RSD-TR-5-83

MULTI-BUS COMPATIBLE FRAME GRABBER

J. L. Turney

J. V. Hill

May 1983

Center for Robotics and Integrated Manufacturing

Robot System Division

COLLEGE OF ENGINEERING
THE UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN 48109

1. *INTRODUCTION*

A vision system generally consists of a camera, a digitizer, a computer, and a video interface such as a frame grabber. In the system the camera and digitizer provides digital video data at a rate of several megabytes per second. For example, the General Electric TN2500 digitizing camera provides an 8-bit pixel (picture element) at a rate of 4.5 million pixels per second. At this rate of incoming data a computer cannot directly perform meaningful processing on the data. Therefore, vision systems, contain a hardware interface between the camera and the computer to capture incoming video data and to hold the data until the computer has time to process the captured data. This is the role of a frame grabber.

Essentially, a frame grabber is a dual port memory with one port of the memory connected to the camera and the other port connected to the computer. A video frame of data from the camera is digitized and brought in, i.e. "grabbed", through one port and held in the frame grabber's memory until the computer has time to process the data, which it accesses through the second port. Once the image data has been captured it may be thresholded, or edges in the image data may be determined, or any other processing may be applied to it.

In the Robotics Lab, we employ several Intel 8086 based boards as dedicated processors for vision applications. We decided to build a Multi-Bus[1] compatible frame grabber board which would operate in conjunction with our General Electric TN2500 digitizing camera, in order to provide ourselves with a system component which would occupy little space, would derive its power and timing from existing systems, and which above all would be inexpensive to reproduce. This report discusses the result of out efforts -- a simple and relatively inexpensive frame grabber (cost ~$600).

---

[1] Multi-Bus is a registered trademark of Intel Corporation.

2. *OVERVIEW*

Fig. 1 presents an overview of the board. Every 220 ns, a new pixel of digitized video information is brought in from the TN2500 digitizer (shown in the upper right) and latched into one of the buffers, B53, B52, B51, or B30 in order. The frame grabber's memory requires times greater than 220 ns to write data, so the data is temporarily buffered in B53, B52, B51, and B30. One cycle of this buffering process consists of filling the four buffers. Each cycle takes 880 ns, 220 ns for each new pixel to be brought in and written to a buffer. When the last buffer B30 is written, the other three bytes stored in B53-51 are then output simultaneously to buffers B33-31 after which the contents of B33-30 are output to the frame grabber's memory when the memory is ready to accept data.

As mentioned above 880 ns of time passes between the times that the buffers B33-30 are filled. During the first half this time, 440 ns, the frame grabber's memory is written from the buffers B33-30. The 32 bits in the four buffers B33-30 are written simultaneously.

During the later half of the 880 ns buffering time B33-30 are not connected to the frame grabber's memory, and the computer is allowed access to memory. The 880 ns provided by buffering is thus time sliced between the camera and the computer.

The memory is interfaced to the computer through bidirectional gates B62-60. The computer writes into memory through B13-10 and outputs from memory through B23-20.

Latches B43-B40 provide a interface to a video monitor, whereby the grabbed frame may be displayed. The sync signals to run the monitor are taken from the TN2500 while the actual video information does not come directly from the camera but rather is routed from the frame grabber's memory. A frame can be grabbed and displayed on the monitor without the need for onboard video timing generation logic

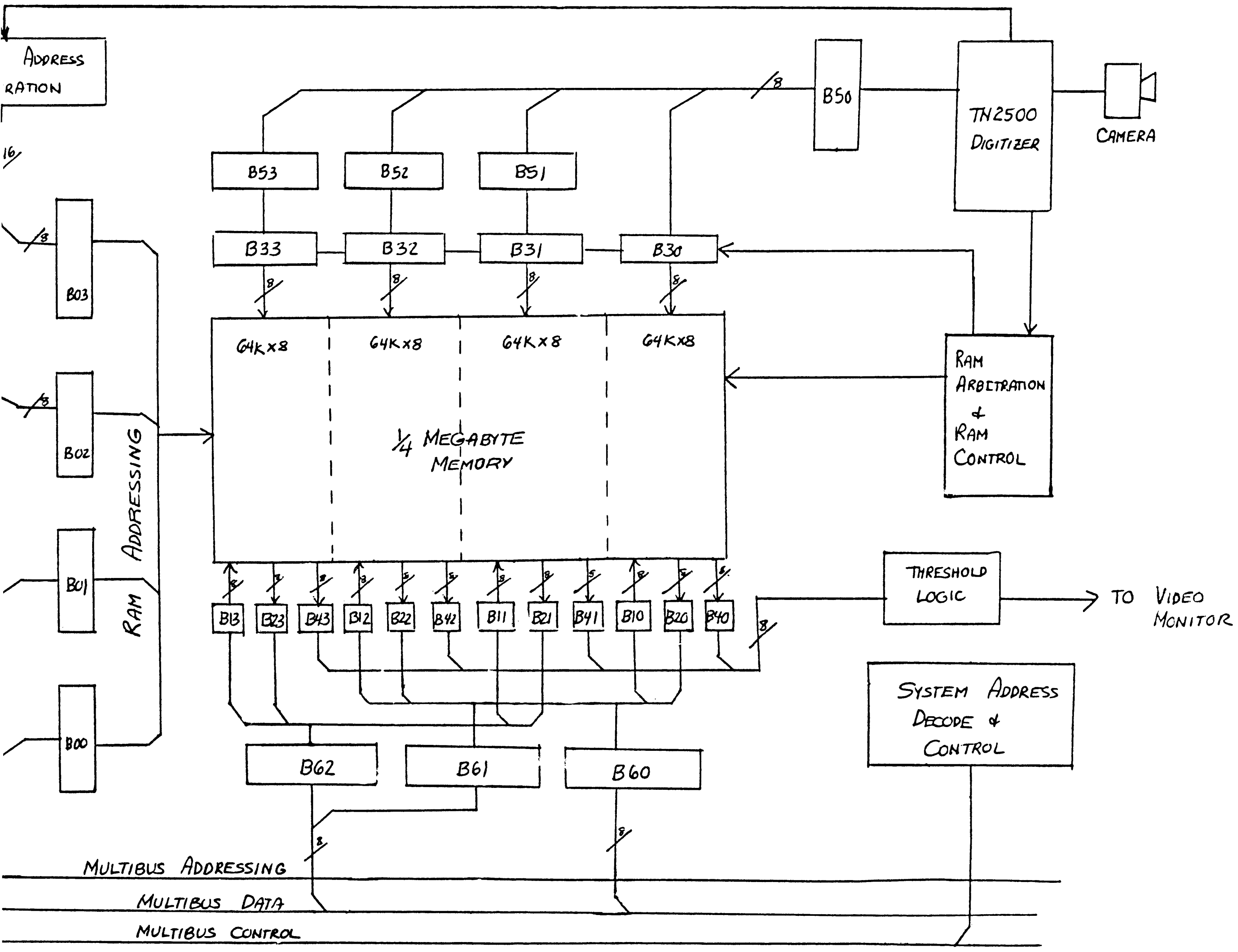


Figure 1. Overview of frame grabber.

since the frame grabber steals the necessary sync signals from signals provided by the TN2500 digitizer.

In the present configuration of the board, data coming out of the frame grabber to the monitor is thresholded before it is output to the monitor. In the future an D/A converter will convert the digital data back to analog for a gray level display. The threshold is software selectable.

Automatic addressing for the memory is generated using signals from the TN2500 digitizer (see left side) during the time the camera has access to the memory. The computer can also address the memory during its 440 ns time slice.

The frame grabber actually has 4 pages consisting of 256 x 256 bytes each of memory. Any one of these four is software selectable. Since the computer can write into each page. One page could be used to display data while the computer is operating on a different page.

Table 1 provides a price breakdown on the cost of the board, and Fig. 2 shows the actual layout of the board.

3. *PROGRAMMING*

The frame grabber, as it is presently configured, resides in Multi-Bus addresses 40000H to 7FFFFH. Jumper J1 on page A1 of the appendix can be altered to relocate the frame grabber.

memory chips	$ 320
logic chips	70
sockets	70
board	130
Total	$ 590

Table 1. Price breakdown for frame grabber.

