
Chapter 10

The Address Decoder

The address decoder's job is to continuously monitor the high order bits of the address bus, and signal the ROM, RAM, and I/O interfaces whenever an address comes along which is intended for them. As such, it has an extremely important function in keeping the computer operating properly.

10-1. Discussion

The SK68K computer has a single address decoder circuit, but it consists of three ICs. Let's look at Fig. 10-1 to understand how it works.

The heart of the decoder is U63, a 16L8 PAL or Programmable Array Logic element. A PAL is somewhat like a fast ROM - it has a number of input lines and output lines; when a combination of ones and zeroes is presented on the input lines, the PAL looks up in its internal memory what to send out on the output lines. But there are several significant differences between a ROM and a PAL - the ROM is more complex because it has a larger number of possible output combinations; being simpler, the PAL can be made faster. The ROM is meant to store numbers; the PAL is meant to replace logic ICs. In this case, the PAL replaces almost a dozen gates and inverters at a saving of space and money.

The PAL uses its inputs like this:

1. If \overline{AS} is negated (high), the PAL does nothing; it needs a low \overline{AS} to ensure that a valid address exists.
2. If address lines A20-A23 are low, representing addresses starting with the binary bits 0000 (i.e., hex addresses from \$000000 through \$0FFFFFF, corresponding to a megabyte of dynamic RAM), the \overline{DRAM} signal is asserted (low). In addition, if the \overline{CAS} input is low, indicating that the dynamic RAM circuitry wants to access a column of dynamic RAM, it asserts one or two of the \overline{CAS} outputs; this activates a group of dynamic

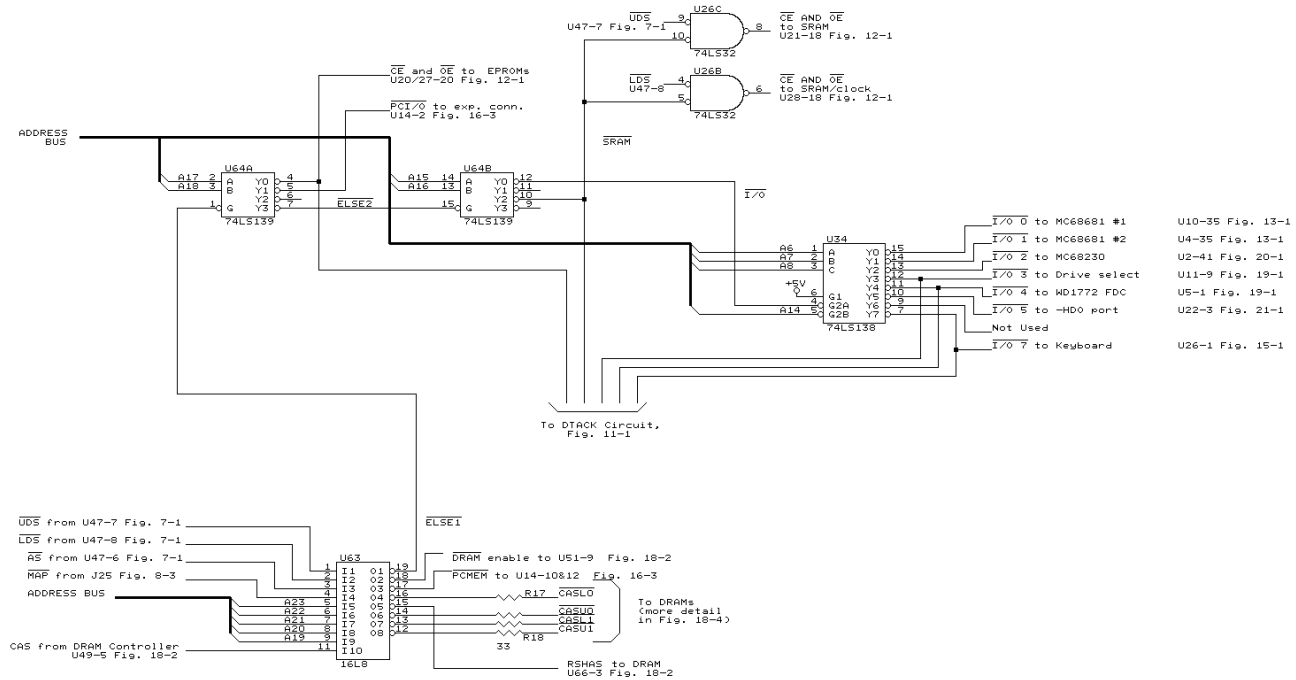


Fig. 10-1. The address decoder.

