A Multiprocessor System for Performing Mail Sorting in Real Time

D. L. Andrews, R. Brown, C. Caldwell and A. R. Hennessey

Department of Electrical Engineering, University of Arkansas, Fayetteville, U.S.A.

This paper describes a special purpose embedded multiprocessor architecture and algorithms developed for performing real time Multi-Line Optical Character Recognition (MLOCR). MLOCR is a computationally intensive real time application involving character image extraction, gray scale thresholding, rotation and scaling of individual characters, and character identification. The computational complexity of the algorithms implemented in the system required custom hardware in a parallel processing environment to meet the real time system requirements. The algorithms as well as the functional partitioning of the algorithms into parallel processing subsystems are discussed.

1. Introduction

The United States Postal Service (USPS) has invested heavily in automation technology to help defer the ever increasing challenges of a competitive business environment [1]. The University of Arkansas is one of many institutions that have been performing research and development for the USPS in the area of Multi Line Optical Character Recognition (MLOCR) systems design [2]. An MLOCR system is a machine used for automatically sorting envelopes at high speed. The functions required to provide a sort decision for a single envelope are computationally intensive and include character image extraction, gray scale thresholding, rotation and scaling of individual characters, and character identification. This paper describes work now nearing completion on a multi-year project to design and develop a small and inexpensive MLOCR machine. This paper discusses the algorithms implemented, and tradeoffs performed in partitioning the computationally intensive algorithms into multiple asynchronous subsystems capable of meeting the real time requirements. This critical step defines the necessary level of parallelism in the system, and also identifies those algorithms that must be implemented in custom hardware to meet real time requirements. The interface specification and data flow between subsystems are also discussed, detailing the real time bus bandwidth and derived subsystem timing requirements that result from algorithm partitioning. The organization of the software is then discussed, outlining the key issues that must be addressed in development of software operating under hard time constraints.

MLOCR Requirements

The functions and algorithms implemented in an MLOCR system are listed in Table 1. Although these functions and algorithms can be computed sequentially on a single CPU machine, the MLOCR system must be capable of performing all processing for each envelope within a window of 100 milliseconds. In order to meet this requirement, the algorithms must be partitioned and executed in parallel. A structured top down design approach was developed for partitioning and mapping the algorithms into individual parallel subsystems. Each subsystem has been assigned a derived timing requirement based on the overall system requirement. The results of mapping algorithms into the various subsystems and derived timing requirements are also listed in Table 1.

The following sections present the algorithms implemented in each subsystem listed in Table 1.

Function	Algorithms	Subsystem	Requirements
Capture and Digitize Image	Capture and Digitize Image	Camera	Synchronize with transport
Store and Transfer Image	Store and Transfer Image	Buffer Board	Transfer 25 MBytes/Sec.
Locate Address Blocks	1–D Pixel Decimation 1–D Image Smoothing 1–D Edge Detection 1–D Edge Smearing Connected Component Analysis 2–D Filtering	VABL	15 msec./Envelope
Separate Characters	Thresholding Connected Component Analysis 2–D Box Filtering 2–D Word Filtering 2–D Line Filtering	HABL	10 msec./Candidate Address Block
Scale, Rotate, Threshold	2–D Interpolation 2–D Thresholding 2–D Windowing	SRT	200 usec./Character
Identify Characters	2–D Template Matching	OCR	200 usec./Character
Control Asynchronous Data	Data Queues and Scheduling	Controller	10 Envelopes/Sec.

Table 1. MLOCR Component Requirements

2. Subsystem Definition

Camera

The camera is responsible for capturing the envelope image as it is transported along a conveyer belt at high speed. The image data is represented as 8–bit gray scale, and is captured at a synchronous rate proportional to the speed of the transport. The operation of the camera is shown in Figure 1. As an envelope moves past the lens of the camera, one column of 1024 pixels is captured every clock strobe. The data is transferred along a synchronous, processor-memory bus [3], first to the buffer board, and then to the VABL board.

The synchronous rate of image capture is determined by the speed of the conveyer belt. One column of pixels is recorded for each clock strobe. The clock strobe signals are set to a frequency corresponding to the speed of the traveling envelope. If the clock strobes are not set in proper proportion, then the envelope image will suffer distortion. If the clock strobe frequency is too fast, then the captured image will appear stretched in length. If the camera clock is too slow, then the captured image will appear as compressed. Either way is not acceptable, since image distortion can greatly affect character recognition results. Ideally, each pixel represents a square area 4.72 mils on a side.

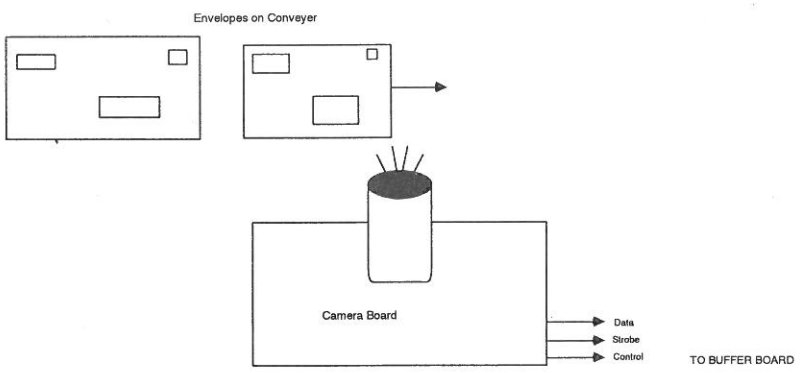

Fig. 1. Camera Operation

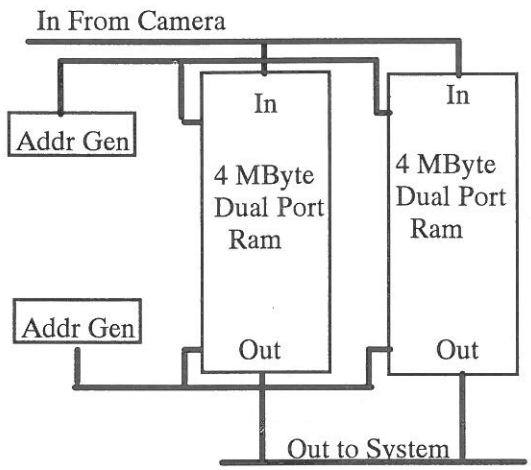

Fig. 2. Buffer Board Block Diagram

Buffer Board

The camera data is input across a high speed dedicated synchronous bus into the buffer board. The buffer board is responsible for storing envelope image data in two interleaved, dual ported RAM buffers. While data is being received by the buffer board, it is also preprocessed to perform camera correction and image enhancement. The improved quality image is then stored in one of the two buffers in an alternating, ping- pong fashion. Figure 2 shows the block diagram of the buffer board memory. Alternating buffers allows processing to continue on the previous envelope while the next envelope image is being acquired. The pixel values input from the camera are stored in the buffer board as 8–bit gray scale data. The dual ported RAM buffers must be large enough to store all pixel data from an envelope. The maximum size is given by the 1024 pixels read vertically by a maximum of 3072 columns. This requires 4 MBytes for each of the two ping pong buffers.