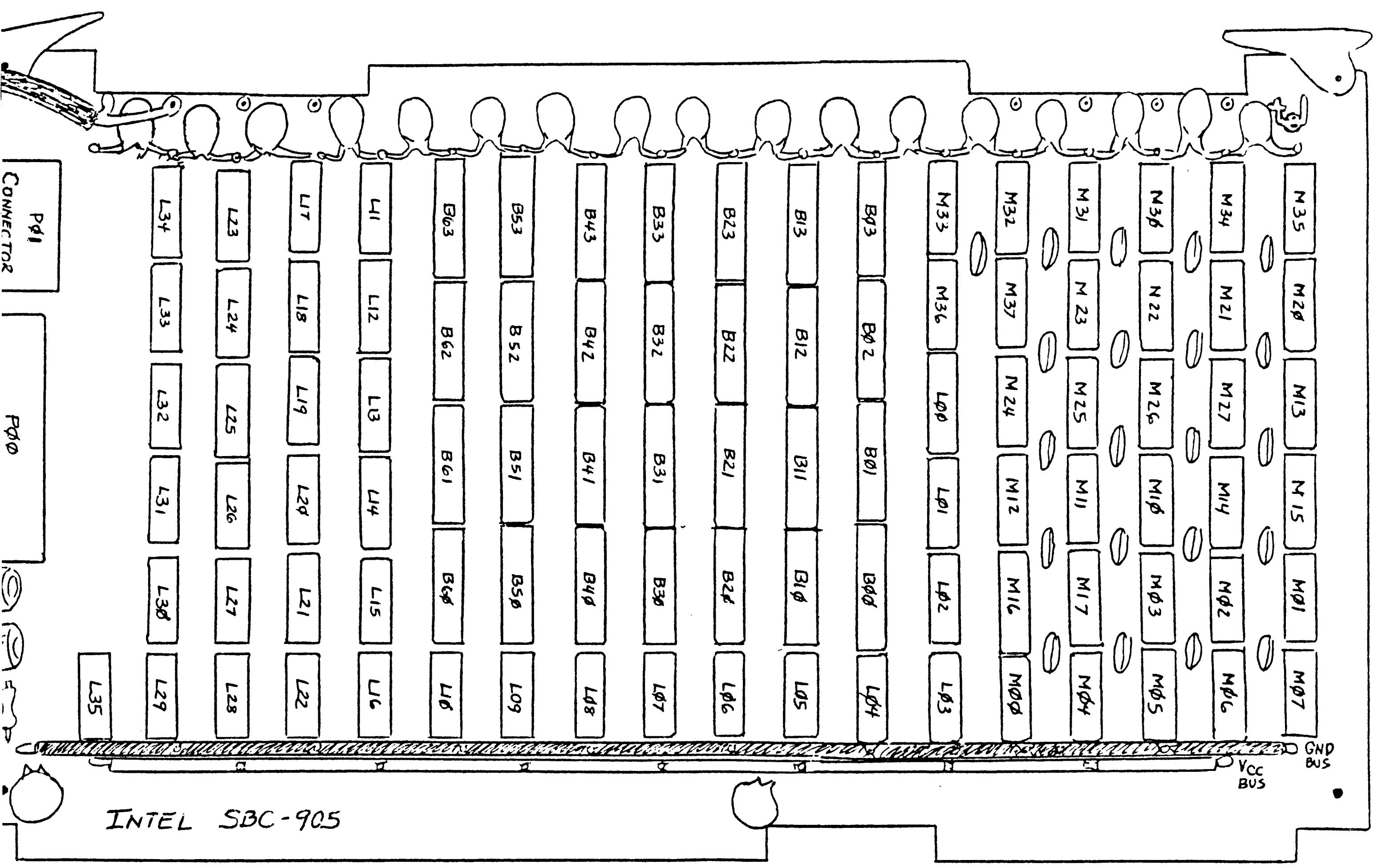


Figure 2. Board layout.

The frame grabber is control by an 8255 PIO. In order to initialize the PIO, the byte 99H must be written to address location C206H. This location may be relocated by alterating J2 and J4 displayed on page A1.

A frame is grabbed by writing 00H to address location C300H. The lowest bit of input from location C200H will be 1 during the frame grab and will go to zero once the frame has been captured. This bit may be polled to determine if the frame grab has been completed.

The monitor image is thresholded by a byte threshold set at location C202H after the PIO has been initialized. This threshold only affects the monitor image but does not affect the memory of the frame grabber or the data read from the memory to the computer. The computer must threshold the data itself once it has captured it.

Appendix B gives a program for grabbing a frame of data and converting it to a runlength encoding.

4. *DETAILED OPERATION*

In the schematics shown in Appendix A inputs are listed on the left side of the page while outputs are listed on the right. Numbers next to the inputs correspond to the page from which the input is obtained. For example, the "3" next to MXACK/ indicates that the input MXACK/ is obtained from a similarly labeled output MXACK/ which is shown on the left side of page 3. The numbers next to an output correspond to the pages on which the output appears as an input. The special prefix of "MB" indicates that the connection is to the Multi-Bus line rather than to a connection on another page.

Page A1 displays the address decode logic which decodes address from the computer to the board off the Multi-Bus. L21 selects the memory. Jumper J1 determines which quarter megabyte of the Multi-Bus memory space the memory of the board will occupy. L19 and L20 select the PIO. J2 and J3 configure the PIO's location. L19 also provides a frame grab request FGREQ/. L25 provides the required response to an IO or memory read or write, since each memory access over the Mult-Bus must be acknowledged with a XACK/.

Page A2 shows the Multi-Bus data interface. B60-62 are bidirectional transceivers.

The 4164 memory chips are dynamic rams with multiplexed address inputs. The desired 16-bit address must be placed on the 4164 memory address lines 8 bits at a time, the lower 8-bit during an RAS/ negative transition and the upper 8-bits during a CAS/ negative transition. L03 provide the RAS/ and CAS/ strobes as well as other signals. MUX is used to multiplex addresses from latches 8 bits at a time. LSTROBE strobes data out of memory into latches for output. L31 MXACK/ output provides the desired Multi-Bus response for a memory access at the correct time.

Since the TN2500 digitizer does not provide any addressing information, addresses during the camera's access to memory are generated from the element rate clock, ERC/, using a series of down counters L11-14 as shown on page A4. Outputs VA0-7 address one of the 256 pixels in each horizontal line of the image. Outputs VA8-F address one of the horizontal lines in the display. During a frame grab VA8 is a different output than during normal monitor display. This is because the TN2500 displays lines in a peculiar way. It displays odd lines twice each during the odd interlace field and even lines twice each during even interlace field. This results in an undesirable overlap of lines. Lines appear in the order 1,2,1,2,3,4,3,4,5,6,5 · · · . Using L15 together with L08 and L09 eliminates this problem and allow lines to be appear in the order as 1,1,2,2,3,3,4,4,5,5 · · · .

Page A5 shows the PIO and its connections to both the frame grabbing latches L32 and the thresholding logic L17 and L18. The thresholded data is level shifted to provide a simple video output. L15 is used to clean up the display. It pulls the output low prior to a horizontal sync to provide a uniform delay to the low caused by the sync, and it blanks the screen when portions of memory which are not filled by the frame grabber are scanned.

Page A6 shows how the memory and video address are multiplexed into the memory.

Page A7 displays the buffers responsible for latching and holding the video data during the frame grab, while page A8 shows the data interface from memory to the monitor.

Pages A10 and A11 display the data interface from and to the computer bus.

Page A11 shows how the memory chips are connected. In the interest of conserving space, only one example out of a group of eight memory chips is shown representing typical connections for members of each group.

Page A12 shows how the connector brings signal from the TN2500 digitizer to the board.

