Professional Program Session Record 21

New Developments in Programmable Logic

21

Wescon/82

Electronics Show & Convention/September 14-16
Convention Center
Marriott Hotel
Hilton Inn at the Park
Anaheim, California

A UNIVERSAL LANGUAGE FOR PROGRAMMABLE LOGIC

Robert Osann
Assisted Technology, Inc.
2381 Zanker Road, Suite 150
San Jose, CA 95131

INTRODUCTION

During this year we have seen Programmable Logic Devices (PLDs) enter a very significant growth phase. A number of major semiconductor companies have entered this rapidly expanding marketplace in order to second-source existing products and provide proprietary designs. Competition has greatly increased causing a corresponding decrease in prices. This era bears many similarities to that of the microprocessor around 1975. It took years, however, before the need for high level languages (by definition, universal) and universal microprocessor development systems became painfully apparent. These lessons must be applied if PLDs are to fulfill their potential and emerge as the most significant advance in Systems Design of the early 80's.

In recent years, different families of PLD's (PALs, FPLAs, etc.) have been supported by different languages. A universal language for PLDs eliminates confusion, while increasing design flexibility and providing an upward growth path to even higher level design languages. These higher level tools will support entire sub-systems where PLDs from different families are intermixed to provide the best overall design solution. Such a growth path will be necessary if the industry is to ever reach the goal of an integrated and efficient computer-aided engineering environment. The CUPL language (pronounced "couple") is a hardware compiler specifically designed for programmable logic. As the first universal language of its kind it will provide the most-important first step toward the higher-level design tools of the future.

MAJOR CONSIDERATIONS

Device Independence

Device independence is the most important aspect of a "universal" language and implies that a PLD source specification should not be tied to a particular PLD type or family. In practice, however, some dependent consideration must be given to pin assignment at this point in time. This is due to some devices having more product terms associated with some pins than with others. The actual specification, however, bears no mention of device type as this is only required upon invoking the compiler to create a fuse pattern for a particular device type.

Device independence is also enhanced by a pin declaration format which allows a pin to be defined as active high with the corresponding equation written in positive logic, even though the target device may have an inverting output structure. This is accomplished by the compiler's ability to perform deMorgan's Theorem and subsequently minimize the result by removing duplicate, zero, and contained product terms. Such an operation does, however, tend to use a lot of product terms. If a product term limitation arises, an appropriate error message is given.

Flexibility/Ease of Use

In order to accomplish universal support, a language must first gain acceptance among the engineering community. This is best accomplished by incorporating features which make the hardware engineer more productive while making his or her task less difficult. A variety of shorthand notations and macro-type features have therefore been incorporated into the language. These features reduce the amount of source code which must be written, in turn reducing the possibilities of errors. The language is also relatively free-form and relies no more than necessary on position dependence. Another potential restriction has been removed by allowing names up to 31 significant characters in length.

Shorthand notations are provided for use in representing groups of variables, the most useful being that for a list such as:

```
                    [a19..12]
```

which is used in place of:

```
     a19,a18,a17,a16,a15,a14,a13,a12
```

The above list may also be represented by a single variable name when the field function is utilized as in:

```
       field addr = [a19..12];
```

Subsequently, "addr" can be used to represent the list of address bit names.

Since one of the most common applications for PLDs at this time is address decoding, a simple range function has been provided which will work for even binary boundaries as long as the target PLD contains a sufficient number of product terms. If the range of E4ØØØ thru E7FFF hexadecimal were to be decoded from the field "addr" shown previously, the function would be written:

```
  addr:[E4ØØØ..E7FFF]
```

where the boundaries are E4ØØØ (lower) and E7FFF (upper), inclusively.

A variation on, and subset of, the range function is the equality function which is expressed in general as:

```
variable_field:constant_field
```

This function is especially usefull in specifications for state machines as in:

```
state_bitØ=(ST:Ø # ST:2 # ST:4 # ST:6)&
           advance & !reset
```

where ST had been previously defined as follows:

```
field ST=[state_bitØ..2]
```

For both the range and equality functions, constants are understood to be in hexadecimal unless it is otherwise stated that they are in octal or binary.

Probably contributing the most to ease of use are the following properties of the language: parenthetical operations and expression substitution. The distributive property A&B#A&C=A&(B#C) greatly reduces the number of names which must be typed when entering a PLD source specification if there are common subsets among groups of product terms. Expression substitution also reduces the size of equations while enhancing readability. This is accomplished by allowing a symbolic name to be used in one expression, that same symbolic name being elsewhere defined as equal to a different expression. As an example:

```
chip_select = inrange & strobe & !inh
where,
inrange = !a15 & !a13 & a12
and,
strobe = memr # memw
```

Another feature which makes the language easier to use concerns the handling of outputs with programmable tri-state enables. If an output is to be always enabled, as is most often the case, no declaration is written. Otherwise, the enable function is used and takes the form of:

```
          variable.oe = expression
                      or
          [list].oe = expression;
```

Using this function, a PLD whose output is connected to a data bus might use the following enable format:

```
[D7,D6,D5,D4,D3,D2,D1,DØ].oe = out_en;
```

or, better yet:

```
[D7..Ø].oe = out_en;
```

where out_en is elsewhere defined using the expression substitution feature.

Another important feature of the CUPL language is the manner in which flip-flop related functions are handled. This is somewhat similar to the output enable function in that an extension is added to the variable name associated with the "Q" output of the flip-flop. The following is a list of the various extensions:

Extension	Function
.d	d-type, "d"input
.j	jk-type,"j"input
.k	jk-type,"k"input
.s	set input
.r	reset input
.p	preset input

An example of the equation for the "Q" output of a D-type flip-lop would be as follows:

```
out.d = inl & (in2 # in3)
```

These and other features of the language will become more apparent in the application examples to be presented later.

Documentation

Good documentation is always important when a product enters production, but in today's fast moving engineering community with its high employee turnover rate, documentation is more important than ever. The CUPL language includes a free-form provision for comments, borrowed from high level programming languages, which takes the form:

```
/* Comment */
```

and may be placed anywhere within the PLD source specification. Although this feature is important, engineers are not well known for their willingness to document their work. It's therefore important for a PLD language to be self-documenting wherever possible. Most of the capabilities mentioned thus far, in particular expression substitution and the range and equality functions, enhance the self-documenting aspect of the language.

