3 Decoding on the PC/104 bus

Although the PC/104 bus has 24 bits for addressing, we consider first the use of the first 20 bits for memory chips, then the first 10 bits for I/O devices. To see what is needed, we will examine the interfacing of some memory devices.

3.1 Accessing a memory device

As we have considered the address, data and control busses from the processor point of view, now we consider them from the *memory* point of view.

3.1.1 Memory address, data, control bus

In contrast to the *processor* address bus, the *memory* address bus will tell a memory device which particular location is being accessed. The number of *memory* address lines required depends on the size of the memory device. (e.g. a device with 1K (i.e. 1024) locations, requires 10 address lines, because $2^{10} = 1024$).

In the same way, the quantity of data stored at each *memory* location depends on the particular memory device. It may be a single bit, or 4, 8 or 16 bits per location. Units of memory may be connnected in parallel to meet the processor data width. For example, four 8 bit memory devices may be connected in parallel to a 32 bit processor.

3.1.2 Steps to access memory

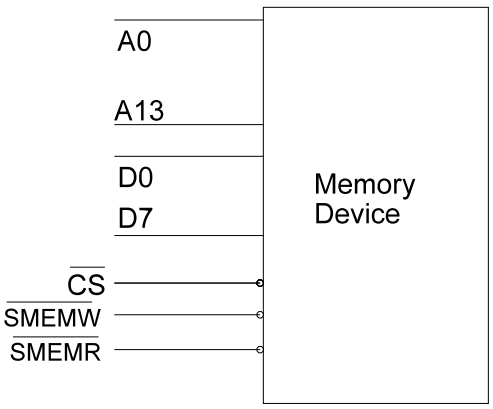


Fig 3.1 Steps to access memory

The block diagram of a memory device is as shown. To access data, the following steps are executed.

1) The address is placed on the address bus. The lower bits will be used by the memory device for internal decoding, the higher bits go to a decoder chip which will generate a chip select (_CS) signal. This will then select one of the several memory devices.

2) READ

If doing a read operation, the processor will generate a read signal (RD) which is connected to the output enable (OE) of the memory chip. The memory device will place the data from the memory location after a short delay. The processor then reads in the data.

3) WRITE

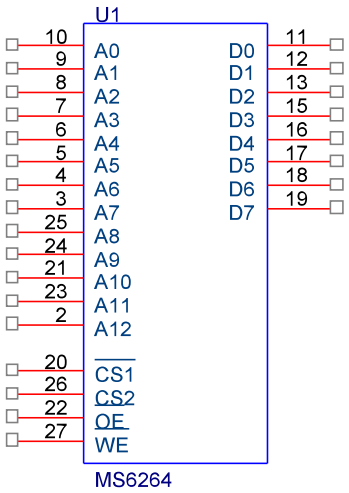
If doing a write operation, the processor will places the data to be written on the data bus, then generates a write (_WR) signal, which is connected to the write enable (_WE) of the memory chip. The memory device will then latch the data in.

3.1.3 Partial and Full Address Decoding

In most practical situations, the devices connected to a processor address bus require fewer address bits than there are bits provided on the address bus. The PC/104 has 20 pins while a 6264 RAM has only 13 pins. Since there are 7 pins unused by the RAM chip, then this is a case of *partial address decoding*. If we have a memory system where all the address pins are used in memory access, we then have *full address decoding*.

As an illustration of how partial and full address decoding are implemented, we shall look at how a memory chip can be connected to a 20-bit address bus.

3.1.3.1 A memory device



Pin Name	Function
A0 to A12	Address input
D0 to D7	Data input/output
CS1	Chip select 1
CS2	Chip select 2
WE	Write enable
OE	Output enable
Vcc	Power supply
Vss	Ground
NC	Pin for no connection

Fig 3.2 6264 schematic

Pins A0 to A12 are the 13-bit address inputs to the 8192 locations in the RAM while I/O1 to I/O8 are the 8 data bits in each location. CS1 to CS2 are the chip select inputs (active high or low depending on whether there is a bar on top) that enable the chip.