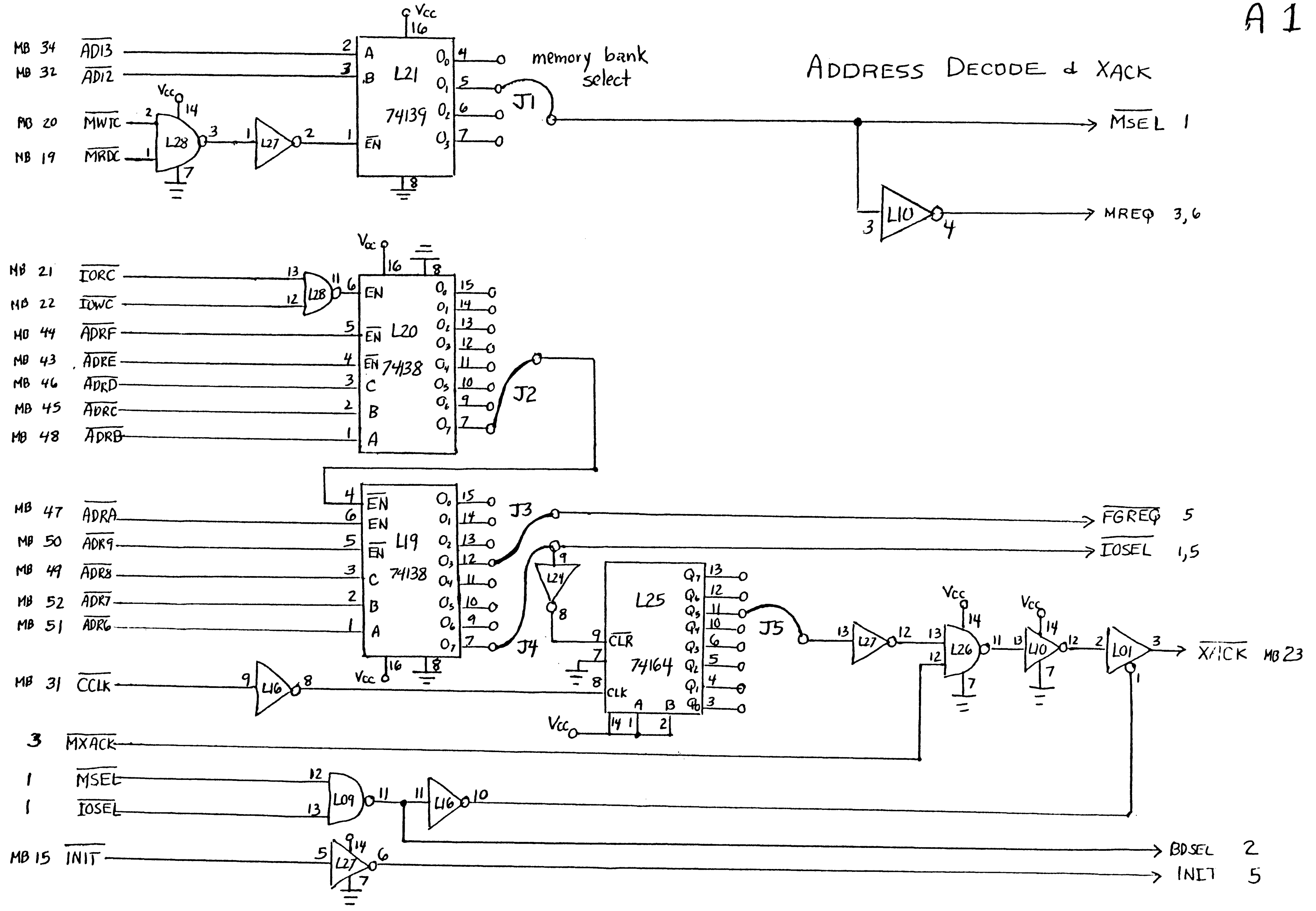
5. *FURTHER WORK*

As mentioned before a D/A convertor will be installed in the frame grabber, so the user will have the option to display either the actual frame grabbed or a thresholded version of the frame grabbed.

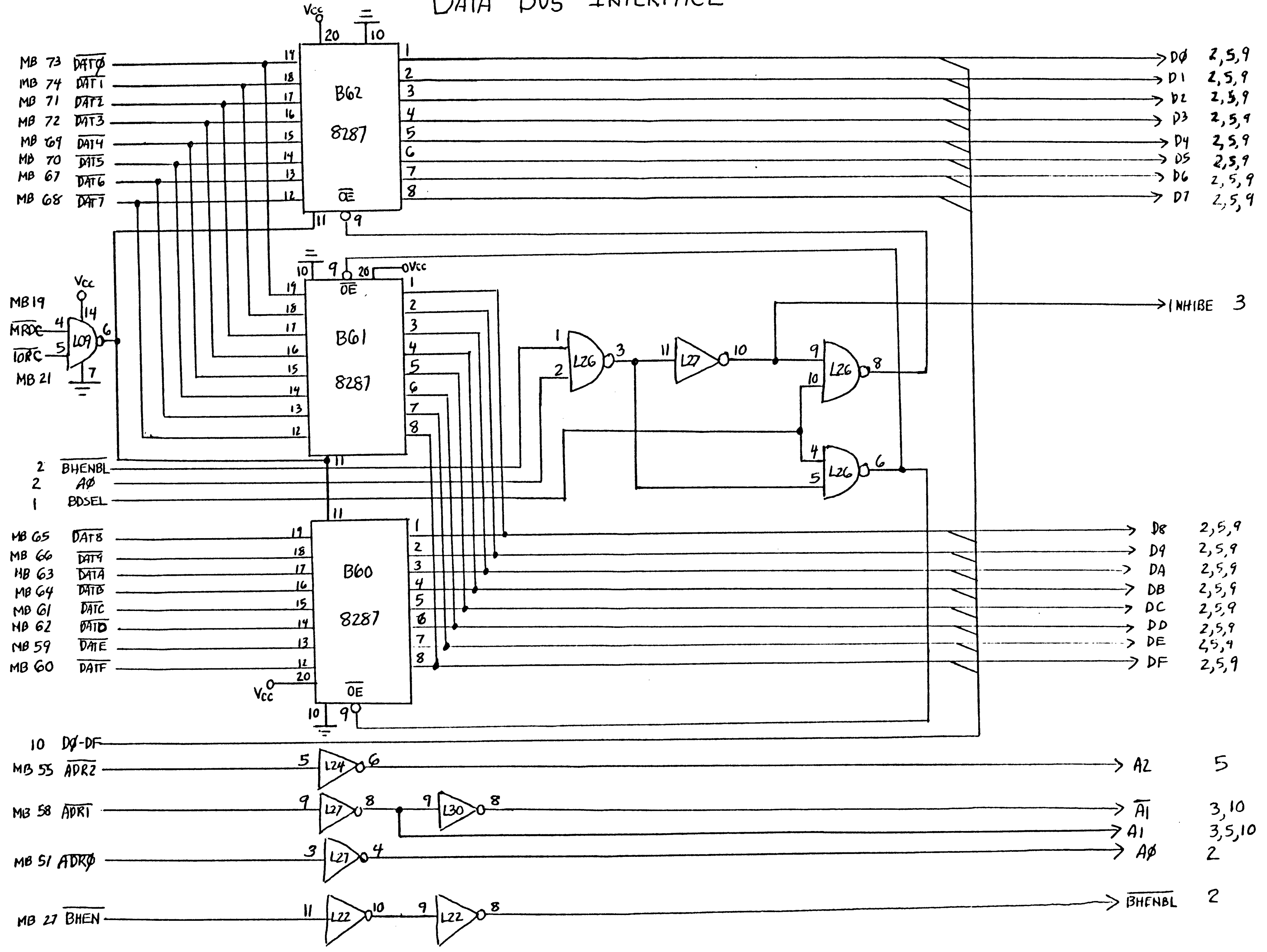
We wish to thank Bob Alsum for doing the bulk of the original design and for doing the tedious job of wire wrapping the board.