Finally, the engineering environment of the future will rely on a common, integrated data base. Having a different language for each family of PLDs would make it difficult to reach this goal. A better choice is CUPL, a universal language.

Expandability

As the technology of digital electronics never stands still, neither should the tools of the designer who uses that technology. Eventually, support tools for PLDs will grow to the point where multiple PLDs will be programmed as the result of a single source specification. Establishing the data structures which properly support the device characteristics and design rules for all families of PLDs provides an easier growth path toward such higher level capabilities.

Key words and operators must also be considered when planning for the future. Words such as "if", "else", and "goto" are reserved for higher level languages supporting sequential machines. Operators such as "+", "*", and "/" will be eventually supported in higher level languages in their true arithmetic sense. Other operators are therefore required to support the basic logical operations. The language utilizes a set of operators which borrow mostly from high-level software programming languages. A portion of the list is shown below:

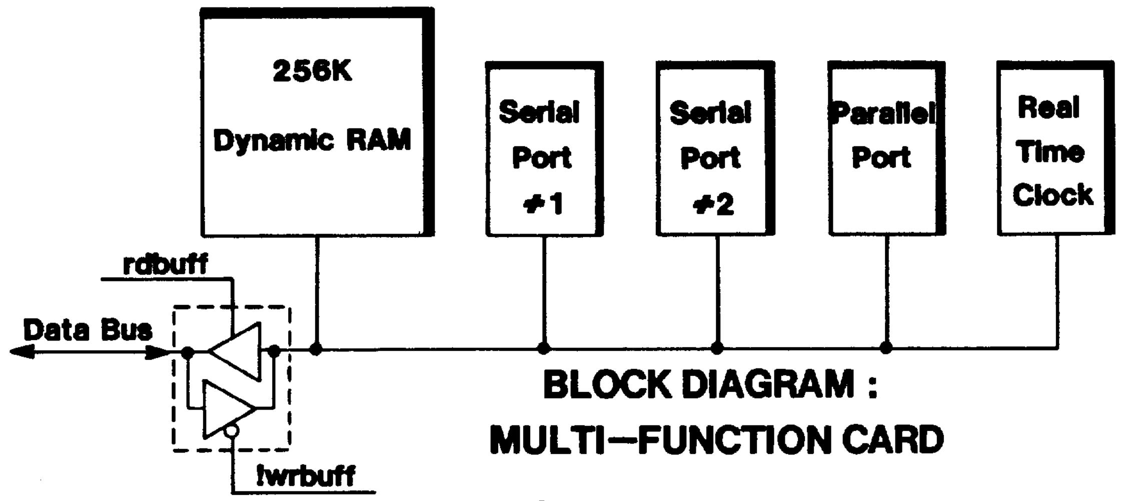
And	&	Add	+
OR	#	Subtract	-
Exclusive-OR	XOR	Multiply	*
Assignment	=	Divide	/
NOT	!		

The CUPL pre-processor will also allow a personal choice of operators without compromising the effectiveness of having a standardized set.

Testing Considerations

As larger quantities of PLDs are used in specific designs, testing becomes an issue of ever increasing importance. Simu ati apability will also be included to allow the designer to further test the accuracy of a source specification before a device is actually programmed. Later versions of this universal language will support testing of non-registered parts by automatically generating test vectors using variations on standard semiconductor test patterns. Registered PLDs will initially be handled by allowing the designer to create a function table of states and transitions which the compiler will convert into test vector format.

In the future, higher level versions of the language will provide a state-machine modelling capability allowing the compiler to automatically generate test vectors for registered PLDs.



BLOCK DIAGRAM :
MULTI—FUNCTION CARD

FIGURE 1

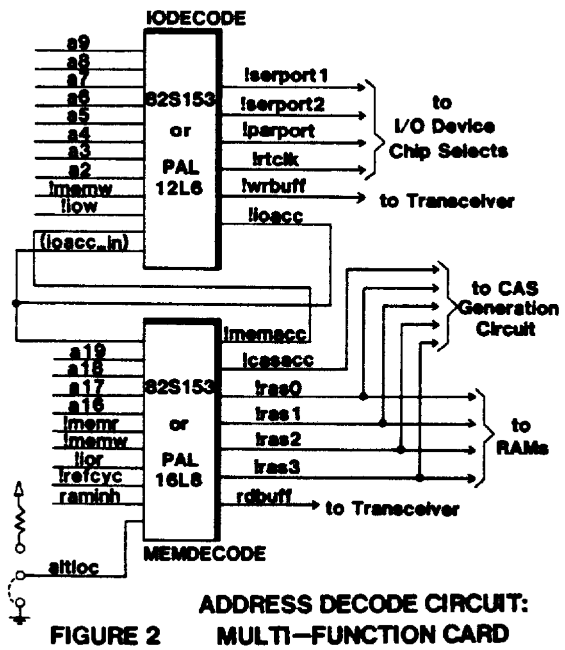


FIGURE 2 ADDRESS DECODE CIRCUIT: MULTI—FUNCTION CARD

CONCLUSION

The CUPL language, just described, does more than provide the engineer with a unified approach to PLD-based designs upon which future products may build. It also includes numerous features which make logic design easier and less error prone with results that are more understandable and therefore easier to maintain.

AN ADDRESS DECODING EXAMPLE

As this type of application is probably the most common for PLDs at this time, a reasonable choice for an example is the sub-system shown in Figure 1. This block diagram depicts a multi-function board similar to that found in many bus-oriented microcomputer systems. The 256K memory consists of four banks of 64K dynamic RAMS, the entire grouping being mappable to either the first or second 256K block of addressable space according to a jumper called "altloc". This memory resides on the same 8-bit bus as four I/O devices.

