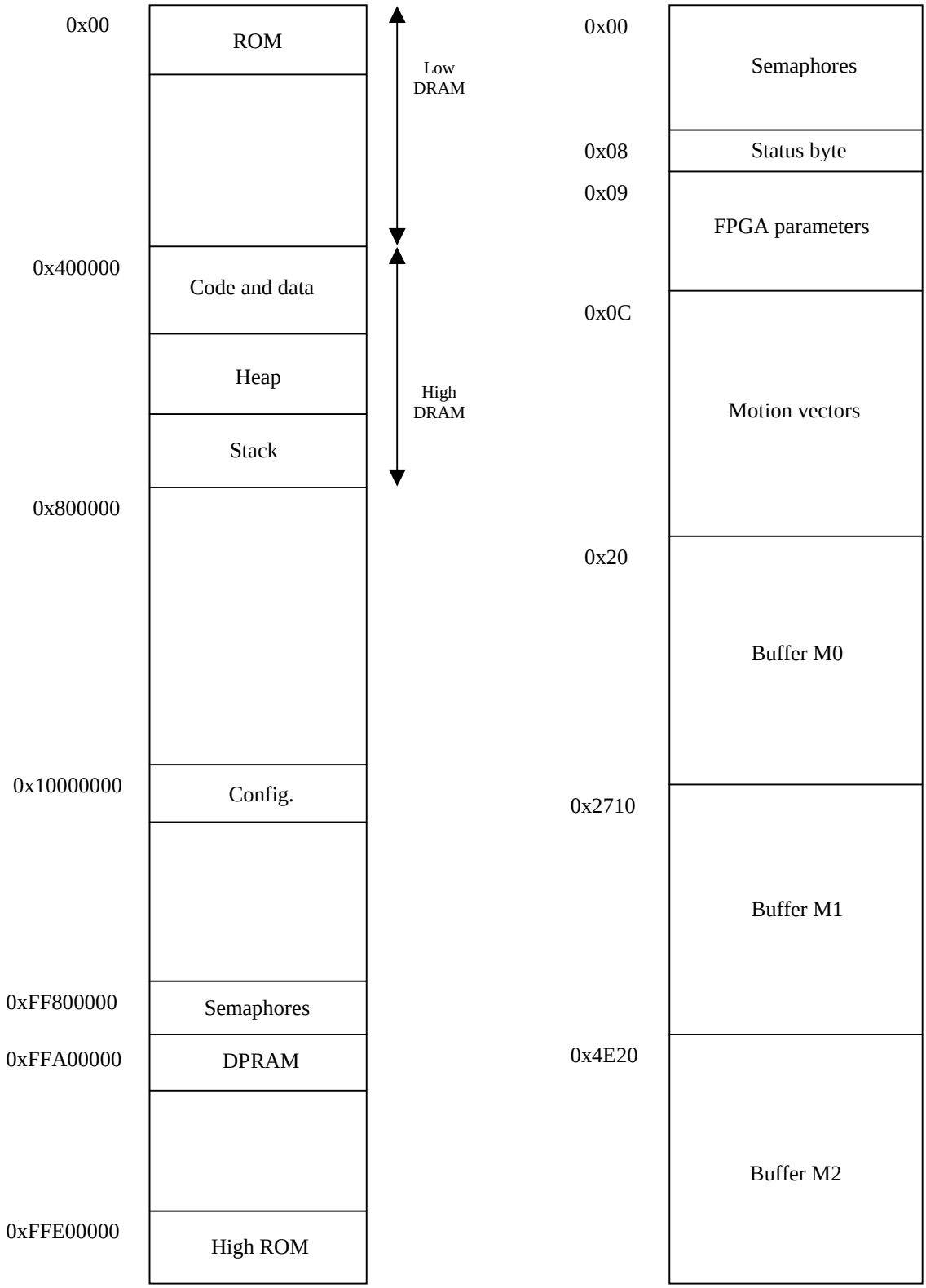
The servo motors which move the camera have an angular motion range of ca. 90 degrees. Their duty cycle is 20 milliseconds, with the recommended high time ranging from 1 to 2 ms. These duty cycles are generated by the back-end FPGA.

The camera's platform is driven by a lever mechanism (see Picture 47), and its angular motion is not entirely linear relative to the motion of the servos. However, within the above high-time range the relationship is reasonably linear, and the platform covers an angle of ca. 46 degrees in each coordinate direction.