6. APPENDICES

Address Decode & XACK

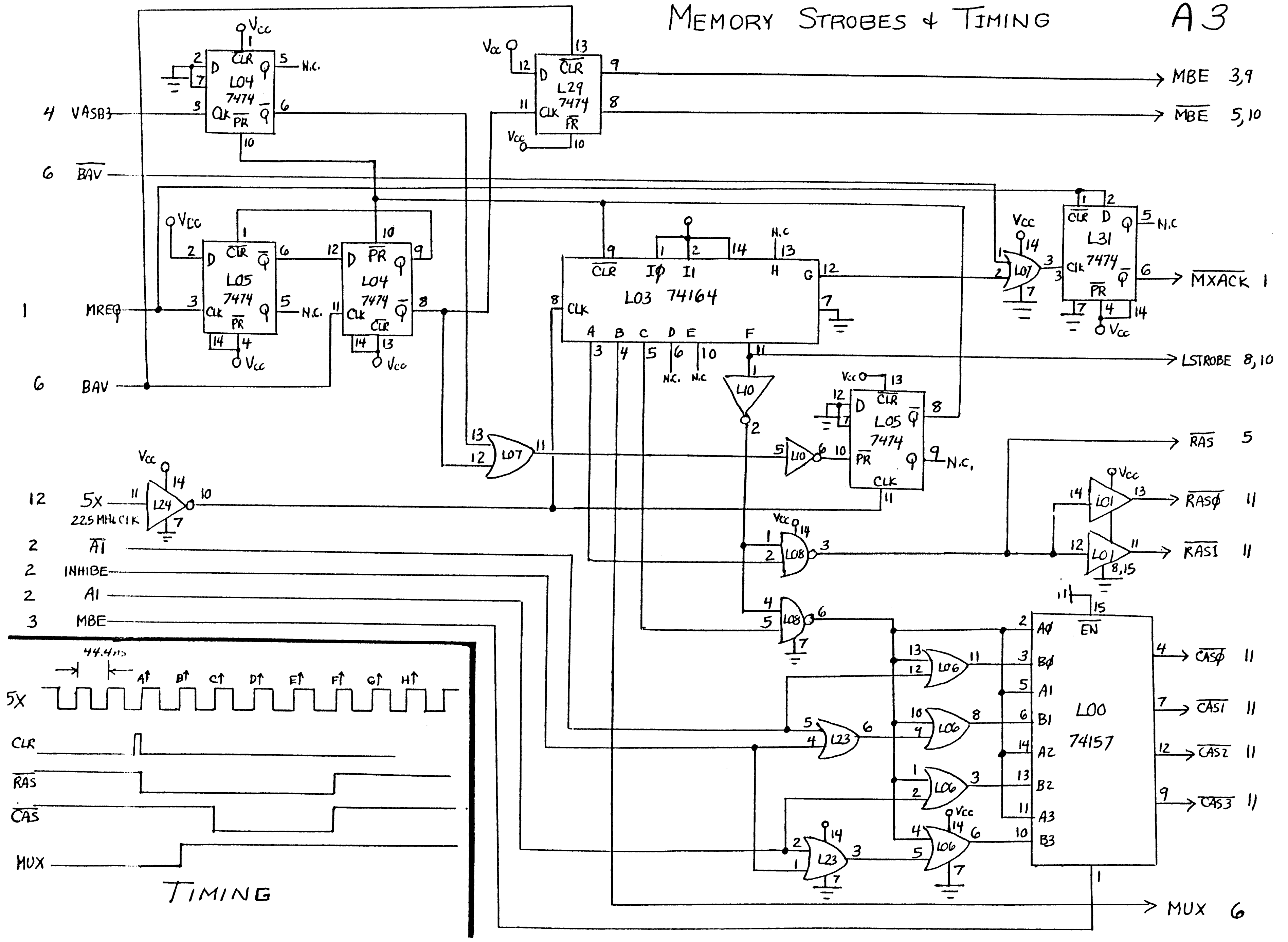


DATA BUS INTERFACE

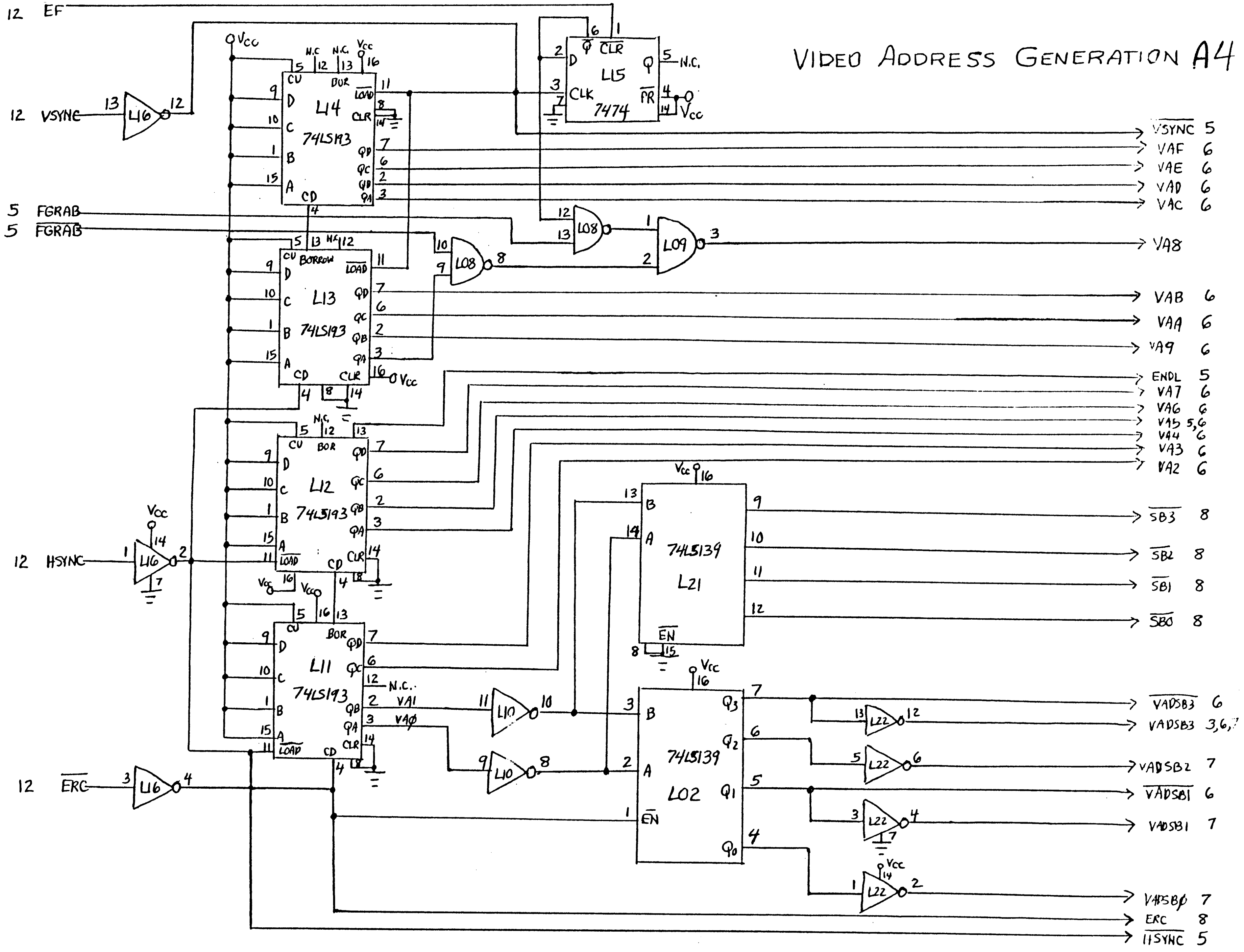


MEMORY STROBES & TIMING

A3

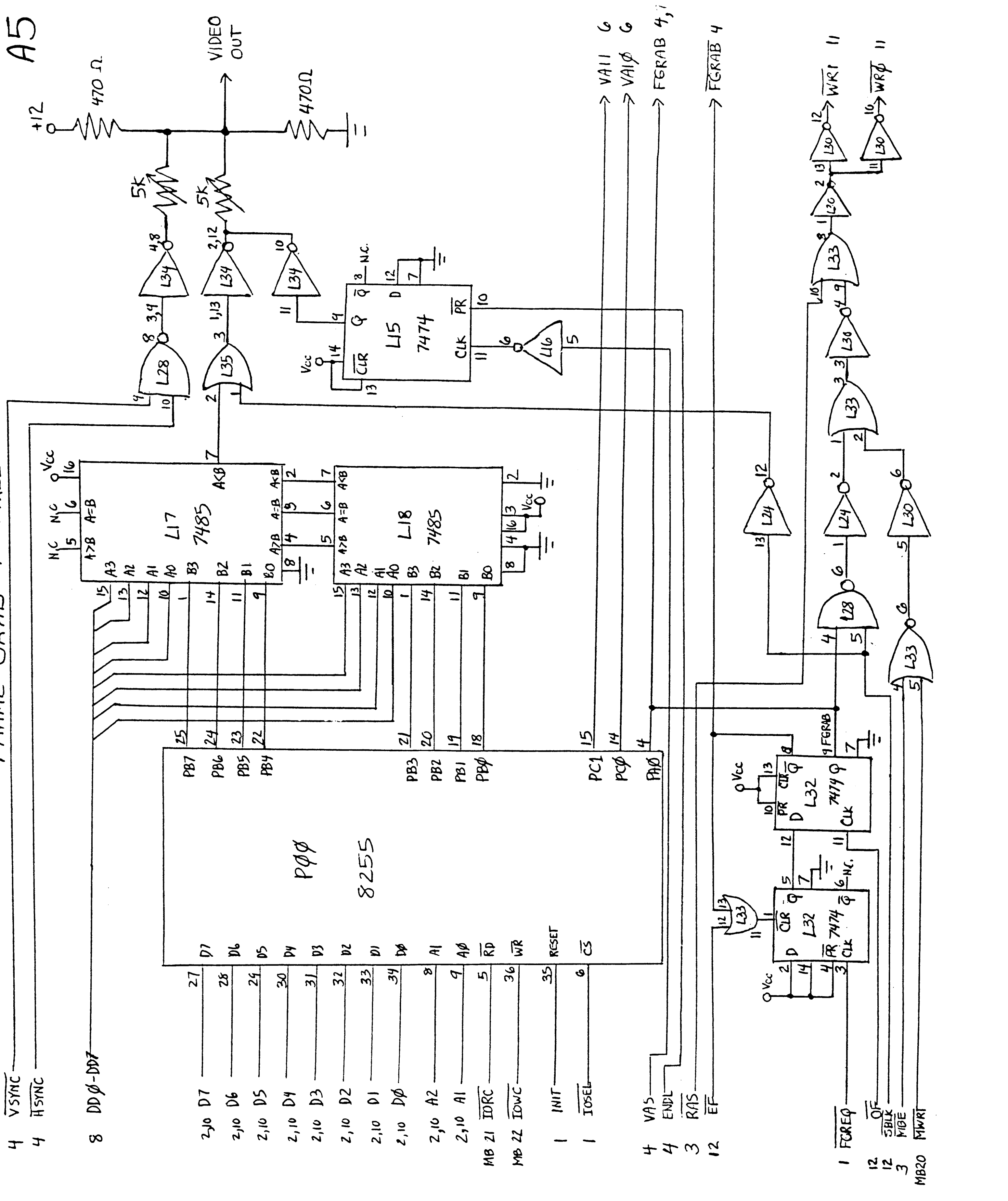


VIDEO ADDRESS GENERATION A4