The two PLDs shown in Figure 2 provide the RAS control signals for the RAMs, the chip select signals for the I/O devices and the enable signals for both paths through the data bus transceiver. These devices could either be two 82S153s or a PAL16L8 and a PAL12L6 with no changes required of the source specifications.

```
                                    partno      p11000153 ;
                                    name        mdecode ;
                                    rev         01 ;
                                    date        5/12/82 ;
                                    designer    Osann/Kahl ;
                                    company     Assisted Technology ;

/*******************************************************/
/*  This device generates the memory RAS signals and   */
/*  initiates the generation of CAS. It also enables   */
/*  the data bus transceiver for both the memory and   */
/*  I/O read cycles.                                   */
/*******************************************************/

pin 1          = !ioacc ;
pin [2..5]     = [a19..16] ;
pin 6          = altloc ; /* map RAM to 40000 thru 7FFFF */
pin 7          = !refcyc ; /* memory refresch cycle */
pin [8,9]      = ![memw,memr] ;
pin 11         = !ior ;
pin 14         = raminh ; /* system RAM inhibit signal */
pin 13         = !memacc ; /* on-board memory being accessed */
pin 12         = !casacc ; /* output to CAS circuitry */
pin [15..18]   = ![ras0..3] ; /*RAM RAS signals */
pin 19         = rdbuff ; /* transceiver enable for reads */

field memaddr = [a19..16] ;

memreq        = memw # memr ;
memacc_eqn    = !raminh & !refcyc & (memaddr:[00000..3FFFF] &
                !altloc # memaddr:[40000..4FFFF] & altloc)    ;

casacc = memreq & memacc_eqn ;
rdbuff = memacc & memr # ioacc & ior ; /* note memacc feeds-back
                                          internally */
memacc = memacc_eqn ;

ras3 = !raminh & memreq & !refcyc & (memaddr:[30000..3FFFF] &
       !altloc # memaddr:[70000..7FFFF] & altloc) # refcyc ;

ras2 = !raminh & memreq & !refcyc & (memaddr:[20000..2FFFF] &
       !altloc # memaddr:[60000..6FFFF] & altloc) # refcyc ;

ras1 = !raminh & memreq & !refcyc & (memaddr:[10000..1FFFF] &
       !altloc # memaddr:[50000..5FFFF] & altloc) # refcyc ;

ras0 = !raminh & memreq & !refcyc & (memaddr:[00000..0FFFF] &
       !altloc # memaddr:[40000..4FFFF] & altloc) # refcyc ;
```

Figure 3

```
                    Partno   PL00000154 ;
                    Name     IODECODE ;
                    Rev      01 ;
                    Date     5/12/82 ;
                    Designer Osann/Kahl ;
                    Company  Assisted technology ;

/***************************************************************/
/* This device generates the chip select signals for the I/O */
/* functions. It also enables the data bus transceiver for   */
/* both memory and I/O write cycles.                          */
/***************************************************************/

pin [1..8] = [a9..2] ;
pin 9      = !memw ;
pin 11     = !iow ;
pin 12     = !ioacc_in ; /* same signal as ioacc */
pin 19     = !memacc ; /* on-board memory being accessed */
pin 17     = !serport1 ; /* serial port #1 chip select */
pin 16     = !serport2 ; /* serial port #2 chip select */
pin 15     = !rtclk ; /* real-time clock chip select */
pin 14     = !parport ; /* parallel port chip select */
pin 18     = !wrbuff ; /*xceiver enable for write cycles */
pin 13     = !ioacc ; /* on-board I/O being accessed */

field   ioaddr    = [a9..2] ;

serport1_eqn      = ioaddr:[2F8..2FF] ;  /********************/
serport2_eqn      = ioaddr:[3F8..3FF] ;  /*   I/O Address    */
rtclk_eqn         = ioaddr:[100..11F] ;  /*     Ranges       */
parport_eqn       = ioaddr:[1F4..1F7] ;  /********************/

wrbuff            = memacc & memw # ioacc_in & iow ;
ioacc             = serport1_eqn # serport2_eqn # rtclk_eqn #
                    parport_eqn ;
serport1          = serport1_eqn ;
serport2          = serport2_eqn ;
rtclk             = rtclk_eqn ;
parport           = parport_eqn ;
```

Figure 4

Figure 3 is the source specification for the PLD named "MEMDECODE" which creates the four RAS signals according to the CPU's address range and the "altloc" jumper. This device also generates "casacc" which signals the CAS circuitry that the appropriate CAS signals should be generated. The "memacc" signal includes no control strobes and is used by both devices for generating the buffer enables. It should be noted the "rdbuff", the transceiver enable for reads, is defined as "true" even though the pin list also defines it as true. If a PAL16L8 were used, the compiler would generate the equivalent of:

```
!rdbuff = !memacc & !ioacc #
          !memacc & !ior #
          !memr   & !ioacc #
          !memr   & !ior;
```

Also note that the field function has been used to define "memaddr" which is subsequently operated on by the range function.

Expression substitution is used in that "memreq" is created and then used in many of the equations. In most cases this reduces the equation size by approximately a factor of two. The expression for "memacc" is initially defined in a similar way as the intermediate variable "memacc_eqn".

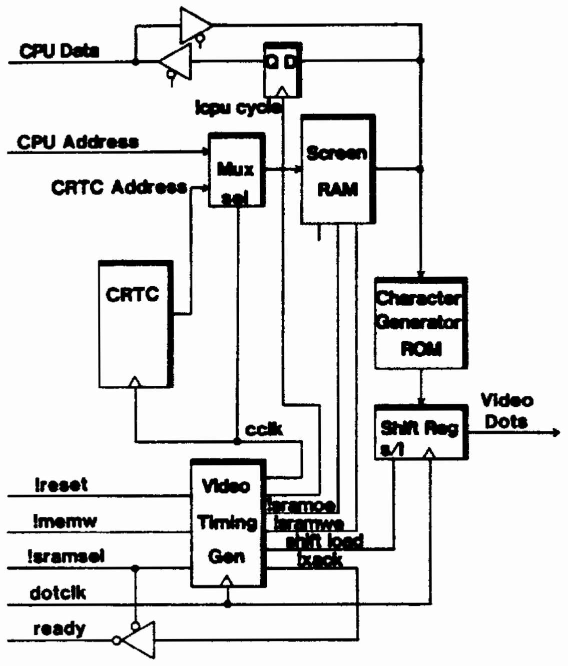


FIGURE 5— VIDEO SUBSYSTEM

This expression comprises almost all of the equation for "casacc" and is therefore used in the "casacc" expression. The variable "memacc_eqn" is then equated to "memacc", the output pin, which in turn is fed back internally to be combined with the "memr" signal in the expression for "rdbuff".

