

Address Bus Decoding*

Charles B. Cameron

April 22, 2005

Design the address bus decoding circuitry for a system with requirements shown in Table 1 using the devices shown in Table 2.

SOLUTION

Memory map

First draw the memory map, as shown in Figure 1. The number of fast

*Course notes for EE461 Microprocessor-based Digital Design

Device	Starting address
768 bytes of high-speed (15 ns) static RAM	0x0400
Four ACIAs	0x1000
One PIA	0x1010
Three buffers	0x1020
2K ROM	0x2800

Table 1: Desired System Configuration

Device	Architecture	Chip-Select Labels
2K RAM (low-speed)	2048×8 bits	CS_0, CS_1, CS_2
4K ROM	4096×8 bits	OE_1, OE_2
MC6820 PIA (Parallel I/O)	4×8 bits	$CS_0, \overline{CS_1}, \overline{CS_2}$
MC6850 ACIA (Serial I/O)	2×8 bits	$CS_0, \overline{CS_1}, \overline{CS_2}$
Octal buffer	1×8 bits	$G_0, \overline{G_1}$
High-speed RAM	256×8 bits	$\overline{CS_0}, \overline{CS_1}$
16K ROM	16,384×8 bits	\overline{OE}
74LS165 Parallel-load 8-bit shift reg	1×8 bits	\overline{LD}
4K low-speed RAM	4096×8 bits	$\overline{OE_1}$

Table 2: Available Devices

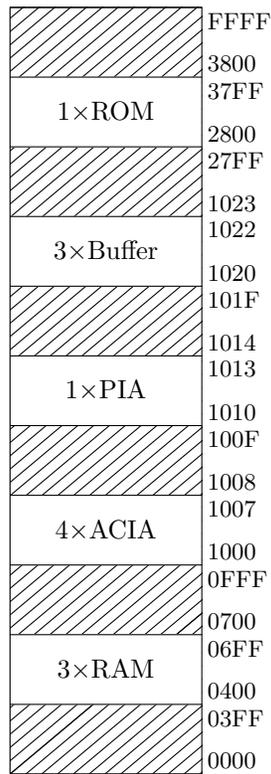


Figure 1: Memory Map. Note that memory-space utilization is not drawn to scale. For example, the 4,096-location ROM is shown as equal in size to the 3-location set of buffers.

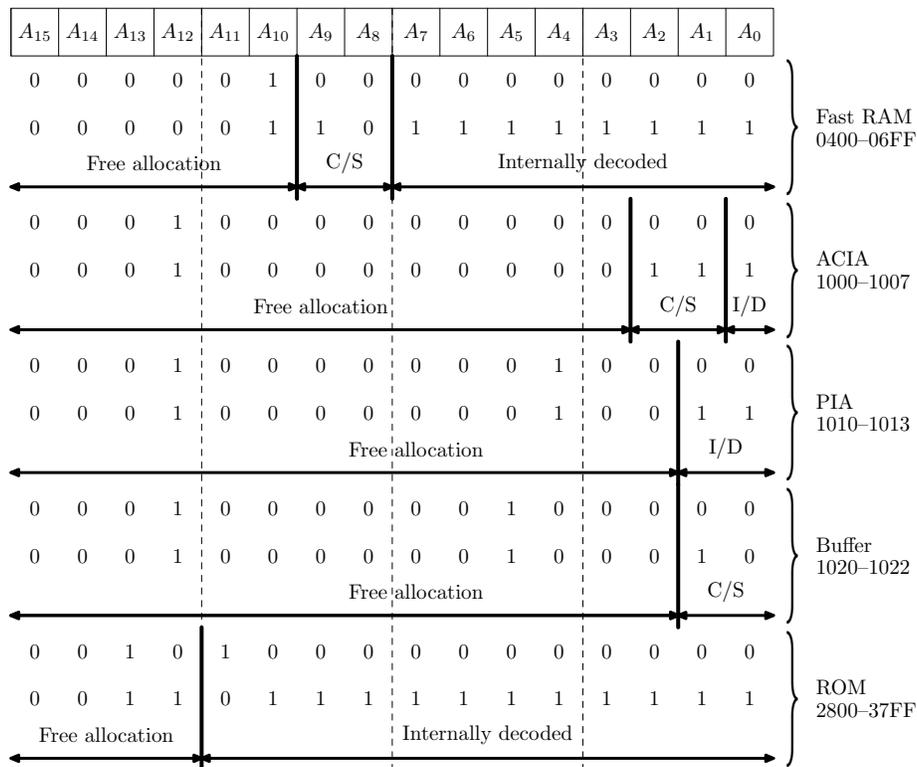


Figure 2: Binary memory map

RAM devices can be obtained by simple arithmetic:

$$(768 \text{ bytes}) \left(\frac{1 \text{ device}}{256 \text{ bytes}} \right) = 3 \text{ devices.}$$

There are two kinds of ROM available: 4K ROM and 16K ROM. In general, the least expensive option is to use the smallest device which meets the requirements. Therefore, we pick the 4K ROM, knowing that about half of it will be completely unused.