While the buffer board is acquiring camera data, it may also receive a request to transfer image data to another processing module. The request may be either for image data on the current envelope being acquired, or for image data from the previous envelope stored in the alternate buffer. The buffer board is capable of transferring image data from either buffer while it is receiving, processing, and storing camera data to one of its buffers. This is an enhancement over the original buffer board design that only allowed data to be read from the buffer that had finished storing data for the previous envelope. This enhancement allows processing to start on an envelope while the envelope is still being captured. Figure 3 shows the effects on timing of this enhancement.

As shown in Figure 3, it takes approximately 80 msecs to read an envelope into the buffer board. As the speed of the transport only allows 125 msecs between the time the front of the envelope is passed by the camera and the time the envelope will arrive at the sort gate, all processing for an envelope must occur in this 125 msec window. If the processing cannot start until all envelope data has been read into the buffer board, then processing cannot start until 80 msecs. This leaves only 45 msecs for all processing. By allowing data to be read out of the buffer board as the envelope data is still being input into the buffer board, processing can be initiated at approximately 15 msecs after the first data is entered. This start time is nondeterministic, and is dependent on the physical location of the address and other printed information on the envelope.

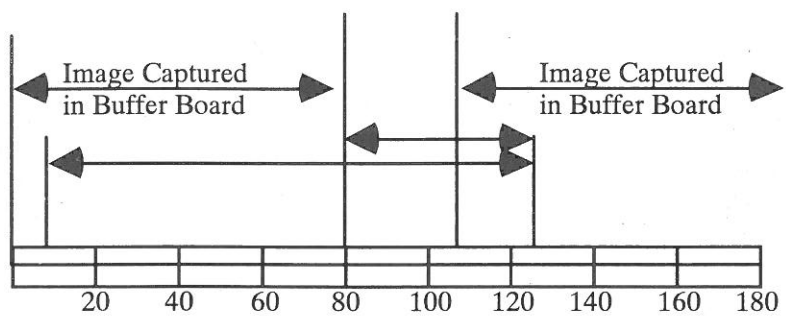

Fig. 3. Timing Enhancement

Vertical Address Block Location (VABL)

The second and most critical step in processing an envelope is to identify all Candidate Address Blocks (CABs). This function is performed in the Vertical Address Block Location (VABL) subsystem. While receiving input data for an envelope, the VABL subsystem may output multiple partial results. As soon as one or more CABs have been located on the envelope, VABL outputs its findings to the system controller. Outputting partial results from VABL as quickly as possible is a key issue in meeting the real time requirements of the system. The V in VABL refers to the fact that the processing is done in vertical raster scan order as the camera supplies pixels in vertical raster scan order. VABL does not make use of the buffer board.

The first partial result is typically output from VABL approximately 25 msecs after the starting edge of the envelope is detected. The actual time when a result is output from VABL is random, and is dependent on the exact position and size of the address block on an envelope. However, envelopes are input to the system with the stamp edge of the envelope passing the camera first, and the return address of the envelope passing the camera last. With this orientation, the actual destination address block is frequently located by VABL fairly early in the process. VABL assigns a rank to every CAB indicating the probability that the CAB contains the actual destination address. The VABL weight value is a rough estimate based on the CAB's dimension, line structure, and position on the envelope.

Only pixels in the area on the envelope identified as a candidate address block are required for subsequent processing. As such, the output of this function serves as the transition between the high bandwidth requirements of the camera bus, and a greatly reduced requirement for processing individual address blocks. The complete envelope image is reduced to just a handful of CAB images. The capability of the system to correctly sort an envelope is directly dependent on correct identification of the destination address block. As such, this first step is critical in the correct operation of the system. Figure 4 shows the processing flow of VABL.

The first processing VABL performs is to binarize the image "on the fly". This is done by identifying the white to black and black to white transitions. A binary image is needed for the connected component [4] analysis used to locate candidate address blocks. This step in performed in dedicated hardware that thresholds [5] the difference in gray scale levels between adjacent pixels.

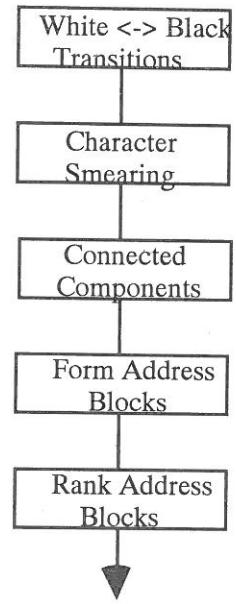

Fig. 4. VABL Processing Flow

A Δ is computed for each pixel relative to pixels above. Whenever a black pixel is identified, it is smeared to the right. Each black pixel is smeared to the right a fixed number of pixels, n, which is greater than the inter character spacing. This smears letters together, converting an entire line of text into a single connected black region. Identification of the transitions and the smearing can be seen in Figure 5. A pixel at (x,y) in the smeared image will be black if any pixel on the line segment between (x–n,y) and (x,y) is black in the binary image.

VABL determines the boundary boxes of connected black regions in the smeared image. These boxes will usually each contain one line of text in an address block. The lines of text are grouped together with nearby lines into candidate address blocks by software. VABL ranks the CABs based on uniformity of line height, spacing, justification, and position of the CAB on the envelope. Thresholding, smearing, and connected component boxing are all done in custom hardware in raster scan fashion.