Figure 4 is the source specification for the PLD "IODECODE" which generates chip selects for the I/O functions as well as the transceiver enable for CPU write cycles. Note that the equations defining the I/O chip selects are first defined as intermediate variables (as in "serportl_eqn = "). This allows "ioacc" to be defined in terms of these variables thereby reducing the size of that expression while also making more apparent the intent of the designer. In fact, both Figures 3 and 4 utilize no comments, except in the pin list, in order to emphasize the self-documenting nature of the language.

A SEQUENTIAL MACHINE EXAMPLE

A common application for a simple state machine exists in CRT display circuits where the video timing generator also arbitrates access to the screen RAM between the CPU and the CRT controller chip. The circuit of Figure 5 represents such an application. In this example the screen RAM is fast enough that two accesses may be made during one complete cycle of the character clock ("cclk"). This is more obvious from the timing diagram of Figure 6. CRT controller read cycles from the screen RAM are always made during the period where "cclk" in Figure 6 is low. CPU accesses are made only while "cclk" is high.

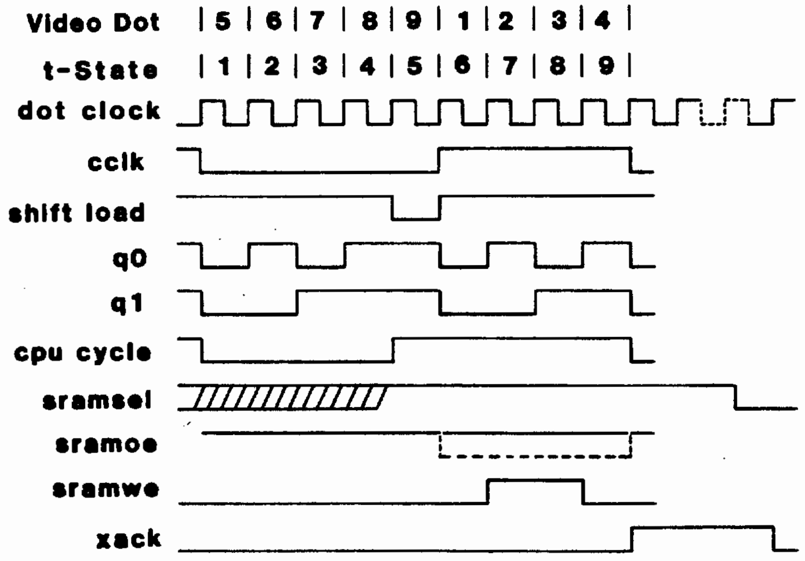

Video Dot | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 |
t-State | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
dot clock
cclk
shift load
q0
q1
cpu cycle
sramsel
sramoe
sramwe
xack

FIGURE 6— VIDEO GENERATOR TIMING DIAGRAM

```
                              Partno       PL0000257;
                              Name         VIDTIM;
                              Date         4/20/82;
                              Revision     03;
                              Designer     R. Osann;
                              Company      Assisted Technology, Inc.;

/*********************************************************************/
/*  This device is clocked by the video dot clock and generates the */
/*  character clock and screen Shift/Load signals as well as arbi-  */
/*  tration between the CPU and CRTC for the screen RAM.            */
/*********************************************************************/

pin 2           = !reset ; /* system reset signal*/
pin 4           = !sramsel ; /* CPU access to screen RAM */
pin 5           = !memw ;
pin [18,19]     = ![q1,q0] ; /* state variable bits */
pin 17          = !cpu_cycle ; /* CPU cycle where VIDTIM
                                  performs arbitration */
pin 16          = shift_load ; /* Shift/Load signal to video S/R */
pin 15          = cclk ; /* CRTC character clock */
pin 14          = !sramoe ; /* screen RAM output enable */
pin 13          = !sramwe ; /* screen RAM write enable */
pin 12          = !xack ; /* transfer acknowledge signal, used for
                             driving system ready signal active */

t1               = !q0 & !q1 & !cclk &  shift_load ;
t2               =  q0 & !q1 & !cclk &  shift_load ;
t3               = !q0 &  q1 & !cclk &  shift_load ;
t4               =  q0 &  q1 & !cclk &  shift_load ;
t5               =  q0 &  q1 & !cclk & !shift_load ;
t6               = !q0 & !q1 &  cclk &  shift_load ;
t7               =  q0 & !q1 &  cclk &  shift_load ;
t8               = !q0 &  q1 &  cclk &  shift_load ;
t9               =  q0 &  q1 &  cclk &  shift_load ;

q0.d             = !reset & (t1 # t3 # t4 # t6 # t8) ;
q1.d             = !reset & (t2 # t3 # t4 # t7 # t8) ;
shift_load.d     = !(!reset & t4) ;
cclk.d           = !(!reset & (t9 # t1 # t2 # t3 # t4)) ;
cpu_cycle.d      = !reset & (t4 & sramsel & !xack # cpu_cycle &
                   (t5 # t6 # t7 # t8)) ;

sramoe.d         = t1 # t2 # t3 # t4 # t9 # cpu_cycle & !memw ;
sramwe.d         = cpu_cycle & memw & (t6 # t7) ;
xack.d           = cpu_cycle & t9 # xack & sramsel ;
```

Figure 7

The basic operation of the circuit is based around a character block that would be nine dots wide when displayed on the CRT screen. The character font normally displayed in such a block would be seven dots wide leaving two dots between characters. The timing diagram of Figure 6 shows "cclk" divided into nine states labeled "t1" thru "t9". The "shift_load" signal occurs during state t5 and subsequently causes the shift register of Figure 5 reload during t6. The nine states (t1..9) are defined as intermediate variables in the source specification of Figure 7. This greatly reduces the overall size of the specification while making it a simple task to write equations for the generated signals directly from the timing diagram. Registered PLDs such as the PAL16R8 and 82S159 may be used here. A "d" type flip-flop structure is chosen as both types of PLDs have this capability. Remembering that for a "q" output to be active in a given t-state the corresponding "d" input must be active during the previous t-state, the timing diagram of Figure 6 provides sufficient information to write the source specification of Figure 7.