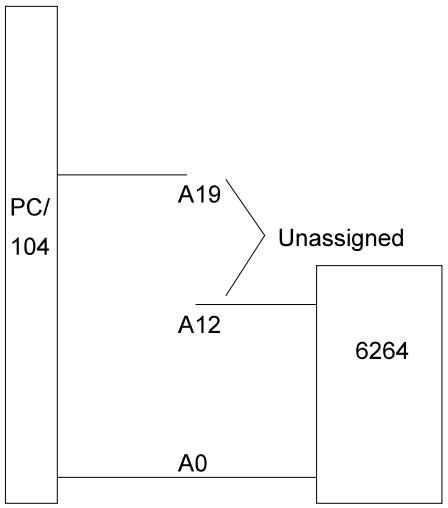


Fig 3.3 Foldover

When the 6264 is connected to the PC/104 address bus as shown in only the states of the address lines A0 to A12 will determine which location inside the RAM is accessed. The other address lines A13 to A19 are ignored by the chip. Hence the location 0 0000 0000 0000B can be accessed by the addresses XXXX XXX0 0000 0000 0000B. Therefore addresses from the processor bus 00000H, 02000H and so on will all access the same *memory* location 0000H.

This is an inefficient way of using the address bus since even though 2^{20} = 1048576 locations can be selected individually, only 8K are used, each responding to 1048576/8192 = 128 different *processor* addresses.

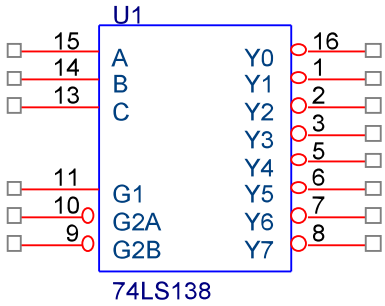
Memory Foldover

When a memory address (or group of physical addresses) responds to several different *processor* addresses, the effect is known as memory foldover. In the example above, there is a 128 times foldover.

In small systems, this is tolerable. But we see two problems here. First is that if more than one device needs to be attached to the bus, how do we fit it in? Secondly is that a programmer may access data from the wrong memory location.

When connecting several memory ICs to an address bus, we can use logic gates as address decoders using K-Maps or standard Boolean logic. However, it is much easier to use demultiplexers or decoders, because of the way memory is laid out, which in many cases is in sequential blocks of addresses, as we will see shortly.

One of the most commonly used demultiplexers is the 74138 1-of-8 demultiplexer. Looking at the output pins, note that the pin that goes low will correspond to the binary input at pins A, B and C. For example, an input of 111 will cause pin Y7 to go low.



PIN No	SYMBOL	NAME AND FUNCTION
1,2,3	A,B,C	Address Inputs
4,5	_G2A, _G2B	Enable Inputs
6	G1	Enable Inputs
8->15	_Y0 to _Y7	Outputs
7	GND	Ground (0V)
16	Vcc	Positive Supply Voltage

Fig 3.4 74LS138 schematic

Truth table

INPUTS						OUTPUTS							
ENABLE			SELECT										
G2B	G2A	G1	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	X	L	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
H	X	X	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

We see that the 74138 can select one of eight possible devices based on its ABC inputs. If we add this to the thirteen inputs of the 6264, we see that we now have sixteen (13+3) inputs, which still causes foldover.

To take care of this, we can still choose to use logic gates, but we introduce another device, the 74LS688 8 bit comparator which may be rather overspecified for the task at hand, but it is versatile.

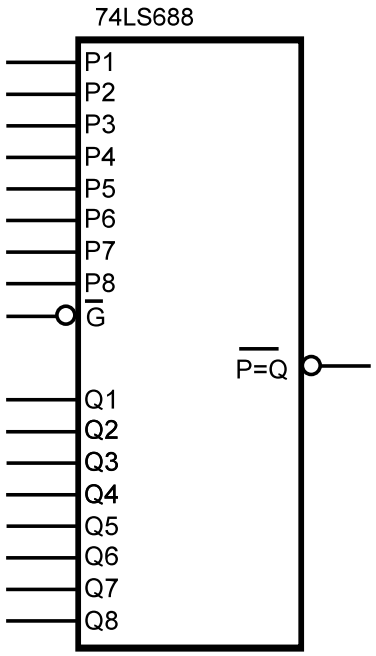


Fig 3.5 74LS688 schematic

PIN No	SYMBOL	NAME AND FUNCTION
1	$\overline{G}$	Enable Input
2,4,6,8,11,13,15,17	P	P word input
3,5,7,9,12,14,16,18	Q	Q word inputs
19	$\overline{P=Q}$	equal to output
10	GND	Ground (0V)
20	Vcc	Positive Supply Voltage

