RJ 10289 (A0301-024) January 20, 2003
Computer Science

IBM Research Report

An Embedded System for an Eye Detection Sensor

Lior Zimet[1], Sean Kao[2], Arnon Amir[3], Alberto Sangiovanni-Vincentelli[2]

[1]University of California at Santa Cruz
1156 High Street
Santa Cruz, CA 95064

[2]University of California at Berkeley
Berkeley, CA 94720

[3]IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

An Embedded System for an Eye Detection Sensor

Lior Zimet[1], Sean Kao[2], Arnon Amir[3] and Alberto Sangiovanni-Vincentelli[2]
[1] University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064 liorz@zoran.com
[2] Dept of EECS, University of California Berkeley, CA 94720. kaos,alberto@eecs.berkeley.edu
[3] IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. arnon@almaden.ibm.com

Real-time eye detection is important for many HCI applications, including eye-gaze tracking, auto-stereoscopic displays, video conferencing, face detection and recognition. Current commercial and research systems use a software implementation and require a dedicated computer for the image-processing task, a large, expensive and complicated-to-use solution. In this paper, we present a hardware-based embedded system for eye detection implemented using an FPGA, with no CPU. A prototype system uses a 1.3MPixel digital imaging sensor at 27Mpixel/sec rate and outputs a compact list of sub-pixel accurate (x,y) eye coordinates via USB communication. By eliminating the CPU in the implementation architecture, the processing rate is only limited by the effective sensor pixel rate. Our design is suitable for single-chip eye detection and eye-gaze tracking sensors.

1. Introduction

Eye detection, the task of finding and locating eyes in images, is used for a many applications. One example is eye-gaze tracking systems, used in HCI for a variety of applications [Hut89,Hyr97,ETRA2], such as pointing and selection [Jac91], activating commands (e.g., [Sal00]), and combinations with other pointing devices [Zha99]. The present implementations of eye gaze tracking systems are software driven and require a dedicated high-end PC for image processing. Yet even high-end PCs cannot provide the desired high frame rate (about 200fps) and high-resolution images required to increase accuracy. Even if the performance requirement were fully met, these solutions are bulky and prohibitively expensive thus limiting their reach in a very promising market.

Any CPU-base implementation of real-time image processing algorithm has two major bottlenecks – data transfer bandwidth and sequential data processing rate. First, video frames are captured, from either an analog or a digital camera, and transferred to the CPU main memory. Moving complete frames around imposes a data flow bandwidth bottleneck. Then the CPU can sequentially process the pixels. A CPU adds overhead to the actual computation. Its time splits among moving data between memory and registers, applying arithmetic and logic operators on the data, and keeping the algorithm flow, handling branches, and fetching code. For example, if each pixel is accessed only ten times along the entire computation (e.g., applying a single 3x3 filter), it would require the CPU to run at least ten times faster than the input pixel rate, and even faster due to CPU overhead. Hence, the CPU has to run much faster than the camera pixel rate. Last, a relatively long latency is imposed by the need to wait for an entire frame to be captured before it is processed. Device latency is extremely noticeable in interactive user interfaces.

The advantage of a software-based implementation lies in its flexibility and short implementation time. A hardware implementation would certainly improve cost and size, but at the price of increased design time and rigidity. An efficient hardware implementation should offer the same functionality and exploit its main characteristic: concurrency. A simple mapping of the software implementation into hardware components falls short of the potential benefits offered by a hardware solution.

Our approach to embedded system design is based on a methodology [San02,Bal02] that captures the design at a high-level of abstraction where maximum concurrency may be exposed. This representation is then mapped into a particular architecture and evaluated in terms of cost, size, power consumption and performance. The methodology offers a continuous trade-off among possible implementations including, at the extremes, a full software and a full hardware realization.

This paper presents a prototype of an embedded eye-detection system implemented completely in hardware. The novelty of this paper is the redesign of a sequential pupil detection algorithm into a synchronous parallel algorithm capable of fully exploiting the concurrency of hardware implementations. The algorithm includes computation of connected components, shape moments and simple shape classification – all of which are computed within a single pass over the image, driven by the real time pixel-clock of the camera sensor. It operates at the camera's pixel clock. The output is a list of pupil location and size, passed via low-bandwidth USB connection to an application. In addition to the elimination of a dedicated PC and thus major saving in cost, the system has virtually no limit in processing pixel rate, limited only by the camera output rate. Minimal latency is achieved as pixels are processed while being received from the camera.