RAMICs. \overline{LDS} and \overline{UDS} decide whether to use one of the \overline{CASL} or \overline{CASU} outputs (for either an odd byte or even byte or both), while A19 splits the megabyte into a lower 512K and an upper 512K. Incidentally the four resistors, parts of 33-ohm resistor packs R17 and R18, slightly slow down the rise and fall times of these signals. The \overline{CASL} and \overline{CASU} each go to eight dynamic RAM chips, and fast rise and fall times cause sharp signal edges which contribute to noise in the memory. The resistors reduce this noise and improve reliability.

3. If the address begins with the three bits 110, representing all addresses beginning with a hex C (1100) or hex D (1101), the \overline{PCMEM} signal is asserted. This signal goes to the six PC-compatible expansion connectors. ¹
4. If all five address lines are 1, representing all addresses beginning with the bits 11111 (addresses between \$F80000 and \$FFFFFF), the line labelled $\overline{ELSE1}$ is asserted; this signal implies that this address is something else besides dynamic RAM or memory on expansion slots.
5. Operation is a bit different if \overline{MAP} is low; if so, then the dynamic RAM is disabled and the $\overline{ELSE1}$ signal is asserted instead of it. This substitutes ROM instead of RAM at address 0.

¹ In keeping with the standards of Intel microprocessors, cards plugged into the expansion connectors can be addressed either as memory or as I/O. In the SK68K, both of these addresses appear as memory, but at different addresses.

In other words, the PAL splits the 16 megabyte address space of the computer into three main areas:

\$000000 - \$0FFFFFF	$\overline{\text{DRAM}}$	dynamic RAM
\$C00000 - \$DFFFFFF	$\overline{\text{PCMEM}}$	PC expansion memory space
\$F80000 - \$FFFFFF	$\overline{\text{ELSE1}}$	all else

Any other address simply doesn't get recognized by the address decoder.

When $\overline{\text{ELSE1}}$ is low, addresses in its range of \$F80000 - \$FFFFFF are further split up into three smaller groups by U64a, depending on address bits A18 and A17. U64 asserts one of its outputs when it is enabled (the $\overline{\text{G}}$ input is low); which output is asserted depends on the binary input on its B and A inputs. It asserts the Y0 output when it receives a 00, the Y1 output when it receives 01, and so on - it simply interprets the B and A inputs as a number between 0 and 3.

1. If A18 and A17 are 00, which occurs for addresses \$F80000 - \$F9FFFF, U64a asserts the $\overline{\text{CE}}$ and $\overline{\text{OE}}$ signals for both EPROMs, enabling them. Note how U64 has bubbles on the output to remind us that the outputs are active low.
2. If A18 and A17 are 01, which occurs for address \$FA0000 - \$FBFFFF, U64a asserts the $\overline{\text{PCI/O}}$ line. This signal goes to the six PC-compatible expansion connectors for accessing I/O addresses on plug-in cards. (These are the I/O addresses, not memory addresses.)
3. If A18 and A17 are 10, which occurs for address \$FC0000 - \$FDFFFF, U64a asserts its Y2 output, but this is not used.
4. If A18 and A17 are 11, which occurs for address \$FE0000 - \$FFFFFF, U64a asserts the $\overline{\text{ELSE2}}$ output.

As we're beginning to see, each IC in the chain in Fig. 10-1 uses the output of its neighbor to the left to narrow down the range of addresses it recognizes. Now U64b takes the $\overline{\text{ELSE2}}$ signal from U64a, and splits the \$FE0000 - \$FFFFFF range into two smaller groups, depending on address lines A16 and A15.