Truth table

Inputs		$\overline{P=Q}$
Data P,Q	Enable $\overline{G}$	
P = Q	L	L
P > Q	L	H
P < Q	L	H
X	H	H

It has only one output, $\overline{P=Q}$. It is quite obvious that when the digital value at the P inputs equal the digital value at the Q inputs, then the $\overline{P=Q}$ pin will go low.

What makes this device versatile is that we do not need inverted versions of address signals, which is true if we use the 74138. We can set the desired bit pattern at either the P or Q inputs.

Another important point is that there are other devices connected to the system that use up certain parts of the memory map. Thus we need flexibility in locating the starting address of our memory block.

EXAMPLE: Design a 64K memory space starting from E0000H, using 6264 memory chips.

In designing a full decoding system using a 74138, we should note the number of address lines are used by the memory chip itself. This is because these lines cannot be used for decoding. Thus only the remaining lines on the host processor address bus is available.

To summarize, we may use the following steps:

1) Identify how many address pins the memory chip uses.
2) Calculate the number of chips that are required in the design.
3) Draw the memory map of the design. Do this by laying out the chips contiguously, that is, the starting address of a chip follows immediately after the last address of the previous chip. Write down the range of addresses of each chip in hexadecimal.
4) Draw a truth table. This is the binary representation of the range of addresses.
5) Looking at the bits, observe any pattern of numbers, for example, running binary numbers in the bits.
6) Fit this pattern to the truth table of the 74138. First, see which address pins are to be assigned to inputs A, B and C. Then see how to assign the remaining address pins to the Enable inputs of the 74138.
7) If there are unassigned *processor* address pins, assign them to the 74688.
8) Finally, produce the schematic.

Applying to the above example:

1) The memory device to be used is a 6264, which is a 8192 by 8 bit device. This device uses address pins A0 to A12. (2^{13} = 8192). Each chip will take up 8192_{10} or 1FFFH addresses.
2) Since we require 64K bytes in the design, we need 8 chips. (64K/8K).
3) This 64K will occupy the range from 0E0000H to 0EFFFFH. So the *first* memory chip will start from E0000H and occupy the address range E0000 - E1FFFH. We continue until we get the following memory map:

RAM 1	-	E0000H to E1FFFH
RAM 2	-	E2000H to E3FFFH
RAM 3	-	E4000H to E5FFFH
RAM 4	-	E6000H to E7FFFH
RAM 5	-	E8000H to E9FFFH
RAM 6	-	EA000H to EBFFH
RAM 7	-	EC000H to EDFFH
RAM 8	-	EE000H to EFFFFH

To draw the truth table, let us look at the first chip. It occupies the address range E0000-E1FFFH. Using binary, and noting that the MSB is address line A19:

MSB

The range in binary is:1111 0000 0000 0000 0000 (E0000H)

1111 0001 0111 1111 1111. (E1FFFH)

We are not interested in the address lines used by the chip at this point. Note that the 0's and 1's change only at lines A0 to A12.