A prototype was built using a Field Programmable Gate Array (FPGA). It processes 640x480 progressive scan frames at 60fps - a pixel rate that is sixteen times faster than the one reported in [Mor00]. By integrating this logic into camera sensor VLSI, this approach can solve the speed, size, complexity and cost of current eye gaze tracking systems.

2. Prior Work on Eye Detection

Much of the eye detection literature is associated with face detection and face recognition, recently surveyed in [Hje01] and [Zha00]. Direct eye detection methods search for eyes without prior information about face location, and can further be classified into passive and active methods. Passive eye detectors work on images taken in natural scenes, without any special illumination and therefore can be applied to movies, broadcast news, etc. One such an example can be found in [Kot96] where a gradient field and temporal analysis are used to detect eyes in gray-level video.

Active eye-detection methods use special illumination and thus are applicable to real time situations in controlled environments, such as eye gaze tracking, iris recognition, and video conferencing. They take advantage of the retro-reflection property of the eye, a property that is rarely seen in any other natural objects. When light falls on the eye, part of it is reflected back, through the pupil, in a very narrow beam pointing directly towards the light source. When a light source is located very close to a camera focal axis (on-axis light), the captured image shows a very bright pupil [Hut89,You75]. This is often seen as the red-eye effect in flash photography. When a light source is located away from the camera focal axis (off-axis light), the image shows a dark pupil. However, neither of these lights allow for good discrimination of pupils from other objects, as there are also other bright and dark objects in the scene that would generate pupil-like regions in the image.

2.1. Frame Subtraction Using Two Illuminations

The active eye detection method used here is based on the subtraction scheme with two synchronized illuminators [Tom89,Ebi93,Mor00]. At the core of this approach, two frames are captured, one with on-axis illumination and the other with off-axis illumination. Then the illumination intensity of the off-axis frame is subtracted from the intensity of the on-axis frame. Most of the scene, except pupils, reflects the same amount of light under both illuminations and subtracts out. Pupils, however, are very bright in one frame and dark in the other, and thus are detected as elliptic regions after applying a threshold on the difference frame. False positives might show due to specularities from curved shiny surfaces such as glasses, and at the boundaries of moving objects due to the delay between the two frames.

Tomono [Tom89] used a modified three CCD camera with two narrow band pass filters on the sensors and a polarized filter with two illuminators in the corresponding wavelengths, the on-axis being polarized. This configuration allowed for simultaneous frames acpture. It overcame the motion problem and most of the specularities problems. However, it required a more expensive, three CCD cameras.

The frame-subtraction eye-detection scheme has become popular in different applications because of its robustness and simplicity. It was used for tracking eyes, eyebrow and other facial features [Har00,Kap02a], for facial expression recognition [Har01], tracking head nod and shake, recognition of head gesture [Dav01], achieving eye contact in video conferencing [Ver02], stereoscopic displays [Per00] and more.

The increasing popularity and multitude of applications suggest that an eye detection sensor would be of great value. An embedded eye-detection system would simplify the implementation of all of these applications. It would eliminate the need for high-bandwidth image data transfer between imaging sensor and the PC, reduce the high CPU requirements for software-based image processing, and would dramatically reduce their cost.

2.2. The Basic Sequential Algorithm

The frame subtraction algorithm for eye detection is summarized in Figure 1. A detailed description can be found in [Mor00]. The on-axis and off-axis light sources alternate, synchronized with the camera frames. After capturing a pair of frames, subtracting the frame associated with the off-axis from the one associated with the on-axis light and thresholding the result, a binary image is obtained. This image contains regions of significant difference. The regions are detected using connected components [Bal82]. Those regions include pupils and false positives, due to object motion and image noise. The non-pupil regions are filtered in step 6, and the remaining pupil regions are reported to the output.

```
For each video frame:
1. Capture an on-axis frame
2. Subtract previously captured off-axis frame
3. Apply threshold to obtain regions image
4. Find regions (connected components)
5. Compute shape properties of each region
6. Filter out the non-pupil regions
7. Compute and report pupil centroids and size
```

Figure 1: Processing steps of the basic sequential algorithm.

The first five steps are applied on images. They consume most of the computational time and large memory arrays for image buffers. In step 5 the detected regions are represented by their properties, including area, bounding box and first-order moments. The filtering in step 6 rejects regions that do not look like pupils. Pupils are expected to show as elliptic shapes in a certain size range. Typical motion artifacts show as elongated shapes of large size and are in general easy to distinguish from pupils[1]. At the last step, the centroids and size of the detected pupils are reported. This output can be used by different applications.