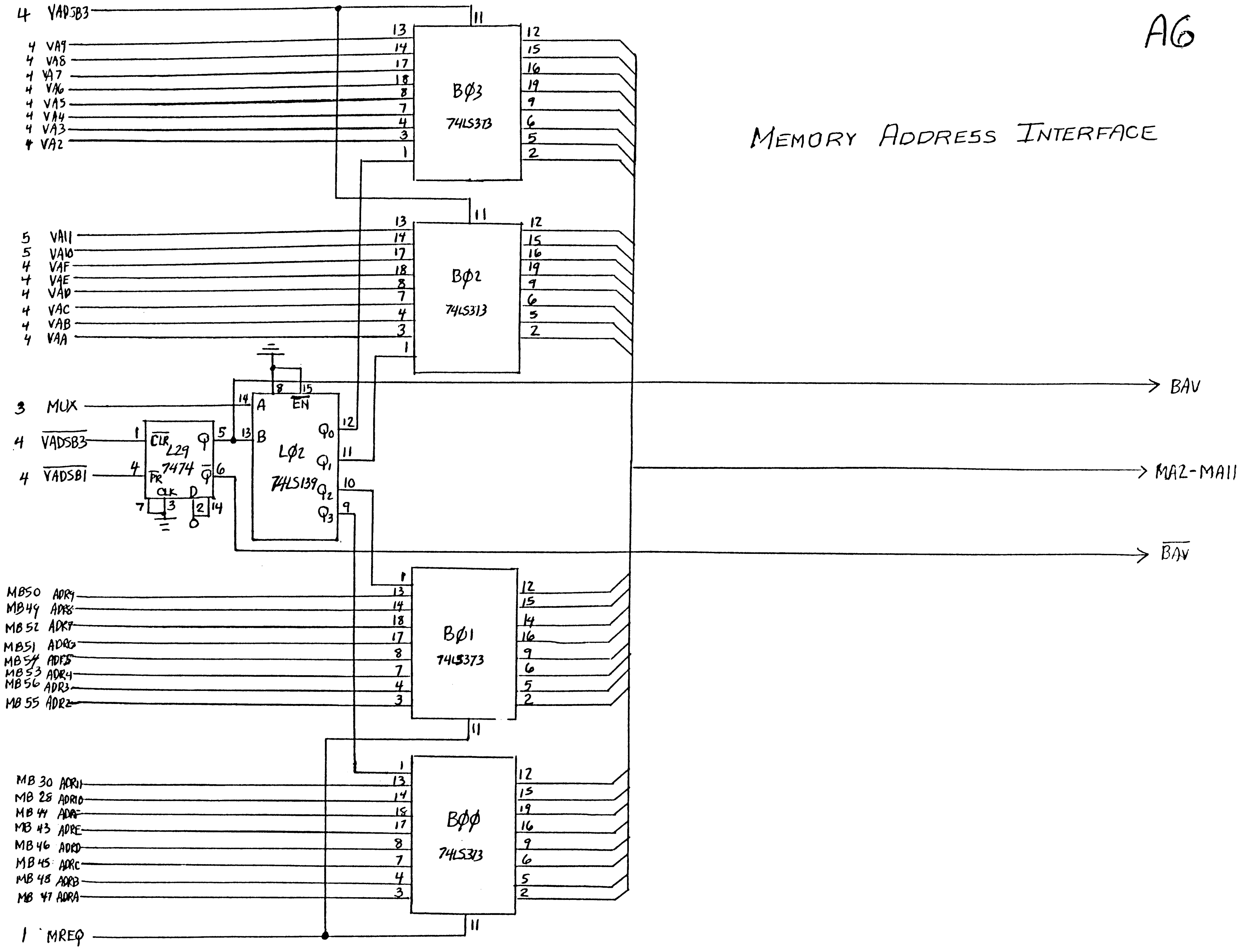
FRAME GRAB & PIXEL THRESHOLDING

A5



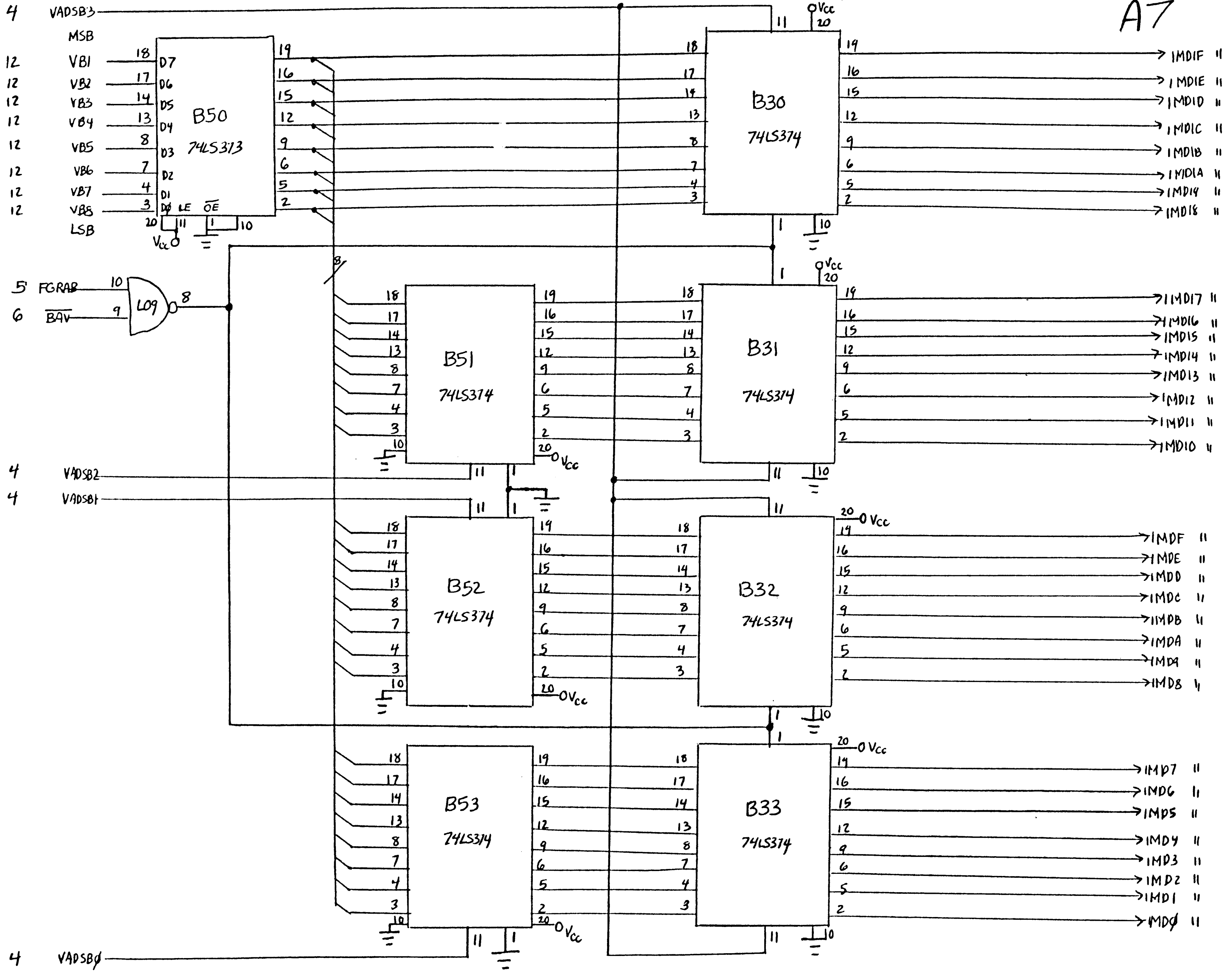
A6

MEMORY ADDRESS INTERFACE



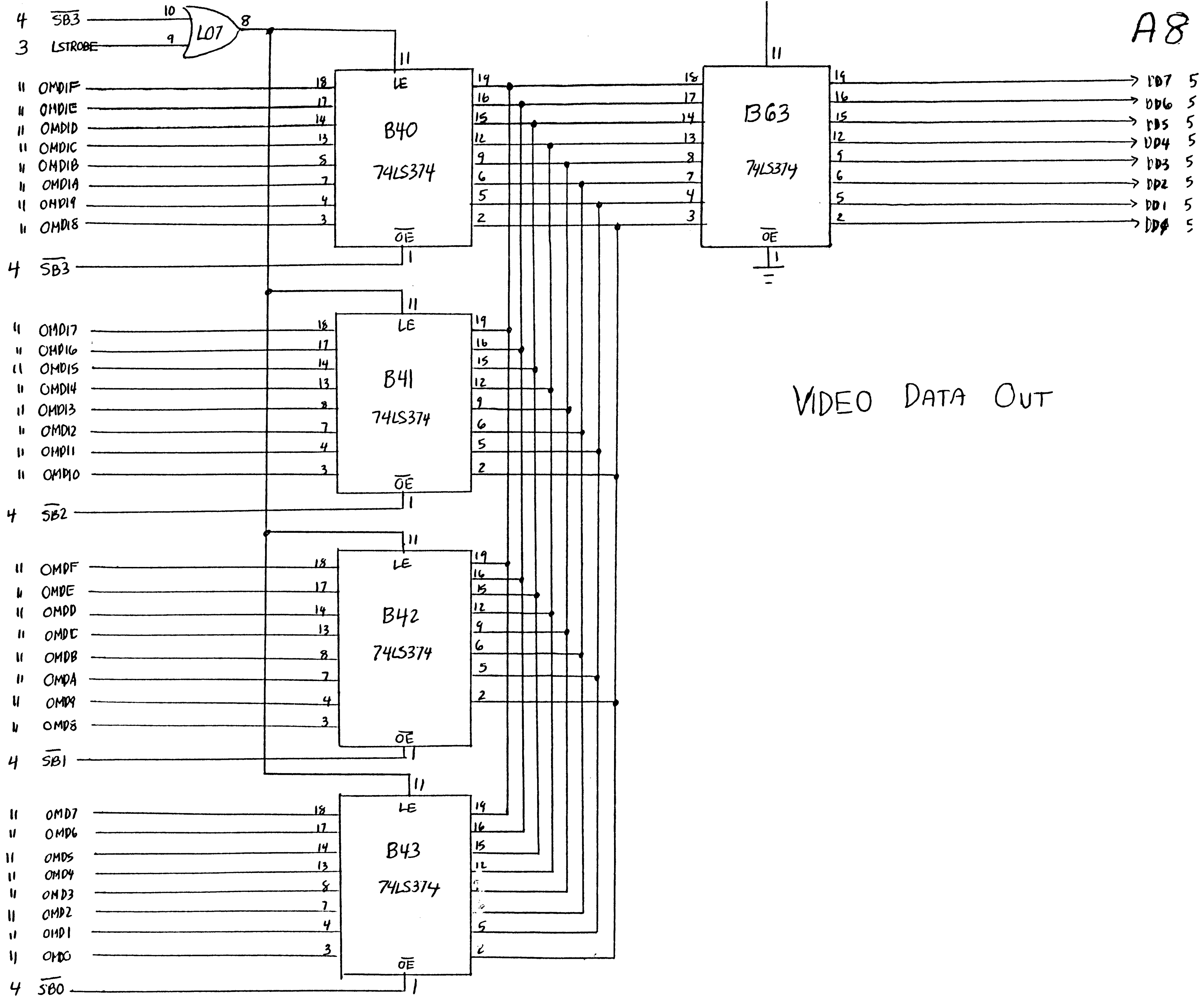
VIDEO DATA IN

A7