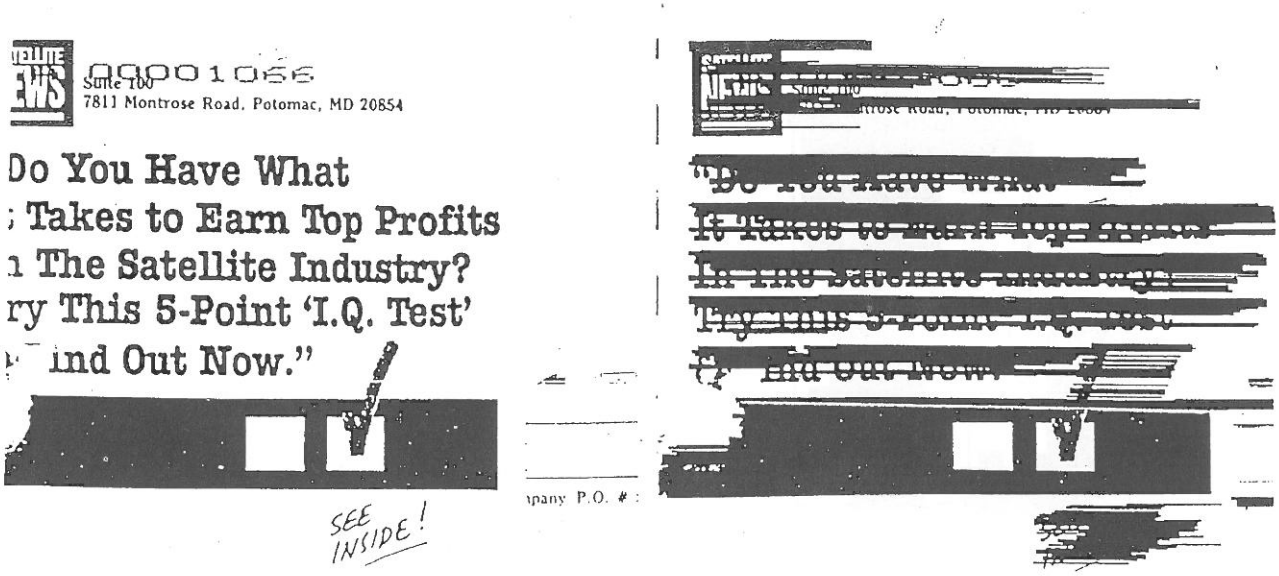

Fig. 5. Character Smearing

Connected component analysis is done in vertical raster scan order by growing connected components one column at a time. The black pixels in any one column form a set of vertical black line segments. Each of these line segments, or intervals, is part of some connected component. If an interval touches an interval in the column to the left, it is part of that intervals connected component. If one interval touches two or more intervals in the column to the left, their connected components must be joined into a single one. The following pseudo code describes the growing of connected components.

```
Initialize connected component list to empty:
Initialize interval list for column 0 to empty:
for ( n= 1; n <= number of columns; ++n)
  {
  Determine black intervals in column n;
  for ( m =1; m <= number of intervals in column n; ++m)
  {
  if(interval m touches no intervals in column n--1)
    start new connected component;
  else if(intervals m touches one interval in column n--1)
    {
      label interval m as part of its connected component;
      update bounding box of that connected component;
      }
    else
      { /** touches two or more intervals **/
      join the connected components for these intervals (if different);
      re-label intervals in them;
      label interval m as part of resulting component;
      update bounding box;
      }
    }
  }
```

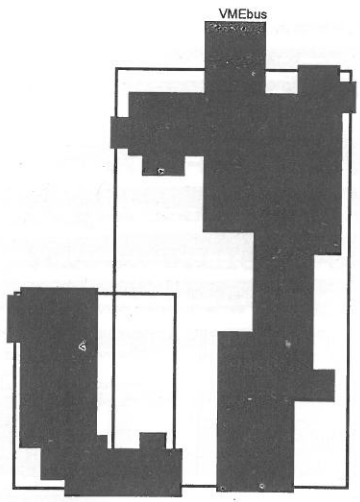

Fig. 6. Two Connected Regions of a Broken Letter

Horizontal Address Block Location (HABL)

After an address block has been identified, pixels within the address block area are processed to find the individual characters. This is performed in the Horizontal Address Block Location (HABL) subsystem. The HABL module receives CAB images from the buffer board, locates all characters within the CAB, and groups them into words. Characters are identified by locating connected black components in the image. HABL returns the bounding boxes of these components as letter boxes. It tries to merge the boxes of broken characters, and tries to eliminate "flyspecks" and other noise. HABL then attempts to group the letters into lines and words.

Connected component analysis is performed in HABL using the same algorithm described in VABL, but in a horizontal raster scan order. Intervals in a row are compared to intervals in the two rows immediately above rather than to just one row above. This patches together some slightly broken characters. Other broken characters can be patched together by noting that the boundary boxes of their pieces overlap. This is shown in Figure 6 for a broken "J".

HABL requires a better thresholding algorithm than VABL to avoid broken or merged characters. Missing pixels are not crucial in VABL since smearing covers these missing pixels. HABL's thresholding algorithm calculates thresholds using the minimum and maximum intensities in each of a fixed grid of small windows. Three different thresholds are used, and the resulting boxes are combined in software. Thresholding and connected component boxing are performed in dedicated hardware. The remainder of the processing is performed in software.

HABL re-ranks CABs based on letter box structure, and assigns its own weight value to indicate the probability of the CAB containing the destination address. The HABL weight value is considered to be more accurate than the previously determined VABL weight.

Scale, Rotate, Threshold (SRT)

Characters can appear in different fonts and sizes, and may also be of varying contrast levels. The SRT subsystem processes each character located by HABL. The first step taken is to scale and rotate the character image to a standard dimension of 16×24 bytes, with each byte representing a single pixel in 8 bit gray scale format. While scaling, the SRT algorithm can remove up to ± 7 degrees of skew by rotating the character. Rotation of the character image is necessary when the address block is not placed squarely on the envelope. These two operations can be combined into a single operation [6] and performed simultaneously. Each character image can be viewed as a 2D array of pixel points. The matrix operations for scaling and rotation can be combined into a single operation given by:

$$[x', y'] = [x\, y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$$

$$= [x\, y] \begin{bmatrix} S_x \cos\phi & S_x \sin\phi \\ -S_y \sin\phi & S_y \cos\phi \end{bmatrix}$$

where S_x and S_y are the x–dimension and y–dimension scaling factors, respectively. These equations represent the general form for computing the new points of a scaled and rotated character. In a real time system, computing cosf and sinf is very costly. The angle ϕ in our application is limited to ± 7 degrees. In order to meet the real time requirements, $\sin\phi$ and $\cos\phi$ are pre computed for $-7 < \phi < +7$ in 1 degree increments and stored in a lookup table. The

equations applied to a character are now given as

$$x' = xS_x \cos\phi - yS_y \sin\phi$$
$$y' = xS_x \sin\phi + yS_y \cos\phi$$

When computing the new coordinate values x', y', the index values computed for accessing the pixels from the original character may not fall evenly on a pixel value. For example, assuming $S_x = 3$, $S_y = 2$, $\phi = 5$,

$$x' = x3\cos 5 - y2\sin 5 = x2.988 - y.1742$$
$$y' = x3\sin 5 + y2\cos 5 = x.2613 + y1.992$$