3. Synchronous Pupil Detection Algorithm

To use a synchronous hardware implementation, it was necessary to modify the basic sequential algorithm of Section 2.2. While a software implementation is straightforward, a hardware implementation using only registers and logic gates operating at pixel-rate clock is far more complicated. The slow clock rate requires any pixel operation to be performed in a single clock and any line operation to be finished within the number of clock cycles equivalent to the number of pixels per image line.

Figure 2 shows a block diagram of the parallel algorithm. The input is a stream of pixels. There is only a single frame buffer, used to store the previous frame for the frame-to-frame subtraction. The output of the frame subtraction is stored in a line buffer as it is being computed in a raster scan. The output is a set of XY locations within the frame region of the centroid of the detected pupils. This set of XY locations is updated once per frame.

Connected components are computed using a single-pass algorithm, similar to the region-coloring algorithm [Bal82]. Here, however, the algorithm operates in three parallel stages, running in a pipeline on three lines: when line L_i is present in the line buffer, it is scanned and line components, or just *components*, are detected. At the same time, components from line L_{i-1} are being connected (merged) to components from line L_{i-2}, and those line components are being updated in the frame regions table.

The shape properties, including bounding box and moments (step 5 Figure 1) are computed simultaneously with the computation of the connected components (step 4). This saves time and eliminates the need for an extra frame buffer, as being used between step 4 and step 5 in Figure 1. Consider the computation of a property $f(s)$ of a region s. The region s is being discovered on a line-by-line basis and is composed of multiple line components. For each line object that belongs to s, say s_1, the property $f(s_1)$ is computed and stored with the line object. When s_1 is merged with another object, s_2, their properties are combined to reflect the properties of the combined region $s = s_1 \cup s_2$. This type of computation can be recursively applied to any shape property that can be computed from its value over parts of the shape. The property $f(s)$ can be computed using the form:

$$f(s) = g(f(s_1), f(s_2))$$

where $g(,)$ is an appropriate aggregation function for the property f. For example, aggregation of moments is simply the sum, $m_s^{ij} = g(m_{s1}^{ij}, m_{s2}^{ij}) = m_{s1}^{ij} + m_{s2}^{ij}$, whereas for the aggregation of normalized moments, the regions areas A_{s1}, A_{s2} are needed:

$$\overline{m}_s^{ij} = g(\overline{m}_{s1}^{ij}, \overline{m}_{s2}^{ij}) = (A_{s1}\overline{m}_{s1}^{ij} + A_{s2}\overline{m}_{s2}^{ij})/(A_{s1} + A_{s2})$$

The four properties which define the bounding box, namely $X_s^{\min}, X_s^{\max}, Y_s^{\min}, Y_s^{\max}$, are aggregated using, e.g., $X_s^{\min} = \min(X_{s1}^{\min}, X_{s2}^{\min})$.

Properties that are computed here in this manner include area, first order moments and bounding box. Other properties that may be useful are higher order moments, average color, color histograms, etc.

In the pipelined architecture, each stage of the pipeline performs the computation on the intermediate results from the previous stage, using the same pixel clock. The memory arrays in the processing stages keep the information of only one or two lines at a time. Additional memory arrays are used to keep components properties and components location, and to form the pipeline.

To get the effect of bright and dark pupils in consecutive frames, the logic circuit drives control signals to the on-axis and off-axis infrared light sources. These signals are being generated from the frame clock coming from the image sensor.

3.1 Pixels Subtraction And Thresholding

The digital video stream is coming from the sensor at a pixel rate of 27Mhz. The data is latched in the logic circuit and at the same time pushed into a FIFO frame buffer. The frame buffer acts as one full frame delay. Figure 3 depicts the subtraction and threshold module operation.

[1] More elaborate eye classification methods were applied in [Coz99]