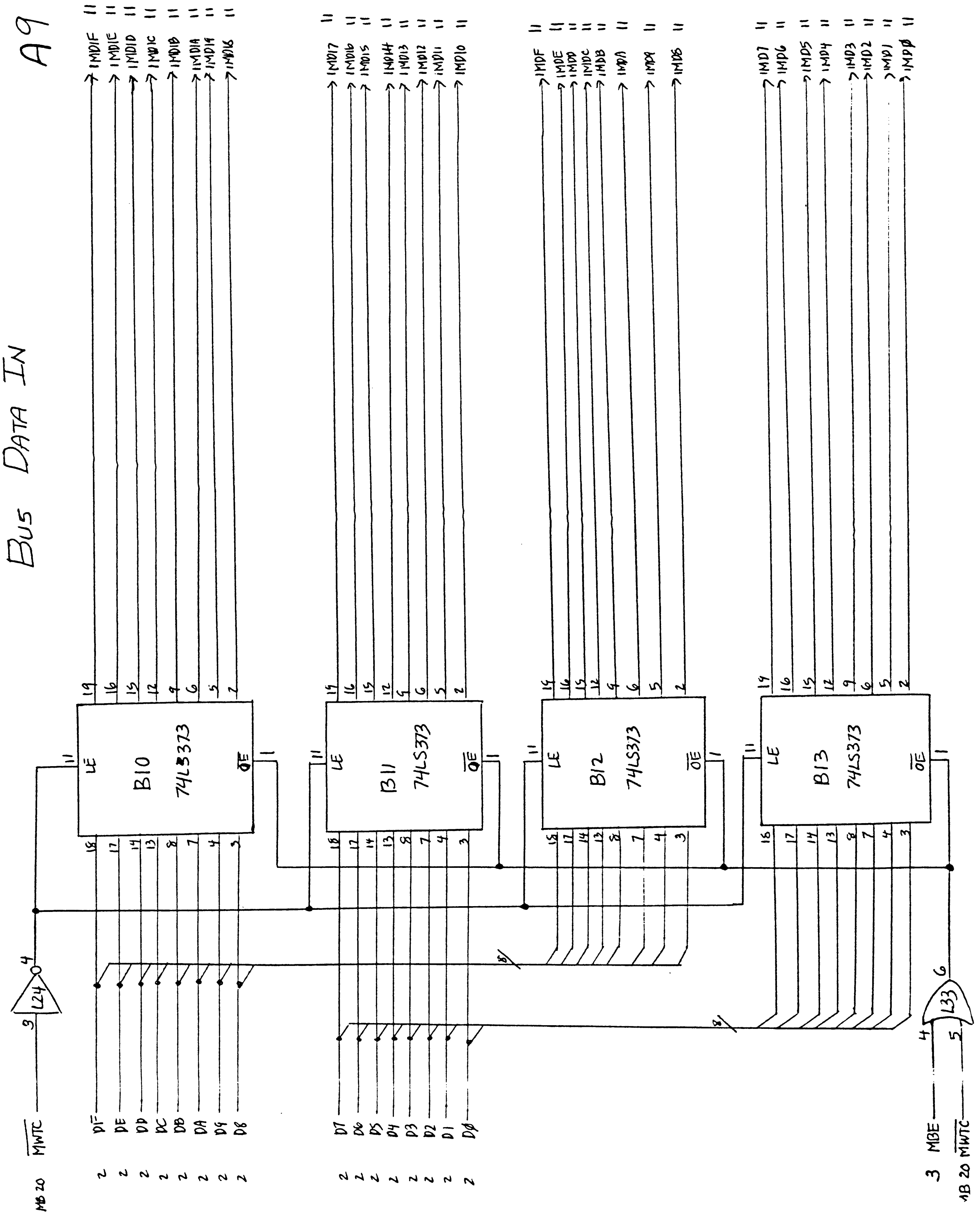
A8

VIDEO DATA OUT



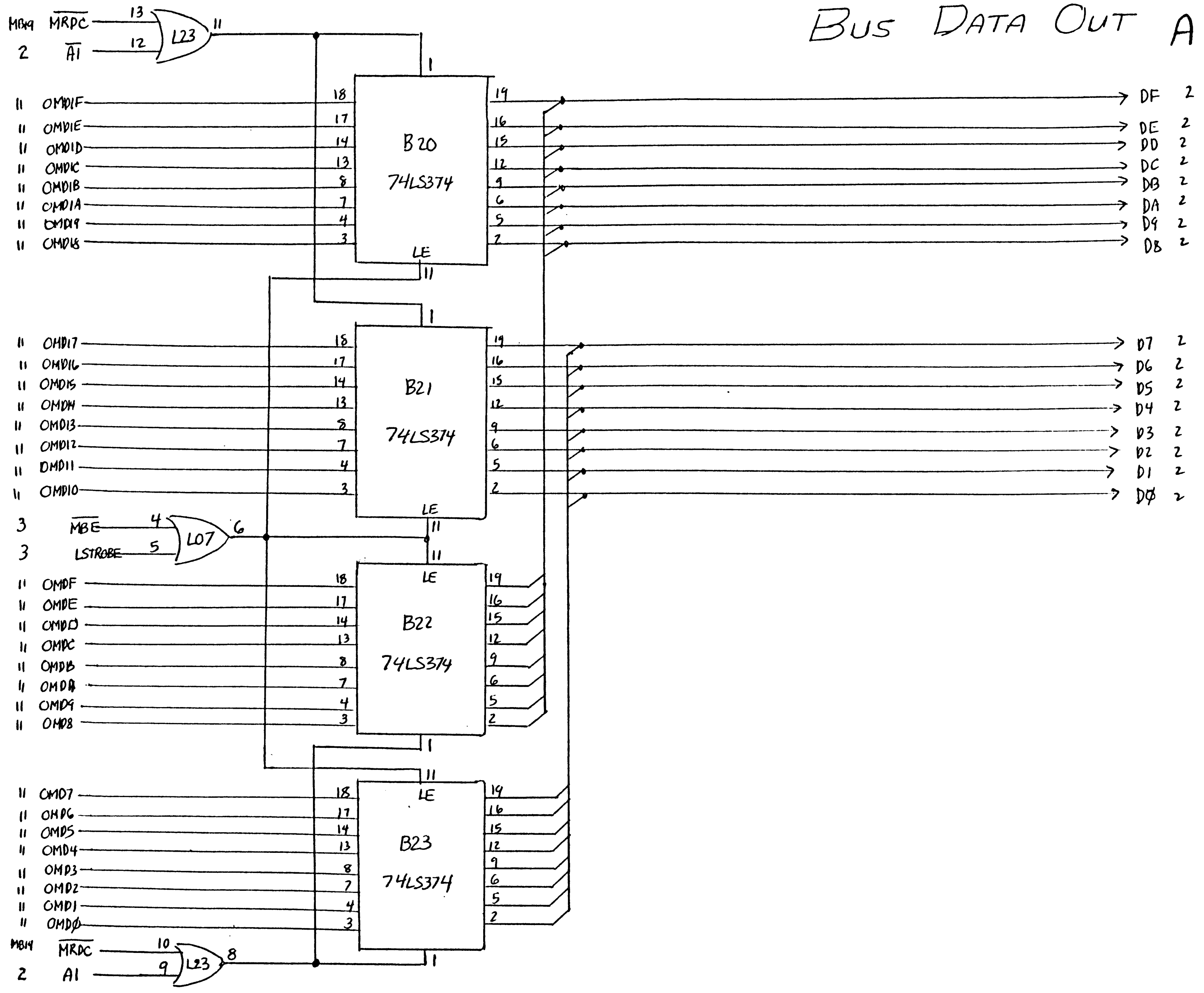
Bus Data In

A9



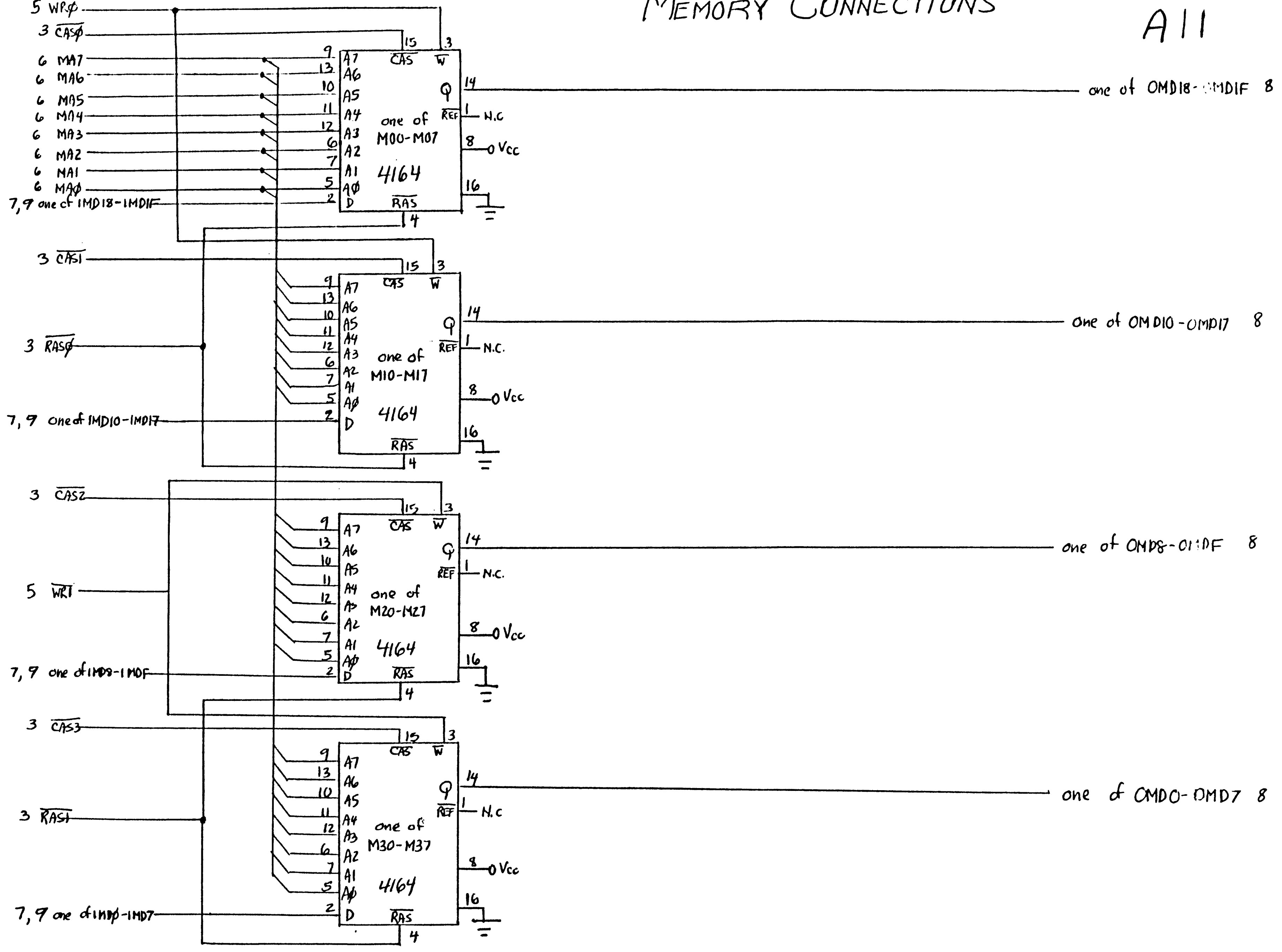
B10 74LS373
B11 74LS373
B12 74LS373
B13 74LS373
LE
OE
L24
L33
MB20 MWTC
3 MBE
MB 20 MWTC
DF DE DD DC DB DA D9 D8
D7 D6 D5 D4 D3 D2 D1 DØ
IMD1F IMD1E IMD1D IMD1C IMD1B IMD1A IMD19 IMD18
IMD17 IMD16 IMD15 IMD14 IMD13 IMD12 IMD11 IMD10
IMDF IMDE IMDD IMDC IMDB IMDA IMD9 IMD8
IMD7 IMD6 IMD5 IMD4 IMD3 IMD2 IMD1 IMDØ
8