Now, consider transforming the two pixel locations [0,0] and [2,1]. In matrix notation,

$$N[0\,0] = O[0\,0]\begin{bmatrix} 2.988 & .2613 \\ -.17421 & .992 \end{bmatrix} = O[0,0]$$

$$N[2\,1] = O[2\,1]\begin{bmatrix} 2.988 & .2613 \\ -.17421 & .992 \end{bmatrix} = O[5.8, 2.5]$$

$N[0\,0]$, the new pixel value at 0,0, is the pixel from the original character at coordinate 0,0, $O[0,0]$. However, $N[2\,1]$, the new pixel value at 2,1 is the pixel from the original character at coordinate 5.8,2.5. In this case the scale/rotate algorithm performs a linear interpolation using the fractional values to compute a weighted average gray scale value for the new pixel.

Finally, the SRT subsystem performs thresholding on the character image. Yet a third thresholding method is needed for SRT, since the subsequent Optical Character Recognition (OCR) function is even more sensitive to thresholding errors than HABL. Better thresholding is possible as the number of pixels to threshold is smaller. Thresholding reduces a character image of 8–bit gray scale data requiring 384 bytes of storage, to a binary character image of 16×24 bits, requiring only 48 bytes of storage.

Two different thresholding algorithms are performed on each character: window thresholding, and line thresholding. Line thresholding is implemented independently in both the vertical and horizontal directions. A majority vote is performed on the output of the three results to determine if the pixel should be 0 or 1. Each of these methods operates on the basic principle of comparing the gray scale value of a pixel to a computed neighborhood average. If the pixel is darker than the average, then the pixel is assigned a value of 1. If the pixel is brighter than the average, then the pixel is assigned a value of 0. The general operation of the two methods (horizontal and vertical line thresholding only differ in the axis used) are discussed below

Window Threshold

A 5×5 sliding window is used to compute the average pixel intensity value for the window thresholding function. The average, $I_{avg.}$, for pixel P_{xy} can be stated as

$$I_{avg.} = \frac{1}{25}\sum_{i=-2}^{2}\sum_{j=-2}^{2} P_{x+i\,y+j}$$

After the average is computed for pixel P_{xy}, the pixel value is compared to the average and set to 1 or 0 accordingly.

Line Threshold

The line thresholding technique uses a vertical or horizontal slice of the character. In contrast to window thresholding which computes a separate threshold for each pixel, line thresholding produces the threshold values for a line segment at a time. This is achieved by thresholding the line in segments between critical points. A critical point is a pixel with a local minimum or maximum value. The threshold used for pixels between two critical points is the average of the intensities of the critical points. The approximate algorithm (for horizontal line thresholding) can be stated as:

```
for row i
start = P0,i
for col = 1 to 16 in row i
  if Pcol,row is local minima,maxima{
    threshold = (start + Pcol,row) / 2
    for all pixels between start,Pcol,row{
      if (Pixel < threshold) Pixel = 1
      else Pixel = 0
    }
  start = Pcol,row
}
```

The outputs of all three algorithms are input into a majority vote circuit that determines the final 1,0 value for each pixel.

The complexity of the combined thresholding algorithm is 0(n), where n is the number of pixels representing the character. Although the algorithm is only O(n), all 384 pixels must be processed in the 200 usec requirement. This allows only 520 nsecs of processing per pixel. The per pixel processing of the window threshold alone requires calculating the average of 25 8 bit pixel values. This average must be computed in under 520 nsecs. Due to the amount of processing required for each character the SRT subsystem must be implemented on a custom designed processor using high speed logic. The output of the SRT subsystem is a Binary Character Image (BCI) character.

Optical Character Recognition (OCR)

Finally, the OCR subsystem receives BCI data output by SRT and performs one of several template matching algorithms to determine the character identity. If a unique identification cannot be obtained, the OCR module will output a list of possible matches. The system permits multiple OCR methods to be implemented in parallel to increase the probability of a unique identification. This module also processes a vast amount of data to obtain its results. The BCI input is compared with about 100 templates, contained in a data base of over 5,000 templates. Since the OCR module must also meet a throughput requirement of one character every 200 usecs, it also is implemented in custom designed hardware.

System Controller

The system controller is the focal point of the MLOCR system. All other modules in the system are designed to perform a specific function at high speed. The system controller commands each of the subsystems to perform their individual tasks, and then maintains the results. All asynchronous I/O is communicated with the system controller through a message passing protocol [7].

The system controller must operate in real time to insure that processing is completed for all envelopes within a specified period. It is the responsibility of the system controller to provide the sort decision before the envelope reaches the sort gate. To perform its requirements in real time, the controller is interrupted and data driven to respond immediately to all system messages. In addition, the controller maintains watchdog timers on various events to prevent subsystems from exceeding their allocated time frames.

The system controller is responsible for coordinating the processing of image data as it propagates through all subsystem components. The data consists of a combination of synchronous camera data containing the digitized image, and asynchronous message data containing the intermediate results of processing modules. The controller must set the conditions for processing modules to continuously receive inputs and communicate results.