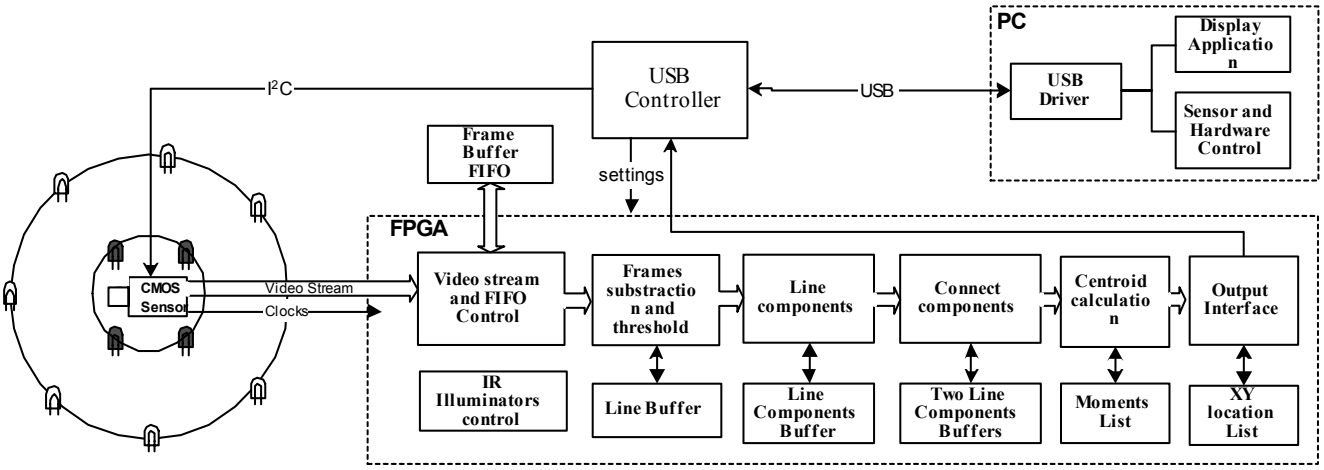

Figure 2: System block diagram, showing the main hardware components and pipeline processing.

Since each consecutive pair of frame is processed, the order of subtraction changes at the beginning of every frame. The on-axis and off-axis illumination signals are used to coordinate the subtraction order. To eliminate the handling of negative numbers, an offset is added to the on-axis frame. The subtraction operation is done pixel by pixel and the result is then compared to a threshold that may be modified for different light conditions and noise environments. The binary result of the threshold operation is kept in a line buffer array that is transferred to the Line Components Properties module on the next line clock.

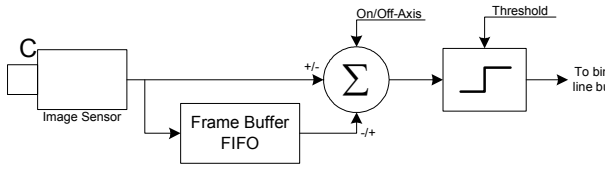

Figure 3 - Subtraction and Thresholding

3.2 Detecting Line Components and their Properties

Working at the sensor pixel clock, this module is triggered every line clock to find the components properties in the binary line buffer resulting from the subtraction and threshold module. Every "black" pixel in the line is part of a component, and consecutive black pixels are aggregated to one component.

Several properties are computed for each of these components: the starting pixel (x_{start}), the ending pixel (x_{end}), and first-order moments. The output data structure from this module is built of several memory arrays that contain the components properties. For a 640x480 frame the arrays with the corresponding bit width are: x_{start} (10 bits), x_{end} (10 bits), M_{00} (6 bits), M_{10} (15 bits), M_{01} (15 bits). Higher order moments would require additional arrays. Preliminary filtering of unreasonable sized objects can be done in this stage using the information in M_{00}, since it contains the number of pixels in the component.

3.3 Computing Connected Components

The line components and their properties are connected to components of previous lines to form regions in the frame. The connect components module is triggered every line clock, and works on two consecutive lines. It receives the x_{start} and x_{end} arrays and provides at the output a component identification number (ID) for every component and a list of merged components. Since the module works on pairs of consecutive lines, every line is processed twice.

There is never any representation of the boundary of a region throughout the entire process. As regions are scanned line by line through the pipeline, their properties are computed and held. But their shape is never stored, even not as a temporary binary image, as is a common practice with sequential implementations.

The processing uses the x_{start} and x_{end} information to check for neighboring pixels between two line components in the two lines. The following condition is used to detect all (four-) neighbors line components:

$$\left(x_{C\_start} \le x_{P\_end}\right) \& \left(x_{C\_end} \ge x_{P\_start}\right)$$

where the subscripts P, C refer to the previous and current lines, respectively. Examples for such regions are show in Figure 4. Using this condition, the algorithm checks for overlap between every component in the current line against all the components in the previous line. In software this part can be easily implemented as a nested loop. In hardware, the implementation involves two counters with a synchronization mechanism to implement the loops. If a connected component is found, the next internal loop can start from the last connected component in the previous line saving computation time.

Since the module is reset every line clock and each iteration in the loop takes one clock, the number of components per line is limited to the square root of the number of pixels in a line. For a 640x480 frame size it can accommodate up to 25 objects per line. A more efficient, linear time merging algorithm exist, but its implementation in hardware was somewhat more difficult.