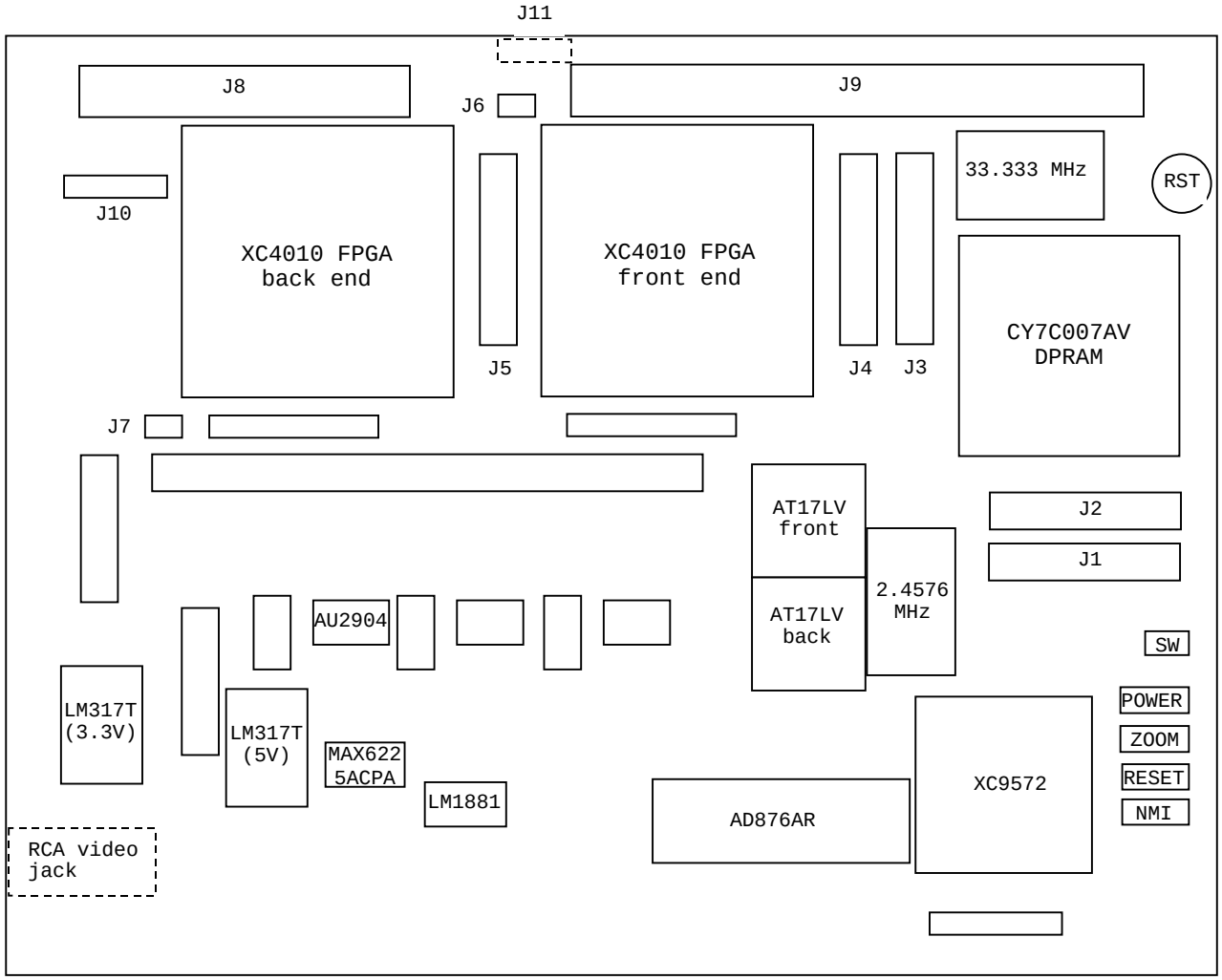
In the current circuit design, the 1 to 2 ms range corresponds to the range of 2048 clock ticks of high time, or 45.3 ticks per degree of platform motion. Combined with the above camera ratio, this yields 28.3 ticks per one pixel of displacement, the constant of proportionality between motion/displacement vectors and the motor movement.