In order to simplify the design process, we will introduce a notation. Instead of writing two lines with all the 0's and 1's, we will represent the above range by:

```
A19              A12    A0
1111   0 0 0 0  0 X--------X.   E0000 - E1FFFH
```

"X" means that the value for the address line can be 0 or 1. The dashed line "X---X" means that address lines from A0 to A12 can be 0 or 1. So they are not available for general use. This highlights the fact that only address lines A19 to A13 are available for decoding. Proceeding in similar fashion for the other chips, we have the following truth table.

```
A19 A18 A17 A16 A15 A14 A13 A12 -- - - - - - - - - - A0
1   1   1   0   0   0   0   x------------------------x        E0000-E1FFFH
1   1   1   0   0   0   1   x------------------------x        E2000-E3FFFH
1   1   1   0   0   1   0   x------------------------x        E4000-E5FFFH
1   1   1   0   0   1   1   x------------------------x        E6000-E7FFFH
1   1   1   0   1   0   0   x------------------------x        E8000-E9FFFH
1   1   1   0   1   0   1   x------------------------x        EA000-EBFFFH
1   1   1   0   1   1   0   x------------------------x        EC000-EDFFFH
1   1   1   0   1   1   1   x------------------------x        EE000-EFFFFH
```

1. Looking at the truth table, we can see the address lines A15 to A13 taking on values that increment, starting from 000 to 111. If we assign these lines to a 74138, we can enable up to 8 devices one at a time, depending on the values of A15 to A13.

2. Comparing this with the truth table of the 74138, we see that A15 should be assigned to input C, A14 to B and A13 to A. Note that this incrementing bit pattern is only true for this situation. Other situations will have different patterns at different address lines. The important thing is to look out for a pattern to fit to the truth table of the 74138.

3. It is convenient to left A19 to A16 be enabled by the 74688, so it activates when A19-A16 has the value ‘E’.

The final design is shown below:

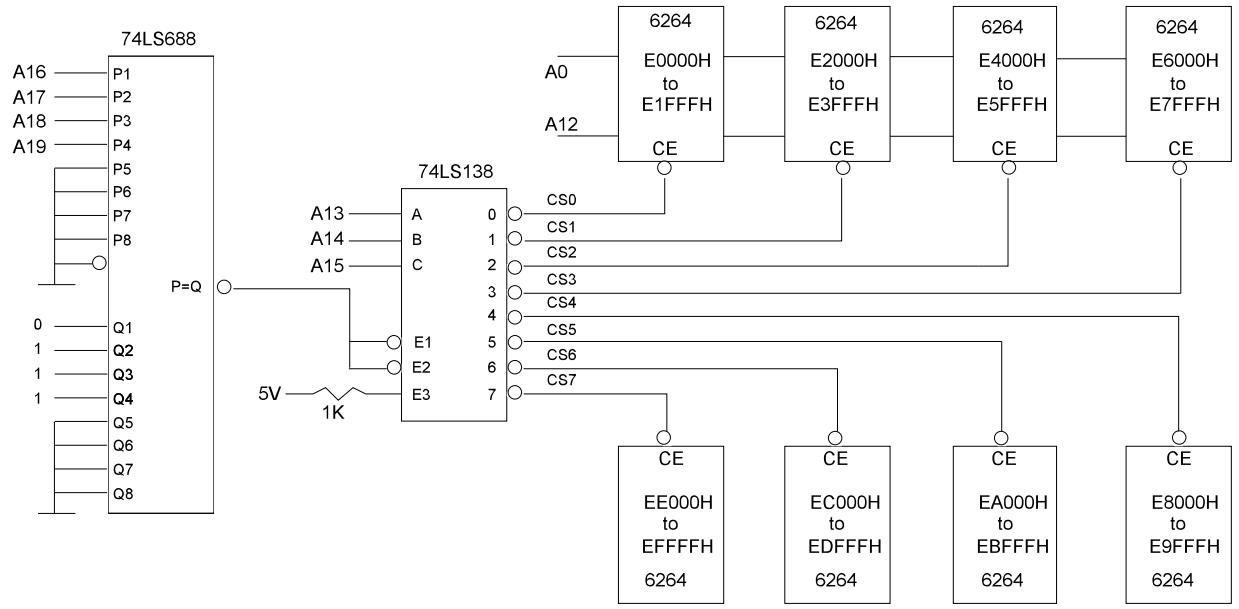


Fig 3.6 Circuit which memory maps 64KB of SRAM starting from E0000H

3.2 Input/Output Decoding

As we saw above, memory devices use only some of the available address lines. The lower address lines will select a single memory location in the memory device. The higher address lines from the system are used by a decoder to select one of several memory devices. Also, the memory device will have an Output Enable pin for a read and a RAM device will also have a Write Enable pin.