Signals "qØ" and "q1" are state variable bits used in conjunction with "shift_load" and "cclk" to create nine unique states without using any more outputs than necessary. The "sramsel" signal indicates that the CPU is attempting to access the screen RAM and, if present at the end of t4, will cause "cpu_cycle" to become active during the second half of "cclk" as shown in Figure 6. Also, when "sramsel" is active, the buffer driving the CPU ready signal in Figure 5 is enabled allowing "xack", currently inactive, to place the CPU in a wait state. The CPU access then occurs during the time "cclk" is high with "xack" becoming active at the completion of "cpu_cycle" thereby removing the wait state. The "xack" signal then remains active until ("sramsel") becomes inactive. The output enable ("sramoe") and write enable ("sramwe") signals for the screen RAM occur as shown in Figure 6. Here a read is always performed by the CRT controller while "cclk" is low and either a read or write cycle is performed while "cclk" is high depending on the type of CPU access.

List File

```
Cupl 1.0   Assisted Technology, Inc.  Copyright (c) 1982

                                Partno   PL00000154 ;
                                Name     IODECODE ;
                                Rev      01 ;
                                Date     5/12/82 ;
                                Company  Assisted technology ;

        /***************************************************************/
        /* This device generates the chip select signals for the I/O */
        /* functions. It also enables the data bus transceiver for   */
        /* both memory and I/O write cycles.                         */
        /***************************************************************/

        pin [1..8] = [a9..2] ;
        pin 9      = !memw ;
        pin 11     = !iow ;
        pin 12     = !ioacc_in ; /* same signal as ioacc */
        pin 19     = !memacc ; /* on-board memory being accessed */
        pin 17     = !serport1 ; /* serial port #1 chip select */
        pin 16     = !serport2 ; /* serial port #2 chip select */
        pin 15     = !rtclk ; /* real-time clock chip select */
        pin 14     = !parport ; /* parallel port chip select */
        pin 18     = !wrbuff ; /*xceiver enable for write cycles */
        pin 13     = !ioacc ; /* on-board I/O being accessed */

        field   ioaddr     = [a9..2] ;

        serport1_eqn       = ioaddr:[2F8..2FF] ;   /********************/
        serport2_eqn       = ioaddr:[3F8..3FF] ;   /*    I/O Address   */
        rtclk_eqn          = ioaddr:[100..11F] ;   /*      Ranges      */
        parport_eqn        = ioaddr:[1F4..1F7] ;   /********************/

        wrbuff             = memacc & memw # ioacc_in & iow ;
        ioacc              = serport1_eqn # serport2_eqn # rtclk_eqn #
                             parport_eqn ;
        serport1           = serport1_eqn ;
        serport2           = serport2_eqn ;
        rtclk              = rtclk_eqn ;
        parport            = parport_eqn ;
```

Fuse Plot for PAL12L6

```
partno    PL00000154
name      IODECODE
rev       01
date      5/12/82
designer
company   Assisted technology

column                  11 1111 1111 2222 2222 2233
          0123 4567 8901 2345 6789 0123 4567 8901

row     8 ---- ---- --   --   --   --   ---x ---x ioacc_in&iow
row     9 ---- ---x --   --   --   --   ---- -x-- memacc&memw

row    16 -xx- x--- x-   x-   x-   x-   ---- ---- !a8&a9&a7&a6&a5&a4&a3

row    24 x-x- x--- x-   x-   x-   x-   ---- ---- a8&a9&a7&a6&a5&a4&a3

row    32 x--- -x-- -x   -x   --   --   ---- ---- a8&!a7&!a6&!a5

row    40 x--x x--- x-   x-   x-   -x   x--- ---- a8&!a9&a7&a6&a5&a4&!a3&a2

row    48 x--x x--- x-   x-   x-   -x   x--- ---- a8&!a9&a7&a6&a5&a4&!a3&a2
row    49 x--- -x-- -x   -x   --   --   ---- ---- a8&!a7&!a6&!a5
row    50 x-x- x--- x-   x-   x-   x-   ---- ---- a8&a9&a7&a6&a5&a4&a3
row    51 -xx- x--- x-   x-   x-   x-   ---- ---- !a8&a9&a7&a6&a5&a4&a3

pins              11                        11   11
          2211 3399 4400 5500 6600 7700 8822 9911
polarity  HLHL HLHL HLHH HLHH HLHH HLHH HLHL HLHL

pin:    1    a9
pin:    2    a8
pin:    3    a7
pin:    4    a6
pin:    5    a5
pin:    6    a4
pin:    7    a3
pin:    8    a2
pin:    9  ! memw
pin:   11  ! iow
pin:   12  ! ioacc_in
pin:   13  ! ioacc
pin:   14  ! parport
pin:   15  ! rtclk
pin:   16  ! serport2
pin:   17  ! serport1
pin:   18  ! wrbuff
pin:   19  ! memacc
```

H&L Plot for 82S153

```
*POLLLLLLLLLL
*P  00 *I  -------- *BI -------LL- *BO AA.....AAA
*P  01 *I  -------- *BI L--------L *BO AA.....AAA
*P  02 *I  -HHHHHLH *BI ---------- *BO A.A...AAAA
*P  03 *I  -HHHHHHH *BI ---------- *BO A..A..AAAA
*P  04 *I  ---LLLH- *BI ---------- *BO A...A.AAAA
*P  05 *I  HLHHHHHL *BI ---------- *BO A....AAAAA
*P  06 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  07 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  08 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  09 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  10 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  11 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  12 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  13 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  14 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  15 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  16 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  17 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  18 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  19 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  20 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  21 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  22 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  23 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  24 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  25 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  26 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  27 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  28 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  29 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  30 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  31 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  D9 *I  00000000 *BI 0000000000
*P  D8 *I  -------- *BI ----------
*P  D7 *I  -------- *BI ----------
*P  D6 *I  -------- *BI ----------
*P  D5 *I  -------- *BI ----------
*P  D4 *I  -------- *BI ----------
*P  D3 *I  -------- *BI ----------
*P  D2 *I  00000000 *BI 0000000000
*P  D1 *I  00000000 *BI 0000000000
*P  D0 *I  00000000 *BI 0000000000
```