Bus Data Out A1C



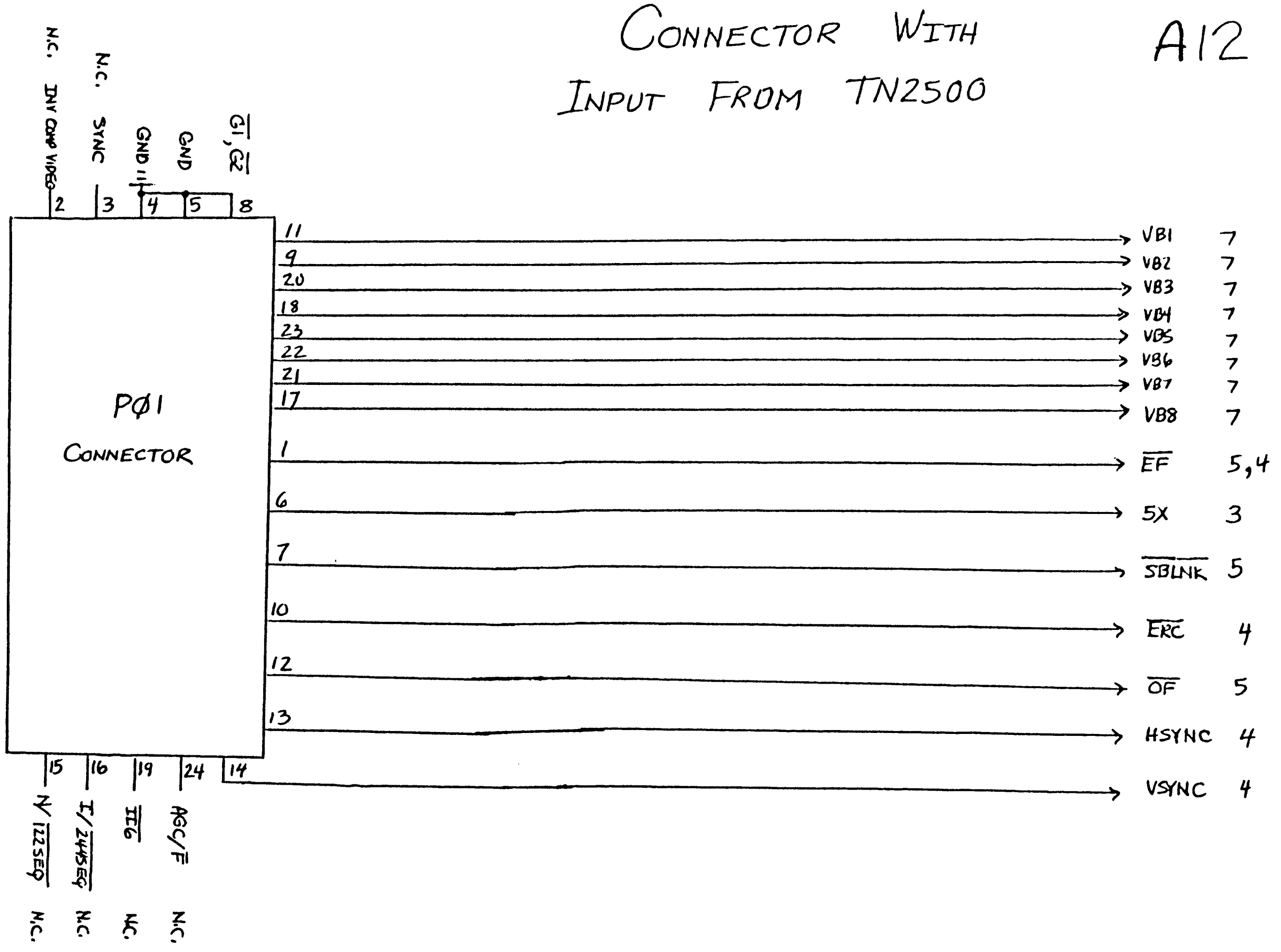
Memory Connections

A11



Connector With Input From TN2500

A12



B1

```
SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE ENCODE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM86.86 :F2:ENCODE.PLM NOOBJECT


                /**************************************************************/
                /***                                                        ***/
                /***     This program performs a run-length encoding on     ***/
                /***     a frame stored in the frame_grabber's memory.      ***/
                /***     It is written to be executed on an Intel 86/12A    ***/
                /***     single board computer, using the Intel iSBC 957B   ***/
                /***     monitor and an Intel Series III as the console.    ***/
                /***                                                        ***/
                /***                                                        ***/
                /**************************************************************/



                encode : do;

       1          declare (transit,white,threshold)         byte;
       1          declare (i,j,indx,indx2)                  word;
       1          declare true                              literally '1';
       1          declare false                             literally '0';

                 /** encode contains the encoded image **/
       1          declare encode(1024)                      byte;

                 /** frame_grabber memory starts at location 40000H **/
       1          declare buffer(0ffffh)                    byte at (40000h);



                /*** external procedures provided by the 957B monitor ***/

       1        ci : procedure byte external;
       2         end ci;

       1        co : procedure(char) external;
       2          declare char byte;
       2        end co;

       1        exit : procedure external;
       2         end exit;


                 /*** ascii$co is used to print a byte value on the console.   ***/
                 /*** (value is displayed in hex)                               ***/

       1         ascii$co : procedure(char) public;
       2          declare (temp,char) byte;

       2            temp=shr(char,4);
       2            if temp>=10 then temp=40h or (temp-9);
       2                        else temp=30h or temp;
       2            call co(temp);
       2            temp=0fh and char;
       2            if temp>=10 then temp=40h or (temp-9);
       2                        else temp=30h or temp;
       2            call co(temp);
       2          end ascii$co;
```

```
               /***                                ***/
               /*** get threshold level from console ***/
               /***                                ***/

                /** print prompt **/

                call co(3fh);

               /** input hex value **/

                indx=ci; call co(indx);
                if indx>39h then indx=indx+9;
                threshold=shl((0fh and indx),4);
                indx=ci; call co(indx);
                if indx>39h then indx=indx+9;
                threshold=threshold or (0fh and indx);

             /** display value to validate input **/

                call asciico(threshold);
                call co(0dh); call co(0ah);



             /*** instruct the frame-grabber to get a frame ***/
                output(0c300h)=0;


             /*** set-up frame_grabber's monitor output ***/
                output(0c206h)=99h;
                output(0c202h)=threshold;

             /*** wait for complete grab ***/
           /*     do while not(input(0c200h));
                end;
           */   call time(4000);



                indx2=0;

               /** encode rows 5 to 230 and columns 28 to 250 to avoid camera   **/
               /** noise at the frame edges                                      **/

                do i=5 to 240;
                   indx=i*256;
                   encode(indx2)=i;
                   white=true;
                   transit=false;

                   do j=28 to 250;

                      if ((buffer(indx+j)<threshold) and white) or


                         ((buffer(indx+j)>threshold) and not(white))
                         then do;
                            indx2=indx2+1;
                            encode(indx2)=j;
                            white=not(white);
                            transit=true;
                          end;
                   end;


                   if not(white)
                      then do;
                        indx2=indx2+1;
                        encode(indx2)=231;
                       end;


                   if transit
                      then do;
                        indx2=indx2+1;
                        encode(indx2)=0ffh;
                        indx2=indx2+1;
                      end;
                end;


               /** flag end of encoded frame **/
                encode(indx2)=0ffh;



               /** display encoding on the console **/

               j=0;
               do i=0 to indx2;
                call asciico(encode(i)); call co(20h);
                j=j+1;
                if j=16 then do;
                             call co(0dh);
                             call co(0ah);
                             j=0;
                          end;
                end;


             call exit;

           end;
```