3. System Organization

Each subsystem is implemented on one or multiple processor modules. As shown in Figure

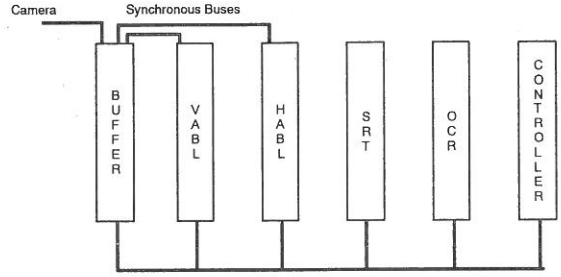

Fig. 7. MLORC System Architecture

Subsystem	Hardware Requirements
Camera	1-Custom Board
Buffer Board	1-Custom Board
Vertical Address Block Location (VABL)	2-Custom Boards, 1–Commercial Board
Horizontal Address Block Location (HABL)	3–Custom Boards, 1–Commercial Board
Scale, Rotate, and Threshold (SRT)	2–Custom Boards
Optical Character Recognition (OCR)	2–Custom Boards
System Controller	1–Commercial Board

Table 2. Subsystem Hardware Implementation

7, all subsystems are connected by an industry standard VME bus [8]. Besides the global VME bus, some processor modules are also equipped with a high speed synchronous bus. The common VME bus is used primarily to communicate control messages across the system. The VME bus can also be used, however, to transfer small portions of image data. Using an industry standard bus eliminates the cost of designing multiple data interfaces for each subsystem, and also promotes an open systems approach [9]. Image transfers requiring higher bandwidth are routed through one of the dedicated, synchronous buses. Image data enters the system over a 16–bit, synchronous bus from the camera running at 14 MHz. The camera data is stored on the buffer board. Unbuffered data can be transferred directly from the camera to HABL along a 16 bit synchronous 14 MHz bus. Data stored in the buffer board can be transferred to HABL along a 32–bit bus running at 10 MHz. The buffer board is also capable of transferring data across the global VME bus to anywhere in the system.

By using the industry standard VME bus, the system is very flexible with regard to compatibility of hardware. Most processor boards in the system are custom designed to perform a specialized task at high speed. Each of the subsystems can be substituted, however, with commercial off-the-shelf (COTS) hardware and system software. The application of COTS proved useful in prototyping and testing the system. However, the system cannot meet overall throughput requirement of real-time operation without most subsystem modules being implemented in custom hardware.

Most of the subsystems require custom hardware to meet the real time requirements. Some of the subsystems require multiple processor boards. In some cases this includes a combination of custom hardware and commercial off-the-shelf (COTS) hardware. Algorithms that could be executed in software and still meet system timing requirements were implemented on COTS hardware. The Motorola 147 and 165 boards were used to host software based algorithms. Table 2 provides a breakdown of each subsystem into its hardware implementation.

The system controller is the only component that does not require custom hardware. Instead, the controller is based solely on a commercial, Motorola 147 board. The 147 board features a 68030 processor, a full VME bus interface, and access to timer registers and mailboxes. The functionality of the system controller is implemented entirely in software. While the software has been written for the Motorola 147 board as its target, it is also highly portable and can be adapted to any general purpose VME based processor.

4. System Data Flow and Timing

Data flow through the system is shown in Figure 8. Subsystem components are represented by rectangular boxes, and data items are represented using cigar shaped boxes. Synchronous camera data is represented by boxes without shading, while asynchronous data is represented by shaded boxes. The system controller plays no direct role in acquiring camera data, or in providing that data to the buffer board or VABL. The controller is only responsible for initializing the buffer board and VABL to enable data acquisition. All other data transfers in the system are driven by messages from the system controller.

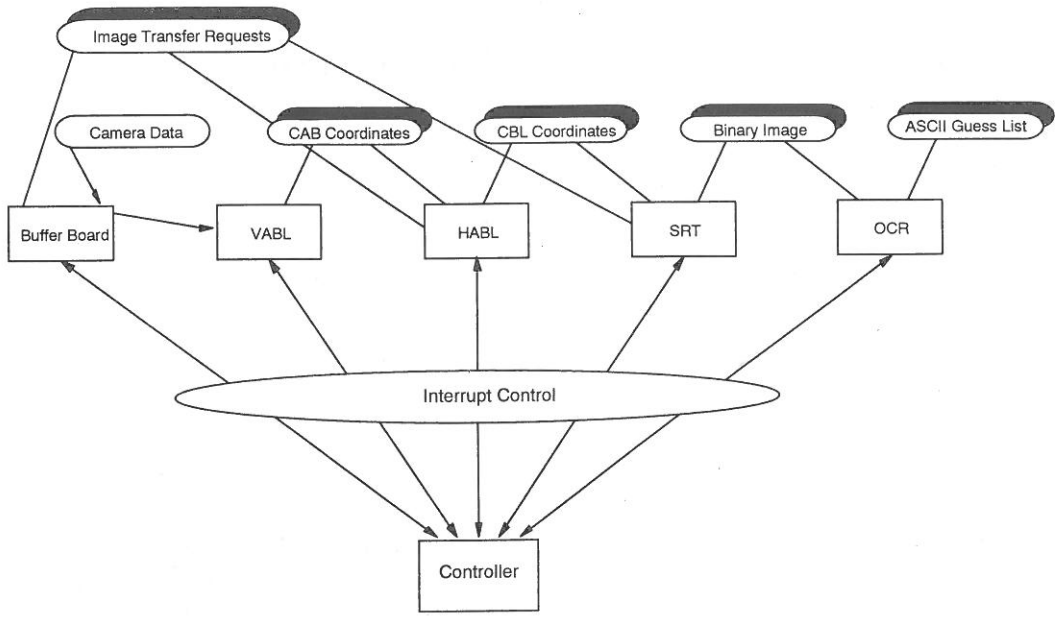

Fig. 8. System Data Flow

The diagram in Figure 8 shows data flowing directly between processing modules without any system controller interaction. This is not the case in actuality. The diagram is merely representative of the data items that flow between modules, and is not indicative of the communications route taken. All subsystems communicate results only with the system controller. The system controller implements these data transfers as standard format and protocol messages.

Routing all data into and out of the controller using a standard transfer protocol provides several advantages to the overall system. First, logging utilities built into the controller provide debug support throughout the complete MLOCR system. In the event of a system failure, a log file can be output from the controller. This log file can be analyzed based on time stamp recordings to determine when and where a subsystem failed. The time stamped data can be unrolled to trace backwards all messages exchanged. The time stamped messages also provide timing information for each subsystem, allowing identification of any system bottlenecks.

Second, routing all data into and out of the controller allowed a set protocol for communications to be established, independent of the exact implementation of each subsystem. This protocol is discussed in more detail in the following sections.