List File

Cupl 1.0   Assisted Technology, Inc. Copyright (c) 1982

```
                         partno       p11000153 ;
                         name         mdecode ;
                         rev          01 ;
                         date         5/12/82 ;
                         designer     Osann/Kahl ;
                         company      Assisted Technology ;

/**********************************************************/
/*  This device generates the memory RAS signals and     */
/*  initiates the generation of CAS. It also enables     */
/*  the data bus transceiver for both the memory and     */
/*  I/O read cycles.                                      */
/**********************************************************/

pin 1           = !ioacc ;
pin [2..5]      = [al9..16] ;
pin 6           = altloc ; /* map RAM to 40000 thru 7FFFF */
pin 7           = !refcyc ; /* memory refresch cycle */
pin [8,9]       = ![memw,memr] ;
pin 11          = !ior ;
pin 14          = raminh ; /* system RAM inhibit signal */
pin 13          = !memacc ; /* on-board memory being accessed */
pin 12          = !casacc ; /* output to CAS circuitry */
pin [15..18]    = ![ras0..3] ; /*RAM RAS signals */
pin 19          = rdbuff ; /* transceiver enable for reads */

field memaddr = [al9..16] ;

memreq         = memw # memr ;
memacc_eqn     = !raminh & !refcyc & (memaddr:[00000..3FFFF] &
                 !altloc # memaddr:[40000..4FFFF] & altloc)    ;

casacc = memreq & memacc_eqn ;
rdbuff = memacc & memr # ioacc & ior ; /* note memacc feeds-back
                                          internally */
memacc = memacc_eqn ;

ras3 = !raminh & memreq & !refcyc & (memaddr:[30000..3FFFF] &
       !altloc # memaddr:[70000..7FFFF] & altloc) # refcyc ;

ras2 = !raminh & memreq & !refcyc & (memaddr:[20000..2FFFF] &
       !altloc # memaddr:[60000..6FFFF] & altloc) # refcyc ;

ras1 = !raminh & memreq & !refcyc & (memaddr:[10000..1FFFF] &
       !altloc # memaddr:[50000..5FFFF] & altloc) # refcyc ;

ras0 = !raminh & memreq & !refcyc & (memaddr:[00000..0FFFF] &
       !altloc # memaddr:[40000..4FFFF] & altloc) # refcyc ;
```

Fuse Plot for PAL 16L8

```
partno    p11000153
name      mdecode
rev       01
date      5/12/82
designer  Osann/Kahl
company   Assisted Technology

column                   11 1111 1111 2222 2222 2233
          0123 4567 8901 2345 6789 0123 4567 8901

row        0 ---- ---- ---- ---- ---- ---- ---- ----
row        1 --x- ---- ---- ---- ---- ---- --x- ---- !ioacc&!memacc
row        2 --x- ---- ---- ---- ---- ---- ---- x--- !ioacc&!memr
row        3 ---- ---- ---- ---- ---- ---- --x- --x- !memacc&!ior
row        4 ---- ---- ---- ---- ---- ---- ---- x-x- !memr&!ior

row        8 ---- ---- ---- ---- ---- ---- ---- ----
row        9 ---- ---- ---- ---- ---- -x-- ---- ---- refcyc
row       10 -x-- -x-- x--- x--- -x-- x--x -x-- ---- !al9&!al8&al7&al6&!altloc
                                                     &!refcyc&!raminh&memw
row       11 -x-- -x-- x--- x--- -x-- x--x ---- -x-- !al9&!al8&al7&al6&!altloc
                                                     &!refcyc&!raminh&memr
row       12 -x-- x--- x--- x--- x--- x--x -x-- ---- !al9&al8&al7&al6&altloc
                                                     &!refcyc&!raminh&memw
row       13 -x-- x--- x--- x--- x--- x--x ---- -x-- !al9&al8&al7&al6&altloc
                                                     &!refcyc&!raminh&memr

row       16 ---- ---- ---- ---- ---- ---- ---- ----
row       17 ---- ---- ---- ---- ---- -x-- ---- ---- refcyc
row       18 -x-- -x-- x--- -x-- -x-- x--x -x-- ---- !al9&!al8&al7&!al6&!altloc
                                                     &!refcyc&!raminh&memw
row       19 -x-- -x-- x--- -x-- -x-- x--x ---- -x-- !al9&!al8&al7&!al6&!altloc
                                                     &!refcyc&!raminh&memr
row       20 -x-- x--- x--- -x-- x--- x--x -x-- ---- !al9&al8&al7&!al6&altloc
                                                     &!refcyc&!raminh&memw
row       21 -x-- x--- x--- -x-- x--- x--x ---- -x-- !al9&al8&al7&!al6&altloc
                                                     &!refcyc&!raminh&memr

row       24 ---- ---- ---- ---- ---- ---- ---- ----
row       25 ---- ---- ---- ---- ---- -x-- ---- ---- refcyc
row       26 -x-- -x-- -x-- x--- -x-- x--x -x-- ---- !al9&!al8&!al7&al6&!altloc
                                                     &!refcyc&!raminh&memw
row       27 -x-- -x-- -x-- x--- -x-- x--x ---- -x-- !al9&!al8&!al7&al6&!altloc
                                                     &!refcyc&!raminh&memr
row       28 -x-- x--- -x-- x--- x--- x--x -x-- ---- !al9&al8&!al7&al6&altloc
                                                     &!refcyc&!raminh&memw
row       29 -x-- x--- -x-- x--- x--- x--x ---- -x-- !al9&al8&!al7&al6&altloc
                                                     &!refcyc&!raminh&memr

row       32 ---- ---- ---- ---- ---- ---- ---- ----
row       33 ---- ---- ---- ---- ---- -x-- ---- ---- refcyc
row       34 -x-- -x-- -x-- -x-- -x-- x--x -x-- ---- !al9&!al8&!al7&!al6&!altloc
                                                     &!refcyc&!raminh&memw
row       35 -x-- -x-- -x-- -x-- -x-- x--x ---- -x-- !al9&!al8&!al7&!al6&!altloc
                                                     &!refcyc&!raminh&memr
row       36 -x-- x--- -x-- -x-- x--- x--x -x-- ---- !al9&al8&!al7&!al6&altloc
                                                     &!refcyc&!raminh&memw
row       37 -x-- x--- -x-- -x-- x--- x--x ---- -x-- !al9&al8&!al7&!al6&altloc
                                                     &!refcyc&!raminh&memr
```

```
row      48 ---- ---- ---- ---- ---- ---- ---- ----
row      49 -x-- -x-- ---- ---- -x-- x--x ---- ---- !al9&!al8&!altloc
                                                    &!refcyc&!raminh
row      50 -x-- x--- -x-- -x-- x--- x--x ---- ---- !al9&al8&!al7&!al6&altloc
                                                    &!refcyc&!raminh

row      56 ---- ---- ---- ---- ---- ---- ---- ----
row      57 -x-- x--- -x-- -x-- x--- x--x -x-- ---- !al9&al8&!al7&!al6&altloc
                                                    &!refcyc&!raminh&memw
row      58 -x-- x--- -x-- -x-- x--- x--x ---- -x-- !al9&al8&!al7&!al6&altloc
                                                    &!refcyc&!raminh&memr
row      59 -x-- -x-- ---- ---- -x-- x--x -x-- ---- !al9&!al8&!altloc
                                                    &!refcyc&!raminh&memw
row      60 -x-- -x-- ---- ---- -x-- x--x ---- -x-- !al9&!al8&!altloc
                                                    &!refcyc&!raminh&memr

pins               11   11   11   11   11   11   11
            2211 3388 4477 5566 6655 7744 8833 9911
polarity    HLHL HLHL HLHL HLHL HLHL HLHL HLHL HLHL


pin:    1  ! ioacc
pin:    2    al9
pin:    3    al8
pin:    4    al7
pin:    5    al6
pin:    6    altloc
pin:    7  ! refcyc
pin:    8  ! memw
pin:    9  ! memr
pin:   11  ! ior
pin:   12  ! casacc
pin:   13  ! memacc
pin:   14    raminh
pin:   15  ! ras0
pin:   16  ! ras1
pin:   17  ! ras2
pin:   18  ! ras3
pin:   19    rdbuff
```