1. If A16 and A15 are 00, which occurs for addresses \$FE0000 - \$FE7FFF, U64b asserts the $\overline{\text{I/O}}$ signal which goes to U34.
2. If A16 and A15 are 10, which occurs for addresses \$FF0000 - \$FF7FFF, U64b asserts the $\overline{\text{SRAM}}$ signal which goes to the two static RAM (SRAM) ICs or the clock/calendar. Note how $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$ also come into the picture to enable the upper 8 bits, lower 8 bits, or possibly both. This gating is done with U26b and U26c, and is a good example of the need for bubbles to simplify a circuit. In U26c, for example, if $\overline{\text{UDS}}$ is asserted and $\overline{\text{SRAM}}$ is asserted, then the $\overline{\text{CE}}$ and $\overline{\text{OE}}$ pins of U21 are asserted and this SRAM IC is enabled. Since both inputs and outputs of U26c have bubbles, all of these signals are low when asserted.
3. (If A16 and A15 are either 01 or 11, then U64b asserts one of its unused outputs.)

That brings us to U34, the last IC in the address decoder, which gets the $\overline{I/O}$ signal which is asserted for addresses \$FE0000 - \$FE7FFF. In addition, though, U34 also needs a low or zero on A14, so it actually only operates for addresses in the range from \$FE0000 through \$FE3FFF. U34 now looks at address lines A8, A7, and A6 to decide which of its outputs to assert. Like U64, U34 interprets the three-bit number on its C, B, and A inputs as a binary number between 0 and 7, and asserts Y0 through Y7 in response.

U34's job is to assign addresses to seven I/O devices: the two MC68681 DUART serial ICs, the MC68230 PIT parallel port, a drive select latch which controls the floppy drive, the WD1772 floppy disk controller IC, an optional WD1002 hard disk controller, or a PC-compatible keyboard.

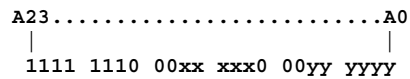
For example, when A8, A7, and A6 are 000, U34 asserts the $\overline{I/O0}$ line on Y0, which enables the first MC68681 DUART.

Now comes a difficult question - what address range does this DUART respond to? Following the circuit from left to right, we see that, for $\overline{I/O0}$ to be low (selecting the first DUART),

- (a) A23 through A19 must be 11111
- (b) A18 and A17 must be 11
- (c) A16 and A15 must be 00
- (d) A14 must be 0
- (e) A13 through A9 are unknown
- (f) A8 through A6 must be 000
- (g) A5 through A0 are unknown.

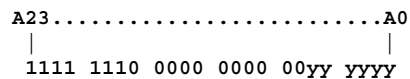
Thus we see that this address decoder does not really look at all the bits of the address bus - there are eleven bits unaccounted for (or ten bits when we realize that A0 doesn't exist.) Let's not worry about A5 through A0, since we will see that they do play a role later. But A13 through A9 are the problem.

Let's put the above bits in order, left to right, like this:



The five unknown bits in the middle are labelled xxxxx, and the six unknown bits on the right are labelled yyyyyy. Clearly these x and y bits could be anything - all zeroes, all ones, or any combination of zeroes and ones.

Let's begin by assuming that the x bits are all zeroes, giving us



Since the bits are already separated into groups of four, let's follow through and convert them to hexadecimal. The first four digits are clearly \$FE00. If the y bits are all zeroes, then the last two digits are \$00; if they are all ones, then the last two digits are \$3F. This tells us that the DUART can be addressed in the range from \$FE0000 through \$FE003F. This is a total of

64 addresses, and is in fact the range of addresses that we use in programming it.

But the x digits need not be zeroes - they could just as well be 00001, which would make the complete address look like this:

```
A23.....A0
|          |
1111 1110 0000 0010 00yy yyyy
```

Again assuming that the y digits can be anything from all zeroes to all ones, this gives us an address range of \$FE0200 through \$FE023F. Continuing like this, we see that the DUART responds to many addresses:

```
$FE0000 through $FE003F
$FE0200 through $FE023F
$FE0400 through $FE043F
$FE0600 through $FE063F
$FE0800 through $FE083F
```

and so on, all the way up to

```
$FE3E00 through $FE3E3F
```

when the x bits are all ones.

This is all an example of *incomplete address* decoding, caused by the fact that the address decoder does not decode all the bits - some of the bits can be anything, and are usually called *don't cares*. Incomplete address decoding results in the use of more addresses than are actually needed.