Priority Interrupt Design

The interrupt driven design of the system controller provides for continuous data flow between the asynchronous subsystems. Each subsystem is continuously kept busy and in full operation as long as data is available for processing. Each time the controller receives output data from a system component, a message is immediately sent to supply that component with its next input. If the next component module in the pipeline is not already active, then it also is immediately sent work to process. Assuming every subsystem is tuned so that no bottlenecks occur, every module sustains continuous operation.

The interrupt routines perform the sorting of available data to be sent to the next subsystem. In this fashion, each subsystem provides more precise information concerning the determination of the destination address. The first weight value provided by the VABL subsystem is computed based on the dimensions, line structure, and position of the CAB on the envelope. Each CAB position and corresponding weight value are sent to the system controller. The system controller uses the VABL assigned weight to prioritize work for the HABL. The results of HABL, and more specifically, the HABL assigned weight is used to prioritize work to SRT and OCR.

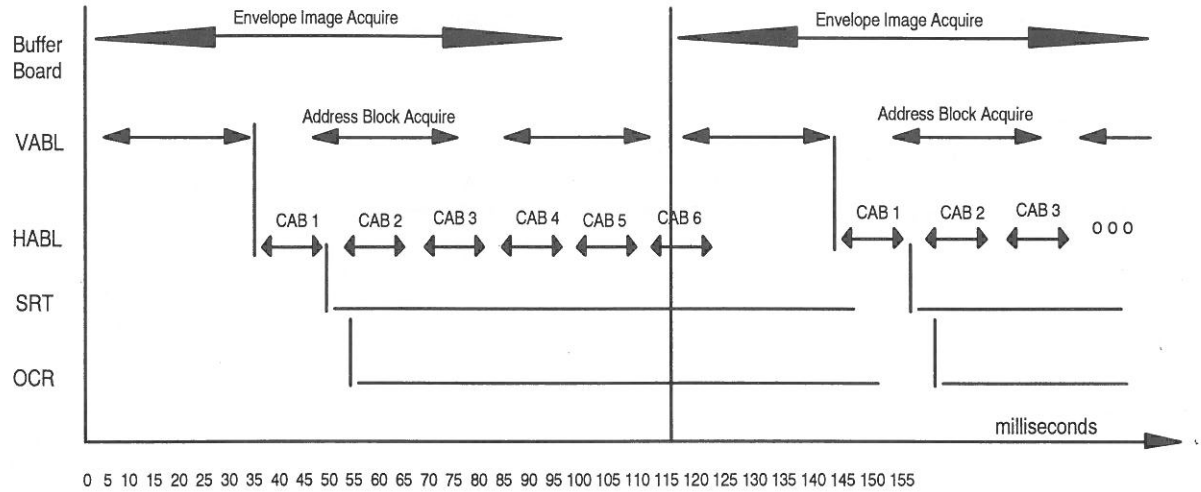

Fig. 9. Data Processing Timeline

The diagram in Figure 9 illustrates the timing of the MLOCR data. In the time line shown for a typical envelope stream, it is assumed that the envelope entering the system at time t=0 is the first envelope and that all processors are idle. The buffer board starts acquiring the image at t=0 and VABL starts looking for Candidate Address Blocks. At approximately 25 msecs, VABL outputs a message containing all CAB's found thus far. The highest priority CAB is then sent to HABL for processing. When HABL returns, the first character data is available for SRT to process. When SRT returns at approximately 40 msecs, the first character is available to be sent to OCR. At this time, all processors are in full operation.

Each board executes asynchronously and interrupts the controller when output data is available. If more input data is available when a processor sends its output to the controller, then execution can continue on that processor. Otherwise, the board is left in an idle state until more input data is obtained from the preceding board's output.

When image acquisition for the next envelope begins at approximately 100 milliseconds, the HABL, SRT, and OCR subsystems may still be processing information extracted from the previous envelope. The CAB's and characters from the previous envelope continue to be processed while CAB's for the next envelope are being generated. In this way, all MLOCR subsystems sustain full operation. After the first envelope has been processed, subsequent envelopes appearing in rapid succession are effectively overlapped to maintain full capacity of the MLOCR pipeline.

A key design consideration of the MLOCR system is the non-deterministic nature of the processing involved. Envelopes are non-uniform in size, and the destination address block is not constrained to be of a particular size or occupy a specific set of coordinate values on the envelope. This non-determinism requires each subsystem to operate asynchronously, processing a varying amount of data for each envelope. The VABL subsystem may identify only two or three CAB's on one envelope, and may identify five or six CAB's on the next envelope. Each CAB contains a variable amount of information for further processing, including a varying number of words, characters, and font sizes. In order to coordinate the processing of each asynchronous subsystem, the system controller coordinates the transfer of information between subsystems based on its data driven, interrupt paradigm. The overall objective of the controller is to provide the correct sort decision in the quickest possible time. In order to expedite processing of the asynchronous data, the controller maintains prioritized work queues for each subsystem. Output from one subsystem is sorted and placed into a pending work queue for the next subsystem. The work queues are sorted and updated in data driven interrupt service routines.

5. Communication Protocol

The standard communications protocol uses a combination of registers and interrupts operating in a predefined handshake sequence. This provides for seamless integration of various application specific hardware into the system. All data transfers are defined as full 32–bit long word transfers, taking advantage of the full VME bus bandwidth. Also, all system components, except the controller, implement the standard register interface shown in Figure 10.

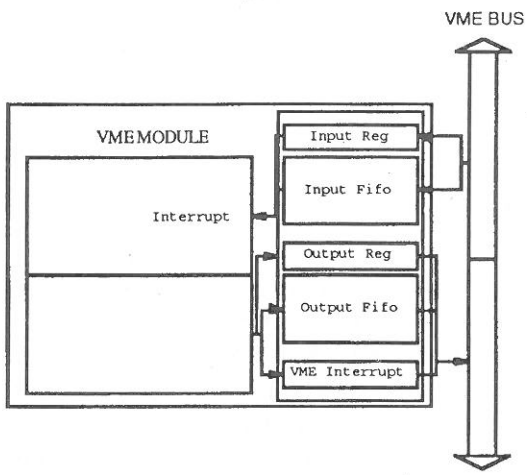

Fig. 10. Subsystem I/O Registers

The system controller sends a message to a component by writing the message data into the module's 32–bit input FIFO. After the data has been placed into the FIFO, the controller notifies the module that it has a message to process by writing a 32–bit op code to the board's input register. Writing the op code to the input register generates an internal interrupt on the board. The module responds to the interrupt by reading the system message from its input FIFO and performing the specified operation. Results are communicated back to the system controller by a similar message passing scheme using the board's output registers. The output message is written to the board's 32–bit output FIFO, a 32–bit op code is written to the output register, and then a VMEbus interrupt is raised to inform the system controller that a message is ready. The controller completes the cycle by reading the output message from the interrupting board's registers.