3.2.1 Single address decoding

However, buffers and latches have only a single address location. The PC architecture has used up quite a few I/O addresses so that there are few left for the user. Also, other devices connected to the system may have used up other address.

Thus we need to decode each device exactly, so we do not activate other I/O by accident. What this means is that for a 16 bit I/O address space such as that available on Intel microprocessors we need to use all address lines, giving us theoretically 64K addressable I/O devices. But we saw that the PC implementation only uses 10 bits, making the hardware requirements somewhat simpler.

Another consideration is that since the address bus is shared by the main processor and DMA processes, the DMA operation may activate the I/O addresses accidentally. The ISA bus provides for this by using the AEN signal to signify that a DMA operation is taking place when the AEN signal is high. This is used in decoding.

Also, since buffers have only one control pin (normally the Enable pin) and latches use the clock pin, we need an external gate to control the access, so that processor reads are directed to buffers and writes are directed to latches. Of course, if there is no confusion as to the assignment of addresses, the gates may not be required.

Example: Design an input port of address 330H and an output port of address 331H.

Truth Table

```
A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
1  1  0  0  1  1  0  0  0  1          331H
1  1  0  0  1  1  0  0  0  0          330H
```

We let A0-A2 be decoded by the 74138. The bit pattern on the 'Q' inputs correspond to bits A3-A9. We decode when AEN is low, namely when no DMA is occuring.

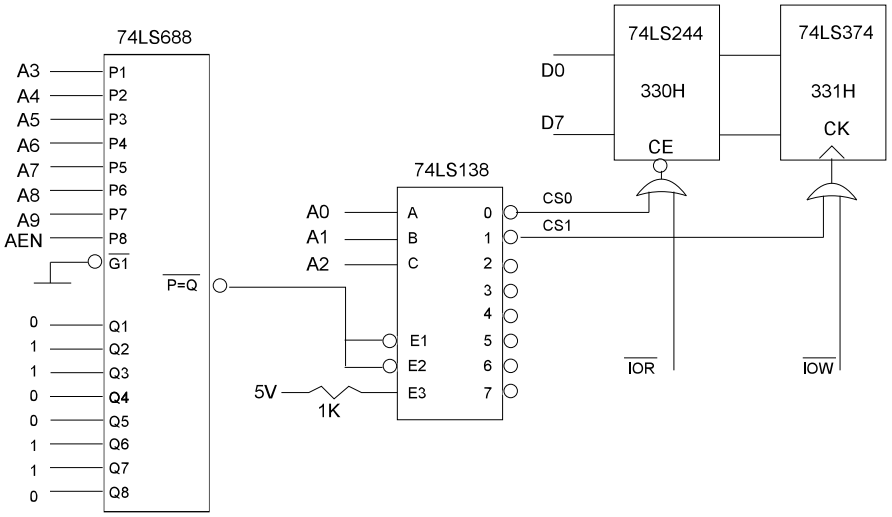


Fig 3.7 I/O decoding with 74LS688

As we see, the 74688 in conjunction with a 74138 allows us to decode individual I/O addresses. The 74688 activates a block of 8 devices through the 74138. The *starting* address of this block will have the A0-A2 pins at logic 0. This address is known as the *base* address as all the other I/O addresses are simply added to the base to get its address.

3.3 Summary

In summary, we saw how memory and I/O devices may be decoded on the PC/104 bus. The TTL chips 74688 and 74138 helped us to easily and flexibly do this. The main differences between memory and I/O is that:

i) Memory devices contain many storage locations within itself. Selecting a memory chip through a decoder only enables the chip. Internal decoding circuitry *within* the memory chip will select individual memory locations.

Buffers and latches, being individual devices, need full decoding to be done externally as it does not have the circuitry internally.

ii) In addition to enable pins, memory devices have control pins which determine whether data is written or read to.

Buffers and latches are normally activated through a single control pin. Again, external gates may be needed to correctly decode the signal.