A component from the current line that is connected to a component from the previous line is assigned with the same component ID as the one from the previous line. Components in the current line, which are not connected to any component in the previous line, are assigned with new IDs. In some cases, a component in the current line connects two or more components from the previous line. Figure 5 shows an example of three consecutive lines with different cases of connected components, and a merge. Obviously, the merged components should all get the same ID. Those components, which were assigned an ID would no longer need that ID after the merge. This ID can then be re-used. This is important in order to minimize the required size of arrays for component properties. In order to keep the ID list manageable and within the size constraint in the hardware, it was implemented as a linked-list. The list pointer is updated on every issue of a new ID or whenever a merge occurred. The ID list is kept non-fragmented. To support merging components, the algorithm keeps a merge list that is used in the next module in the pipeline. The merge list has an entry for each component with the component ID to be merged with, or zero if no merge is required for this component.

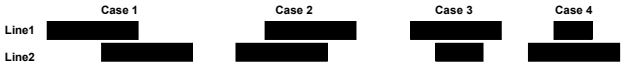

Figure 4: Four cases of connected components

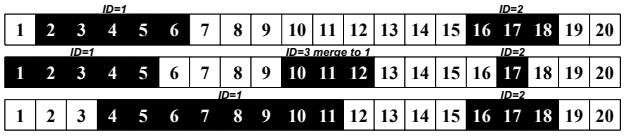

Figure 5: Three lines connect components example

3.4 Updating the Regions List

This module receives the list of IDs, merged components list, and the moments of components that were processed in the connect component module. It keeps a list of regions and their corresponding moments for the entire frame, along with a valid bit for each entry. Since regions are forming as the system advances through lines of the frame, the moments from each line are being updated to the correct entry according to the ID of the components. Moments of components with the same ID as the region are simply added. When two regions merge, their moments are also added, and the entry of the merged region is invalidated. When a new region is assigned to an invalidated entry, it updates the valid bit accordingly. This module is triggered every line clock. The list of components of the entire frame and their moments is ready at the time of receiving the last pixel of the frame.

3.5 Reporting the Detected Regions

Once the list of moments of the entire frame is ready, the system checks the shape properties of regions in the rame and decides if a region is a valid pupil. The centroid of a region are computed using:

$$X_c = \frac{M_{10}}{M_{00}} \quad Y_c = \frac{M_{01}}{M_{00}}$$

This module is triggered every frame clock. It copies the list of moments computed in the previous module, to allow concurrent processing of the next frame, and prepares a list of regions centroids. This compact list can be easily transferred to an application system, such as a PC. Every location can be expressed in a 19 bit number for a 640x480 frame. A USB (Universal Serial Bus) connection is used to transfer the pupil locations to a PC. A valid bit for the centroids list is updated every time the list is ready. This bit is poled by the USB to ensure correct locations read from the hardware.

4. Simulink Model

The functionality of the system is captured at an abstract level using a rigorously defined model of computation. Models of computation are formalisms that allow capturing important formal properties of algorithms. For example, control algorithms can be often represented as finite-state machines, a formal model that has been the object of intense study over the years. Formal verification of properties such as safety, deadlock free, and fairness is possible without resorting to expensive simulation runs on empirical models. Analyzing the design at this level of abstraction allows finding and correcting errors early in the design cycle, rather than waiting for a full implementation to verify the system. Verifying a system at the implementation level often results in long and tedious debugging sessions where functional errors may be well hidden and difficult to disentangle from implementation errors.

In addition to powerful verification techniques, models of computation offer a way of synthesizing efficient implementations. In the case of finite-state machines a number of tools are available in the market to either generate code or to generate Register Transfer Level descriptions that can be then mapped efficiently into a library of standard cells or other hardware components.

4.1. Synchronous Dataflow Model

The system processes large amounts of data in a pipeline and is synchronized with a pixel clock. A synchronous dataflow model of computation is the most appropriate for this type of application. A synchronous dataflow is composed of actors or executable modules. Each module consumes tokens on its inputs and produces tokens on its outputs. Actors are connected by edges and transfer tokens in a first-in first-out manner. Furthermore, actors may only fire when there is a sufficient number of tokens on all inputs for a particular actor.

All of the major system modules described in section 3 are modeled as actors in the dataflow. Data busses, which convey data between modules, are modeled as edges connecting the actors.

Several Engineering Design Automation (EDA) tools can be used to analyze and simulate this model of computation. We decided to use Simulink over other tools such as Ptolemy, System C, and the Metropolis meta-model because of its ease of use, maturity and functionality.

Simulink provides a flexible tool that contains a great deal of built-in support as well as versatility. The Matlab tool suite that includes arithmetic libraries as well as powerful numerical and algebraic manipulation tools is linked into Simulink. Thus providing Simulink with methods for efficiently integrating complex sequential algorithms into pipeline models and allowing synchronization through explicit clock connections.