H&L Plot for 82S153

```
*POLHLLLLHLLLL
*P  00 *I  -------L *BI --------L- *BO A....A..AA
*P  01 *I  -------- *BI ------L--L *BO A....A..AA
*P  02 *I  -L------ *BI ---------- *BO .AAAAA..AA
*P  03 *I  LHLHHLL- *BI -----L---- *BO .A...A..AA
*P  04 *I  -HLHHLL- *BI -----L---L *BO .A...A..AA
*P  05 *I  LHHHHHL- *BI -----L---- *BO .A...A..AA
*P  06 *I  -HHHHHL- *BI -----L---L *BO .A...A..AA
*P  07 *I  LHLLHLL- *BI -----L---- *BO ..A..A..AA
*P  08 *I  -HLLHLL- *BI -----L---L *BO ..A..A..AA
*P  09 *I  LHHLHHL- *BI -----L---- *BO ..A..A..AA
*P  10 *I  -HHLHHL- *BI -----L---L *BO ..A..A..AA
*P  11 *I  LHLHLLL- *BI -----L---- *BO ...A.A..AA
*P  12 *I  -HLHLLL- *BI -----L---L *BO ...A.A..AA
*P  13 *I  LHHHLHL- *BI -----L---- *BO ...A.A..AA
*P  14 *I  -HHHLHL- *BI -----L---L *BO ...A.A..AA
*P  15 *I  LHLLLLL- *BI -----L---- *BO ....AA..AA
*P  16 *I  -HLLLLL- *BI -----L---L *BO ....AA..AA
*P  17 *I  LHHLLHL- *BI -----L---- *BO ....AA.AAA
*P  18 *I  -HHLLHL- *BI -----L---L *BO ....AA.AAA
*P  19 *I  -HL--LL- *BI -----L---- *BO .....AA.AA
*P  20 *I  -HHLLHL- *BI -----L---- *BO .....AA.AA
*P  21 *I  LHL--LL- *BI -----L---- *BO .....A.AAA
*P  22 *I  -HL--LL- *BI -----L---L *BO .....A.AAA
*P  23 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  24 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  25 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  26 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  27 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  28 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  29 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  30 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  31 *I  00000000 *BI 0000000000 *BO AAAAAAAAAA
*P  D9 *I  -------- *BI ----------
*P  D8 *I  -------- *BI ----------
*P  D7 *I  -------- *BI ----------
*P  D6 *I  -------- *BI ----------
*P  D5 *I  -------- *BI ----------
*P  D4 *I  00000000 *BI 0000000000
*P  D3 *I  -------- *BI ----------
*P  D2 *I  -------- *BI ----------
*P  D1 *I  00000000 *BI 0000000000
*P  D0 *I  00000000 *BI 0000000000
```