6. Software Organization

System controller functions are all software based. However, the module is required to perform low level register access on multiple custom hardware devices. The controller accomplishes this while using general purpose, commercial hardware. The software has been designed using the Object Oriented Paradigm [10] in C++ to maximize performance and maintain portability.

An important objective of the system controller is to keep all boards in the system running at full capacity. The throughput requirement breakdown for each module, which has been given in Table 1, was generated by considering an average envelope and assuming that data is always available for each module to process. This assumption requires there to be minimum lag time as messages and data are passed between modules. The system controller has no explicit requirements listed in Table 1. Instead, the system controller's requirement is merely to provide an environment that ensures system components meet their individual requirements.

The system controller is built upon a number of software sub-modules. These sub-modules, hereafter referred to simply as modules or components, provide all services required by the system controller. The modules are implemented using both C++ classes and C code. These major components of controller software are 1) Data Queues; 2) Data Management; 3) Timer Services; and 4) Interrupt services.

Data Queues

The system controller requires data queues in many different places and for different reasons. The requirement for multiple instances of a basic queue operation was implemented in object oriented C++ classes. The first place where a queue structure was needed was in providing a very fast implementation of dynamic allocation. Since the nature of most of the MLOCR data was variable length and variable size, dynamic allocation of class objects was a convenient tool for data management. Real time constraints of the system made it inadvisable to use the language provided utilities of "new" and "delete" for this purpose. Instead, pools of fixed sized

objects were created and managed using data pointers and queues.

The controller also instantiates queues to store input data for all system components. As output data is received from one processor in the system, input data is acquired for the next. These input data are maintained in data queues. When a board is ready for its next input, the data can be simply retrieved from the specific queue for the processor.

Data Management

The foundation of the Data Management module is the Data Management container class. The Data Management class is actually an empty class: it contains no member functions or data. The only purpose of the container class is to create a common pointer for accessing data of various other class types. There are four data classes that inherit from the Data Management container class: these classes are 1) the Envelope class; 2) the Cab class; 3) the Word class; and 4) the Character class. Since each of the four data classes inherits the Data Management class, each can be referenced by a common class pointer type.

The Data Management module makes use of this inheritance relationship to maintain a tree structure of MLOCR data as shown in Figure 11. The Data Management pointer is also useful in preserving the current data for processing modules. Each time a processor board is sent input data, the current location in the envelope tree structure is saved with a set of Data Management pointers.

The tree structure reflects a basic truth about the nature of envelope data. Each envelope can be expected to have one or more blocks of text that will be searched for the destination address. Each of the candidate address blocks can be expected to contain multiple character box lists, which are roughly equivalent to words. Finally, each word can have one or more characters. Therefore, the system controller requires exactly one envelope class data structure, and all its sub-trees, to process an envelope through the MLOCR pipeline.

Timer Services

The MLOCR system controller is real-time software responsible for insuring that the processing for each envelope is completed by the time the envelope reaches the sort gate. Also, the controller provides watchdog timers for the VABL and HABL subsystems to insure that these subsystems do not exceed their execution time allotment.

The Controller obtains timer services from the two timer registers on the Motorola 147 board. The first register is used to support a system clock with 0.0002 second resolution. The timer register is configured to send a Clock class object an interrupt every 200 microseconds. The clock is used to record the starting times of data items as they are sent to processor modules. The clock is also used in conjunction with the

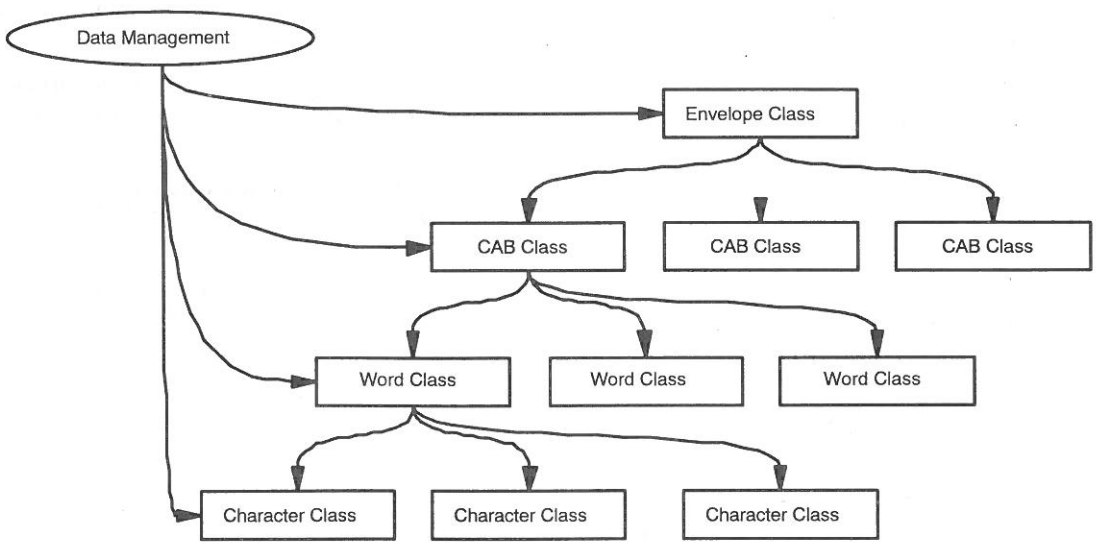

Fig. 11. Data Management Tree

second timer register to provide event timer services. A queue maintains a sorted list of event timers. The timer register is always set to interrupt for the first event in the queue. Each time an elapsed timer expires, notification is made and the elapsed timer register is reset to the next event in the queue.