The image subtraction and thresholding processes corresponding pixels from the current and previous frames on a pixel-by-pixel basis. Input tokens for this actor represent pixels from the sensor and the frame buffer, while output tokens represent thresholded pixels.

Computing connected components processes data accumulated over image lines. This module's actor representation fires only when it has accumulated enough pixel data for a complete line. It outputs only a single token containing all the data structures for the connect components list including object ID tags, component moments, and whether they are connected to any other components already processed. These output tokens are generated on every line of data relative sensor's pixel data.

These tokens are passed to a module that updates the properties of all known regions in the frame. Input is received on a line-by-line basis from the connect components module and regions in the list are updated accordingly. This module accumulates the data about possible pupil regions over an entire frame of data. Thus the output token is generated per frame of data.

The regions data per frame is then passed to an actor that calculates the XY centroid of the regions. This is accomplished with a Matlab script and the XY data output is generated for every frame.

The executable model of computation supported by Simulink has given us a platform to correct errors in the algorithms and in the data passing. It forces the system to use a consistent timing scheme for passing data between processing modules. Finally, this model provided us with a basis for a hardware implementation, taking advantage of a pipeline scheme that could process different parts of a frame at the same time.

5. Hardware Implementation

After verifying the functional specification of the model using Simulink, hardware components were selected and the logic circuit was implemented in HDL. Figure 2 depicts the main hardware blocks. Figure 6 shows a picture of the prototype.

The image capture device is a Zoran 1.3 Mega pixel CMOS Sensor, mounted with a 4.8mm/F2.8 Sunex lens providing 59 degrees field of view. It provides a parallel video stream of 10 bits per pixel at up to 27Mpps, along with pixel-, line-, and frame-clocks. In video mode, the sensor can provide up to 60 frames per second at NTSC resolution of 640x480. The control over the sensor is done through an I^2C bus that can configure a set of 20 registers in the sensor to change different parameters, such as exposure time, image decimation, analog gain, line integration time, sub-window selection, and more. Each of the two infrared light sources is composed of a set of eight LEDs [Mor00]. The LEDs are driven by power transistors controlled from the logic circuit to synchronize the on-axis, off-axis illumination to the CMOS sensor frame rate. The LEDs illuminate at near-Infrared (IR) wavelength and are non obtrusive to human.

The video is streamed to a FPGA where the complete algorithm was implemented. An Altera Apex20K device was selected. It provided enough logic gates as well as the required running speed. The FPGA controls the frame buffer FIFO (Averlogic AL422B device with an easy read and write pointers control). Communication with the application device, usually a PC, is done through a Cypress USB1.1 controller. The USB provides an easy way to transfer pupil data to the PC for display and verification. It also allows easy implementation of the I^2C communication and control of the sensor and of algorithm parameters.

6. Experimentation

The prototype system was tested using circular targets, 7mm in diameter, made of a red reflective material of the kind used for car reflective stickers. These are very directional reflectors, mimicking the eye retro-reflection.

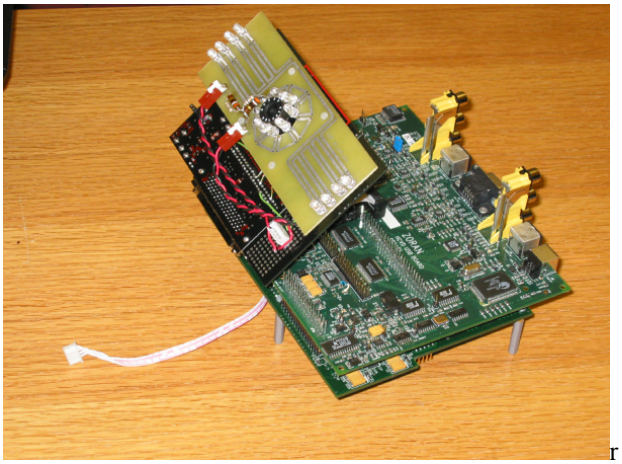 r

Figure 6: System prototype. The lens is circled with the on-axis IR LED-s. Most of the development board is left unused, except of one FPGA and a frame buffer.

Two pairs of targets were placed on a cardboard attached to horizontal bearing tracks and manually moved along a 25cm straight line in front of the sensor, at different locations and distances (about 40-60cm). The system output (x,y) coordinates of the targets was logged and analyzed. Figure 7 shows the result of one such test.