The EPROM is another example of incomplete decoding. As we learned above, the EPROM is assigned addresses from \$F80000 through \$F9FFFF, a total of 128K bytes. But in a typical case, the EPROM may consist of a pair of 27128s, which provide a total of only 32K of EPROM, so that 96K of addresses (128K minus 32K) are wasted. What actually happens is that the 32K of actual EPROM appears in that 128K space four times. That is, the EPROM appears to take up the entire 128K, but on closer examination we see that there are four copies of the same data in that space. Such a loss of addresses in an 8-bit computer with a total of 64K of addresses would be unthinkable; in a computer having 16 megabytes of addresses, the loss of 96K is unimportant.

The same situation occurs with the I/O. Although each of the I/O devices may only require a few bytes, each takes up thirty-two 64-byte chunks of addresses, for a total of 2048 addresses.

10-2. Construction

Install the following components:

U63	16L8 PAL and its socket
R17 and R18	33-ohm resistor packs soldered directly to the board without a socket
U64	74LS139 dual decoder and its socket
U34	74LS138 decoder and its socket

C13, C67 and C70 0.1 μ F disc capacitors

Finally, place the J25 jumper in position 1.

10-3. Testing

Recheck all connections and then apply power.

Although a really thorough test of the address decoder requires sophisticated equipment, we can nevertheless do a simple check with the aid of Fig. 10-2, which shows the waveforms output by the decoder circuit.

As you remember, the 68000 is still executing what it thinks is 4 million OR instructions, looping through memory once every four seconds. Think of Fig. 10-2 as showing what happens during those four seconds (there is a tiny bit at the right end of the figure which shows how everything repeats after those four seconds are up). During those four seconds, however, the 68000 is also counting up through 16 megabytes of memory. Thus the left of the figure corresponds to its trying to access location 0; the right end corresponds to its trying to read location \$FFFFFF, which is at the top of the 16 megabytes; the midpoint corresponds to location \$800000, the 8-megabyte point of memory. So Fig. 10-2 also represents the 16 megabyte range of memory, and some of the more important memory addresses are shown at the bottom.

Now let's look at the $\overline{\text{DRAM}}$ signal. Since dynamic RAM will comprise the first 1 megabyte out of the sixteen, the $\overline{\text{DRAM}}$ signal goes low for the first 1/16th of the four seconds; that works out to one quarter of a second in terms of actual time. It is shown as a solid block in Fig. 10-2, however, because it does not stay low all that time. Like all other signals in the address decoder, $\overline{\text{DRAM}}$ is asserted only when the $\overline{\text{AS}}$ address strobe is on; since $\overline{\text{AS}}$ keeps rapidly cycling high and low, so does $\overline{\text{DRAM}}$. If you connect your LED probe to the $\overline{\text{DRAM}}$ output, pin 18 of U63, you will note that the LED is on continuously (since the signal is high), but once every four seconds, the LED will get slightly dimmer for about a quarter of a second. An oscilloscope would show a continuous high, with a short burst of pulses once every four seconds.

Each of the other address decoder outputs can be checked in the same way except for the four CAS outputs (which will show a constant high

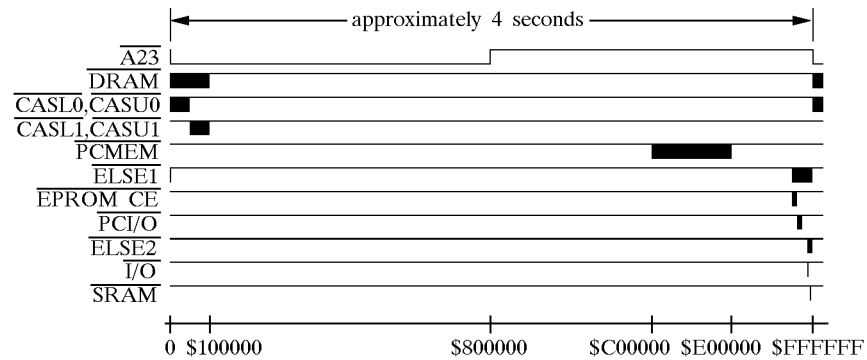


Fig. 10-2. Address decoder output waveforms.

because we are not supplying a CAS input to U63.) Some of the other outputs will be longer - such as PCMEM which will dim the LED for a 1/2 second - while most will be much shorter. The I/O and SRAM outputs will be very difficult to see since they only last about ten milliseconds, just barely long enough to flicker the LED if you watch carefully. The outputs of U34 are too short to see on our LED, although a pulse-catching logic probe or oscilloscope will show a slight flicker once every four seconds.