Interrupt Handlers

Along with timer register interrupts, the system controller also receives VMEbus interrupts from each of the MLOCR components. The nature of these interrupts is message passing. Each time the controller receives one of these interrupts, message data is transferred across the VMEbus. The flow chart used for the message interrupt handler is depicted in Figure 12. Each time the controller receives a message interrupt, it first disables any other interrupt that would try to also access the same data as the current interrupt handler. Then, the controller retrieves the current state of the interrupting processor and copies the message data across the VME bus. If the next subsystem is currently idle, it is immediately sent input data. If there is more data available in the input queue for the current processor, it is also sent another data entry. Finally, any disabled interrupts are restored and the interrupt service routine is terminated.

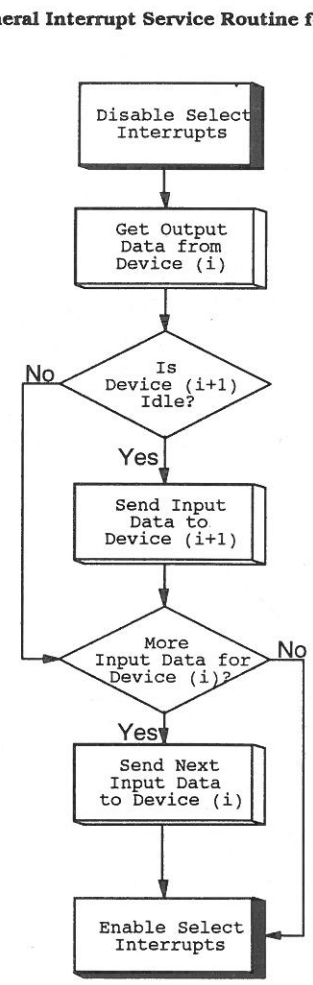

Fig. 12. Interrupt Service Routines

Conclusion

The multiprocessor architecture, algorithms, and controller software presented have been fully implemented in a prototype system. Custom hardware components for the VABL, HABL, and the OCR subsystems have been designed, debugged, and integrated into the system along with general purpose, commercial off-the-shelf hardware. The prototype system used the same VME bus architecture and communications protocol as specified for the real time system. This has facilitated the integration of custom hardware modules, and served as a proof of concept verification of the MLOCR system.

References

1. *Pricing Postal Services in a Competitive Environment*, Washington, D.C.: General Accounting Office, Report to Congress, 1992.

2. PAUL W. PALUMBO ET AL. , "Postal Address Block Location in Real Time", *IEEE Computer*, Vol. 25 No. 7, 1992, pp. 34–42.

3. PATTERSON AND HENNESSY, *Computer Organization and Design, The Hardware / Software Interface*, San Mateo, CA: Morgan Kaufmann, 1994.

4. PRATHER, R. E., *Discrete Mathematical Structures for Computer Science*, Houghton Mifflin Company, Boston MA. 1976.

5. GONZALEZ, R. C., WOODS, R. E., *Digital Image Processing*, Addison Wesley, 1993.

6. FOLEY, J. D., VANDAM, A., *Fundamentals of Interactive Computer Graphics*, Addison Wesley, 1983.

7. DAVID ANDREWS ET AL. , "A Parallel Architecture for Performing Real Time Multi-Line Optical Character Recognition", *Proceedings of The Twenty-fifth Southeastern Symposium on System Theory*, Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 533–536.

8. *Standard for a Versatile Backplane Bus: VMEbus ANSI/IEEE Std 1014–1987*, New York, NY: Institute of Electrical and Electronic Engineers, March 1988.

9. BINH PHAN, BEN MCLEOD, GUS CAVAR, JASBIR SANDHU AND JOHN MAMPE, "An OCR Architecture for Postal Applications", *Proceedings of the Advanced Technology Conference*, Washington, D.C.: The United States Postal Service, 1992, Vol 3, pp. 1339–1345.

10. GRADY BOOCH, *Object Oriented Design*, New York, NY: Benjamin Cummings, 1991.

Received: January, 1994
Accepted: September, 1994

Contact address:
Department of Electrical Engineering
University of Arkansas
Fayetteville, Ark. 72701, U.S.A.
e-mail: dla@engr.engr.uark.edu

DAVID ANDREWS received his B.S.E.E. and M.S.E.E. in Electrical Engineering from the University of Missouri-Columbia in 1983 and 1984, respectively. He received the Computer Engineer Degree and Ph.D. in Computer Science from Syracuse University in 1990 and 1992, respectively. Dr. Andrews worked at General Electric in Syracuse New York, performing research and development for GE in the areas of algorithms and parallel architectures, and was also the technical lead in the development of a fully distributed, fault-tolerant operating system developed for the Seawolf Submarine. Since joining the faculty of Electrical Engineering at the University of Arkansas in 1992, Dr. Andrews has been performing research in the areas of computer architecture, parallel processing, and optimal designs for scalable parallel machines based on multichip modules. Dr. Andrews is a member of the IEEE.

RANDY BROWN received his B.S. in Physics from the University of Missouri-Rolla in 1971, and a Ph.D. in Physics from the University of Wisconsin-Madison in 1978. Dr. Brown performed post-doctoral work at the University of Wisconsin's Integrated Circuit Laboratory for two years. He then joined the faculty of Electrical Engineering at the University of Arkansas-Fayetteville in 1981. His research interests include integrated circuit design, image recognition, neural networks, and computer algorithms.

CHARLES CALDWELL received his B.S.E.E. and M.S.E.E. in Electrical Engineering from the University of California, Berkley in 1964 and 1965, respectively and the Ph.D. from Case Western Reserve University in 1970. He then held a NSF post-doctoral research position at the Laboratory for Medical Electronics and Biocybernetics at the University of Ljubljana, Yugoslavia (now Slovenia) until 1972. Since that time he has been associated with the Department of Electrical Engineering, University of Arkansas, Fayetteville. Dr. Caldwell's interests encompass instrumentation, digital circuits, microprocessor applications and more recently, FPGAs. Dr. Caldwell is a member of IEEE.

ARTHUR HENNESSEY received his B.S.E.E. and M.S.E.E. in Electrical Engineering from Portland State University in 1986, and the University of Arkansas in 1994, respectively. Mr. Hennessey performed software engineering for Boeing and General Electric Companies from 1986 to 1992, and is currently employed by GTE in Chicago where he working on communications software for Airfone applications. Mr. Hennessey's research interests are in software development for embedded real time systems.