List File

```
Cupl 1.0   Assisted Technology, Inc.  Copyright (c) 1982

                       Partno       PL0000257;
                       Name         VIDTIM;
                       Date         4/20/82;
                       Revision     03;
                       Designer     R. Osann;
                       Company      Assisted Technology, Inc.;

/*************************************************************************/
/*  This device is clocked by the video dot clock and generates the */
/*  character clock and screen Shift/Load signals as well as arbi-   */
/*  tration between the CPU and CRTC for the screen RAM.               */
/*************************************************************************/

pin 2          = !reset ; /* system reset signal*/
pin 4          = !sramsel ; /* CPU access to screen RAM */
pin 5          = !memw ;
pin [18,19]    = ![q1,q0] ; /* state variable bits */
pin 17         = !cpu_cycle ; /* CPU cycle where VIDTIM
                                 performs arbitration */
pin 16         = shift_load ; /* Shift/Load signal to video S/R */
pin 15         = cclk ; /* CRTC character clock */
pin 14         = !sramoe ; /* screen RAM output enable */
pin 13         = !sramwe ; /* screen RAM write enable */
pin 12         = !xack ; /* transfer acknowledge signal, used for
                            driving system ready signal active */

t1              = !q0 & !q1 & !cclk &  shift_load ;
t2              =  q0 & !q1 & !cclk &  shift_load ;
t3              = !q0 &  q1 & !cclk &  shift_load ;
t4              =  q0 &  q1 & !cclk &  shift_load ;
t5              =  q0 &  q1 & !cclk & !shift_load ;
t6              = !q0 & !q1 &  cclk &  shift_load ;
t7              =  q0 & !q1 &  cclk &  shift_load ;
t8              = !q0 &  q1 &  cclk &  shift_load ;
t9              =  q0 &  q1 &  cclk &  shift_load ;

q0.d            = !reset & (t1 # t3 # t4 # t6 # t8) ;
q1.d            = !reset & (t2 # t3 # t4 # t7 # t8) ;
shift_load.d    = !(!reset & t4) ;
cclk.d          = !(!reset & (t9 # t1 # t2 # t3 # t4)) ;
cpu_cycle.d     = !reset & (t4 & sramsel & !xack # cpu_cycle &
                  (t5 # t6 # t7 # t8)) ;

sramoe.d        = t1 # t2 # t3 # t4 # t9 # cpu_cycle & !memw ;
sramwe.d        = cpu_cycle & memw & (t6 # t7) ;
xack.d          = cpu_cycle & t9 # xack & sramsel ;
```

```
partno     PL0000257
name       VIDTIM                          Fuse Plot for
rev        03
date       4/20/82                           PAL  16R8
designer   R. Osann
company    Assisted Technology, Inc.

column                    11 1111 1111 2222 2222 2233
           0123 4567 8901 2345 6789 0123 4567 8901

row      0 x-x- --x- ---- --x- ---x ---- ---- ---- !reset&!q0&!ql&shift_load
                                                   &!cclk
row      1 x-x- ---x ---- --x- ---x ---- ---- ---- !reset&!q0&ql&shift_load
                                                   &!cclk
row      2 x--x ---x ---- --x- ---x ---- ---- ---- !reset&q0&ql&shift_load
                                                   &!cclk
row      3 x-x- --x- ---- --x- --x- ---- ---- ---- !reset&!q0&!ql&shift_load
                                                   &cclk
row      4 x-x- ---x ---- --x- --x- ---- ---- ---- !reset&!q0&ql&shift_load
                                                   &cclk

row      8 x--x --x- ---- --x- ---x ---- ---- ---- !reset&q0&!ql&shift_load
                                                   &!cclk
row      9 x-x- ---x ---- --x- ---x ---- ---- ---- !reset&!q0&ql&shift_load
                                                   &!cclk
row     10 x--x ---x ---- --x- ---x ---- ---- ---- !reset&q0&ql&shift_load
                                                   &!cclk
row     11 x--x --x- ---- --x- --x- ---- ---- ---- !reset&q0&!ql&shift_load
                                                   &cclk
row     12 x-x- ---x ---- --x- --x- ---- ---- ---- !reset&!q0&ql&shift_load
                                                   &cclk

row     16 x--x ---x -x-- --x- ---x ---- ---- --x- !reset&q0&ql&sramsel
                                                   &shift_load&!cclk&!xack
row     17 x-x- ---x ---x --x- --x- ---- ---- ---- !reset&!q0&ql&cpu_cycle
                                                   &shift_load&cclk
row     18 x--x --x- ---x --x- --x- ---- ---- ---- !reset&q0&!ql&cpu_cycle
                                                   &shift_load&cclk
row     19 x-x- --x- ---x --x- --x- ---- ---- ---- !reset&!q0&!ql&cpu_cycle
                                                   &shift_load&cclk
row     20 x--x ---x ---x ---x ---x ---- ---- ---- !reset&q0&ql&cpu_cycle
                                                   &!shift_load&!cclk

row     24 x--x ---x ---- --x- ---x ---- ---- ---- !reset&q0&ql&shift_load
                                                   &!cclk

row     32 x--x ---x ---- --x- ---- ---- ---- ---- !reset&q0&ql&shift_load
row     33 x--- ---- ---- --x- ---x ---- ---- ---- !reset&shift_load&!cclk

row     40 ---- ---- ---x x--- ---- ---- ---- ---- cpu_cycle&!memw
row     41 ---x ---x ---- --x- --x- ---- ---- ---- q0&ql&shift_load&cclk
row     42 ---x ---x ---- --x- ---x ---- ---- ---- q0&ql&shift_load&!cclk
row     43 --x- ---x ---- --x- ---x ---- ---- ---- !q0&ql&shift_load&!cclk
row     44 ---x --x- ---- --x- ---x ---- ---- ---- q0&!ql&shift_load&!cclk
row     45 --x- --x- ---- --x- ---x ---- ---- ---- !q0&!ql&shift_load&!cclk
row      48 --x- --x- ---x -xx- --x- ---- ---- ---- !q0&!ql&cpu_cycle&memw
                                                    &shift_load&cclk
row      49 ---x --x- ---x -xx- --x- ---- ---- ---- q0&!ql&cpu_cycle&memw
                                                    &shift_load&cclk

row      57 ---- ---- -x-- ---- ---- ---- ---- ---x sramsel&xack
row      58 ---x ---x ---x --x- --x- ---- ---- ---- q0&ql&cpu_cycle&shift_loac
                                                    &cclk

pins          11   11   11   11   11   11   11   11
            2299 3388 4477 5566 6655 7744 8833 9922
polarity    HLHL HLHL HLHL HLHL HLHL HLHL HLHL HLHL

pin:    2  ! reset
pin:    4  ! sramsel
pin:    5  ! memw
pin:   12  ! xack
pin:   13  ! sramwe
pin:   14  ! sramoe
pin:   15    cclk
pin:   16    shift_load
pin:   17  ! cpu_cycle
pin:   18  ! ql
pin:   19  ! q0
```