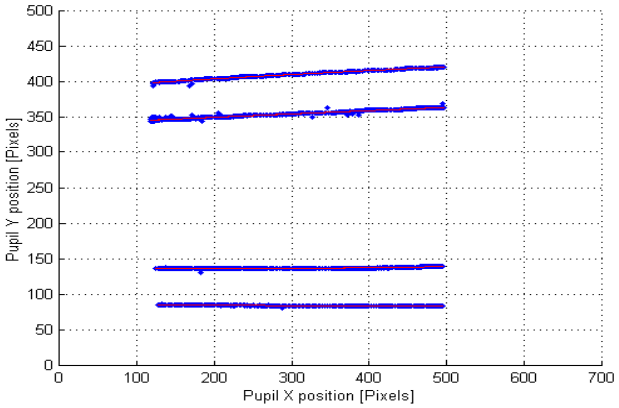

Figure 7: Results of one experiment. The detected (*x,y*) coordinates of four linearly moving targets along approx. 1500 frames are marked with blue dots, and superimposed with their polynomial fits (red).

Table 1 summarizes the results of 10 experiments, total of 15,850 frames. The average false positive detection rate is 1.07%. The average misses rate is 0.2%. Position errors are measured by the fluctuation of data points from a 2nd order polynomial fit to each of the targets in each experiment. The RMS error is 0.9 pixels, and the average absolute error is 0.34 pixels. Note that the position error computation incorporates all outliears.

Target 3 has a noticeable higher RMS error than the others. To this point the reason is still unclear. One limitation of the prototype design was the lack of any video output, e.g., to a monitor. Since the FPGA is directly connected to the sensor, a video output has to be implemented in the FPGA. This could help in monitoring the experiments and finding the cause for differences between the four targets.

Table 1: Experimentation results

Target	Deletion rate	Insertion rate	Position err (RMS)	Position err (abs)
1	0.00000	0.00151	0.344	0.251
2	0.00006	0.01154	0.566	0.290
3	0.00378	0.02415	1.803	0.474
4	0.00398	0.00543	0.884	0.358
Average	0.00195	0.01066	0.899	0.343

7. Conclusions and Future Work

The prototype system developed in this work demonstrates the ability to implement eye detection using simple logic. The logic can be pushed into the sensor chip, simplifying the connection between the two and providing a single-chip eye detection sensor. More capabilities can be added, such as glints detector for eye gaze tracking.

The advantages of hardware implementation over software/CPU are lower manufacturing price, ability to work at high frame rates and low operational complexity (it does not crash). The disadvantages are lower flexibility for changes, more complicate development cycle and limited monitoring ability.

Several points require more work. An optional video output would support monitoring during development and debugging. Work at high frame rates requires very sensitive sensors due to the short exposure time. This limits the range of eye detection in the current prototype. An appropriate optics could improve this deficit. Higher order moments would improve filtering of false positives.

In this paper, we explored one extreme of the possible implementation range: a full hardware solution. Our methodology based on the capture of the processing algorithms at a high level of abstraction is ideal to explore a much wider range of implementations from a full software realization to intermediate solutions where part of the functionality of the algorithm is implemented in hardware and others in software. We wish to compare these other implementations and to use automatic synthesis techniques to optimize the design and its implementations. The Metropolis framework under development at the Department of EECS of the University of California at Berkeley under the sponsorship of the Gigascale System Research Center, can be used as an alternative to or in coordination with Simulink-Matlab to formally verify design properties and to evaluate architectures early in the design process as well as to automatically synthesize implementations.

8. Acknowledgements

We thank Zoran CMOS Sensor team and Video IP core team for providing the sensor module and for the support. We are also grateful to Tamar Kariv for the help with the USB drivers and PC application, and to Ran Kalechman and Sharon Ulman for their good advice with the logic synthesis.

9. References

[Bal82] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice Hall, New Jersey, 1982, pp. 150-152.

[Bal02] F. Balarin, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, M. Sgroi, and Y. Watanabe, "Modeling and Designing Heterogeneous Systems," in J. Cortadella and A. Yakovlev, editors, LNCS: Advances in Concurrency and Hardware Design, Springer-Verlag, pp. 228-273, 2002.

[Bir97] S. Birchfield, An elliptical head tracker, in Proc. of the 31st Asilomar Conf. on Signals, Systems, and Computers, Pacific Grove, CA, November 1997.

[Coz01] A. Cozzi, M. Flickner, J. Mao, S. Vaithyanathan, A Comparison of Classifiers for Real-Time Eye Detection. ICANN 2001: 993-999.

[Dav01] J. W. Davis and S. Vaks, a perceptual user interface for recognizing head gesture acknowledgements, In IEEE PUI, Orlando, FL, 2001.