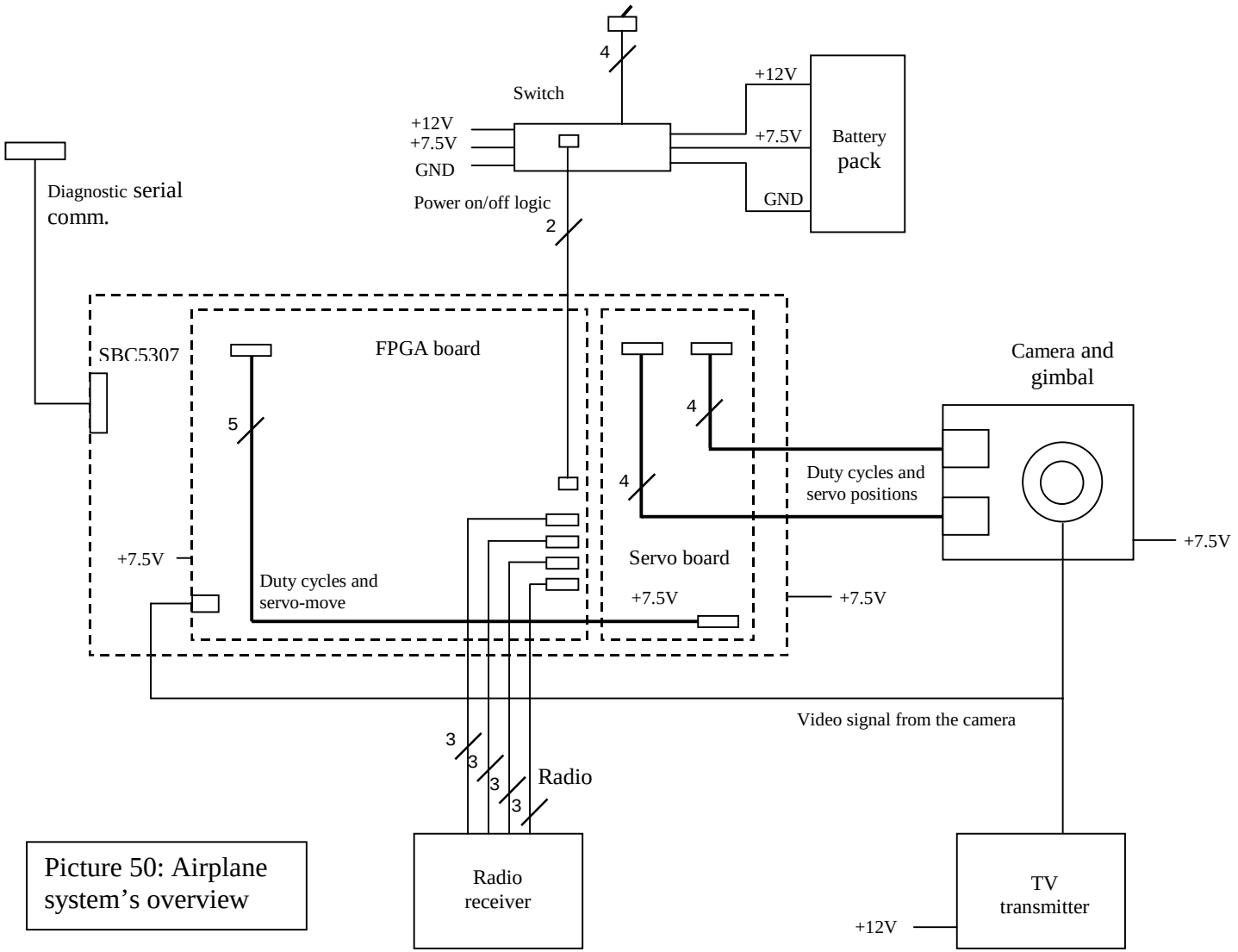
Picture 47: View of the camera gimbal



Picture 48: Memory maps



Picture 49: Layout of the FPGA board



Picture 50: Airplane system's overview

REFERENCES

[1] R.W. Rodieck, “The First Steps in Seeing,” Sinauer, 1998.

[2] David Van den Bout, “The Practical Xilinx Designer Lab Book, Version 1.5,” Prentice-Hall, 1999.

[3] “ColdFire Microprocessor Family Programmer’s Reference Manual,” Motorola, 1997 (MCF5200PRM/AD)

[4] Leon W. Couch, “Digital and Analog Communication Systems,” third edition, MacMillan, 1990, Chapter 5-9.

[5] Data sheet for the video sync separator LM1881, National Semiconductor, February 1995.

[6] Data sheet for the A/D converter AD876, Analog Devices, December 1997.

[7] Data sheet for the CY7CnnnAV Dual-Port Static RAM, Cypress Semiconductor, December 7, 2000.

[8] “Understanding Asynchronous Dual-Port RAMs,” technical note, Cypress Semiconductor, rev. November 7, 1997.

[9] MCF5307 User’s Manual, Motorola, 1998 (MCF5307UM/AD)

[10] Nicolas Franceschini, “Early Processing of Colour and Motion in a Mosaic Visual System,” Neuroscience Research, Supplement 2, pp. S17-S49, Elsevier, 1985.

[11] Jean-Marc Pichon, Christian Blanes, Nicolas Franceschini, “Visual Guidance of a Mobile Robot,” SPIE, v.1195, 1989.

[12] Brian A. Wandell, “Foundations of Vision,” Sinauer, 1995.

[13] S. Yu and T.S. Lee, "What Do V1 Neurons Tell Us about Saccadic Suppression?," Journal of Neural Computing, 2000.

[14] Ronald Lumia, Linda Shapiro, Oscar Zuniga, “A New Connected Components Algorithm for Virtual Memory Computers,” Computer Vision, Graphics and Image Processing, **22**, 287-300 (1983).

CURRICULUM VITAE

Danko Antolovic

Born in 1955, in Zagreb, Croatia

Education:

- B.S. Chemistry, University of Zagreb, Croatia (1978)
- M.S. Chemistry, Johns Hopkins University, Baltimore, MD (1981)
- Ph.D. Chemistry, Johns Hopkins University, Baltimore, MD (1983)
- M.S. Computer Science, Indiana University, Bloomington, IN (2001)

Work experience:

- 25 years in systems analysis and computer software development

  Platforms: desktop, servers, mainframes and supercomputers

  Applications: Robotics, Computer Vision, Industrial Programming, Solar and Gas Combustion Modeling, Quantum Chemical Modeling, Molecular Modeling
- 10 years in scientific research
- 5 years in teaching at the undergraduate level
- 5 years in running a software development business