[ETRA2] Andrew T. Duchowski, Editor, Proceedings of the Symposium on Eye Tracking Research & Applications, March 2002, Louisiana.

[Ebi93] Y. Ebisawa and S. Satoh, effectiveness of pupil area detection technique using two light sources and image difference method, In A.Y.J. Szeto and R.M. Rangayan, editors, Proc. of the 15th Annual Int. Conf. of the IEEE Eng. in Medicine and Biology Society, pp. 1268-1269, San Diego, CA, 1993.

[Har01] I. Haritaoglu, A. Cozzi, D. Koons, M. Flickner, D. Zotkin, Y. Yacoob, Attentive Toys, in Proc. Of ICME 2001, Japan, pp. 1124-1127.

[Har00] A. Haro, I. Essa, and M. Flickner, Detecting and tracking eyes by using their physiological properties, in Proc. of Conf. on Computer Vision and Pattern Recognition, June 2000.

[Hje01] E. Hjelm and B.K. Low, Face Detection: A Survey, Computer Vision and Image Understanding, vol. 83, no. 3, pp. 236-274, Sept. 2001.

[Hyr97] A. Hyrskykari, "Gaze Control as an Input Device", in Proc. of ACHCI'97, University of Tampere, pp. 22-27, 1997.

[Hut89] T. Hutchinson, K. W. Jr., K. Reichert, and L. Frey. Human-computer interaction using eye-gaze input. IEEE Trans. on Systems, Man, and Cybernetics, 19:1527-1533, Nov/Dec 1989.

[Jac91] R.J.K. Jacob. The use of eye movements in human-computer interaction techniques: What you look at is what you get. ACM Transactions on Information Systems, 9(3):152-169, April 1991.

[Kap02a] A. Kapoor and R. W. Picard, Real-Time, Fully Automatic Upper Facial Feature Tracking, Proc. of 5th Int. Conf. on Automatic Face and Gesture Recognition, May 2002.

[Kot96] R. Kothari and J.L. Mitchell. Detection of eye locations in uncon-strained visual images. In Proc. ICPR Vol. I, pp. 519-522, Lausanne, Switzerland, Sep. 1996.

[Mor00] C. Morimoto, D. Koons, A. Amir, and M. Flickner, Pupil detection and tracking using multiple light sources, Image and Vision Computing, 18(4):331-336, March 2000.

[Oli97] N. Oliver, A. Pentland, and F. Berard. Lafter: Lips and face real time tracker. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, pages 123-129, Puerto Rico,PR, June 1997.

[Per00] K. Perlin , S. Paxia , J. S. Kollin, An autostereoscopic display, Proc. of the 27th annual conference on Computer graphics and interactive techniques, p.319-326, July 2000.

[Sal00] D. D. Salvucci and J. R. Anderson, Intelligent gaze-added interfaces, Proc. of ACM CHI 2000, pages 265-272, New York, 2000.

[San02] A. Sangiovanni-Vincentelli, *Defining Platform-based Design,* EE Design, March 5, 2002.

[Sta81] Stark, L., and Stephen Ellis. 1981. Scanpaths Revisited: Cognitive Models Direct Active Looking. In Eye Movements: Cognition and Visual Perception, edited by D. F. Fisher, R. A. Monty and J. W. Senders. Hillside, NJ: L. Erlbaum Associates.

[Sti96] R. Stiefelhagen, J. Yang, and A. Waibel, A model-based gaze tracking system, in Proc. of the Joint Symp. on Intelligence and Systems, Washington, DC, 1996.

[Tom89] A. Tomono, M. Iida, and Y. Kobayashi, a TV camera system which extracts feature points for non-contact eye movement detection, in Proc. of the SPIE Optics, Illumination, and Image Sensing for Machine Vision IV, volume 1194, pages 2-12, 1989.

[Ver02] R. Vertegaal, I. Weevers and C. Soln, Queen's Univ., Canada Gaze-2: an Attentive Video Conferencing System, Proc. CHI-02, Minneapolis, 2002, pp. 736-737.

[You75] L. Young and D. Sheena. Methods & designs: Survey of eye movement recording methods, Behavioral Research Methods & Instrumentation, 7(5):397-429, 1975.

[Zha99] S. Zhai, C. Morimoto and S. Ihde, Manual And Gaze Input Cascaded (MAGIC) Pointing, In Proc. ACM CHI-99, pp. 246-253, Pittsburgh, 15-20 May 1999.

[Zha00] W. Y. Zhao, R. Chellappa, A. Rosenfeld, and P. J. Phillips, Face recognition: A literature survey. UMD CfAR Technical Report CAR-TR-948, 2000