The uppermost address occupied by a particular type of device is found by adding its extent to its starting address and reducing this total by 1. In the case of the fast RAM, for example:

$$\begin{aligned} 0x0400 + 768_{10} - 1 &= 0x0400 + 0x0300 - 1 \\ &= 0x06FF. \end{aligned}$$

Next we prepare binary memory maps for each type of device, as seen in Figure 2. The maps show the lowest and the highest address in binary for each type of device. They also show how the address is subdivided for a particular

Binary memory maps

	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
$\overline{A_{13}} \overline{A_{12}}$	0	0	0	0	0	1											Fast RAM (0400–06FF)
$\overline{A_{13}} A_{12} \overline{A_5} A_4$	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	ACIA (1000–1007)
$\overline{A_{13}} A_{12} A_4$	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	PIA (1010–1013)
$\overline{A_{13}} A_{12} A_5$	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	Buffer (1020–1022)
A ₁₃ {	0	0	1	0													} ROM (2800–37FF)
	0	0	1	1													

Figure 3: Free Allocation Table

device. Some address bits are decoded internally, some are used as chip-select bits to choose from among several identical devices, and some are used to determine the type of device being accessed. Some devices, such as the buffer used here, do not decode any address bits internally. Others, such as the PIA, do not need chip-select bits because there is only one of them. All of them, though, include free-allocation bits.

The ROM’s address range differs from the address ranges of the other types of devices in an important respect. Because the ROM’s addresses start at address 0x2800 and straddle a 12-bit address boundary, the free allocation bits for the ROM are not constant. Below the boundary are addresses from 0x2800 to 0x2FFF; above it lie the remaining addresses, from 0x3000 to 0x37FF. While this may not be ideal, it still can be coped with.

The free-allocation table is shown in Figure 3. The ROM is the only device whose address includes a 1 in A₁₃. Because the ROM straddles a 12-bit boundary, bit A₁₂ may be either a 0 or a 1: it is a *don’t-care* bit.

All other device addresses have a 0 in A₁₃. Of these, only the fast RAM has a 0 in A₁₂.

The remaining device addresses have a 1 in A₁₂. Of these, only the buffers have a 1 in A₅.

Finally, the ACIA and the PIA have a 0 in A₅. They can be distinguished from each other because the ACIA has a 0 in A₄ while the PIA has a 1 in A₄.

Armed with equations for identifying the type of device, we can now turn our attention to the chip-select logic for each type of device. In each case, we need to map the equations in the free-allocation table (Figure 3) to the available chip-select inputs (Table 2). In the case of device types with more than a single device of that type, we also need to map the chip-select address bits to the chip-select inputs of each device.

Often it is desirable to use one chip-select input for the equation for selecting a device type and use the remaining ones for the chip-select equations. This works well for the fast RAM devices, as shown in Figure 4, the ACIAs, as shown in Figure 5; and the buffers, as shown in Figure 7. It does not apply so well in the case of the PIA, as shown in Figure 6, or the ROM, as shown in Figure 8, because there is only a single chip of each of these kinds: there is no chip-select logic. There is only free-allocation logic. This actually leads to

Free-allocation table

Chip-select logic

		A ₉	A ₈	$\overline{CS_0}$	$\overline{CS_1}$
Fast RAM (0400–06FF)	0	0		A ₁₃ + A ₁₂	A ₉ + A ₈
	0	1		A ₁₃ + A ₁₂	A ₉ + $\overline{A_8}$
	1	0		A ₁₃ + A ₁₂	$\overline{A_9}$ + A ₈

Figure 4: Fast RAM Chip Select Logic

		A ₂	A ₁	CS ₀	$\overline{CS_1}$	$\overline{CS_2}$
ACIA (1000–1007)	0	0		$\overline{A_{13}}$ A ₁₂ $\overline{A_5}$ $\overline{A_4}$	A ₂	A ₁
	0	1		$\overline{A_{13}}$ A ₁₂ $\overline{A_5}$ $\overline{A_4}$	A ₂	$\overline{A_1}$
	1	0		$\overline{A_{13}}$ A ₁₂ $\overline{A_5}$ $\overline{A_4}$	$\overline{A_2}$	A ₁
	1	1		$\overline{A_{13}}$ A ₁₂ $\overline{A_5}$ $\overline{A_4}$	$\overline{A_2}$	$\overline{A_1}$

Figure 5: ACIA Chip Select Logic

		CS ₀	$\overline{CS_1}$	$\overline{CS_2}$
PIA (1010–1013)		A ₁₂	A ₁₃	$\overline{A_4}$

Figure 6: PIA Chip Select Logic

		A ₁	A ₀	G ₀	$\overline{G_1}$
Buffer (1020–1022)	0	0		$\overline{A_{13}}$ A ₁₂ A ₅	A ₁ + A ₀
	0	1		$\overline{A_{13}}$ A ₁₂ A ₅	A ₁ + $\overline{A_0}$
	1	0		$\overline{A_{13}}$ A ₁₂ A ₅	$\overline{A_1}$ + A ₀

Figure 7: Buffer Chip Select Logic

		OE ₁	$\overline{OE_2}$
ROM (2800–37FF)		A ₁₃	0

Figure 8: ROM Chip Select Logic

less complicated wiring, though, because only the free-allocation logic equations need to be considered at all.

Some of the chip-select inputs are active-low. In such cases, the input equations must be inverted. So, for example, in the case of the fast RAM, we know from the free-allocation table in Figure 3 that the RAM devices are selected whenever we have $\overline{A_{13}} \overline{A_{12}}$. When we connect the inverse of this to the $\overline{CS_0}$ input we get

$$\begin{aligned}\overline{CS_0} &= \overline{\overline{A_{13}} \overline{A_{12}}} \\ &= A_{13} + A_{12}.\end{aligned}$$

At this point we are ready to draw a schematic showing how to wire the devices together. This can be seen in Figure 9. Some of the inverters are represented by bubbles as, for example, on the four-input AND-gate used to supply the CS_0 input for the four ACIA devices. The data outputs and any control signals not directly related to the decoding problem have been suppressed from the schematic.

Schematic
diagram

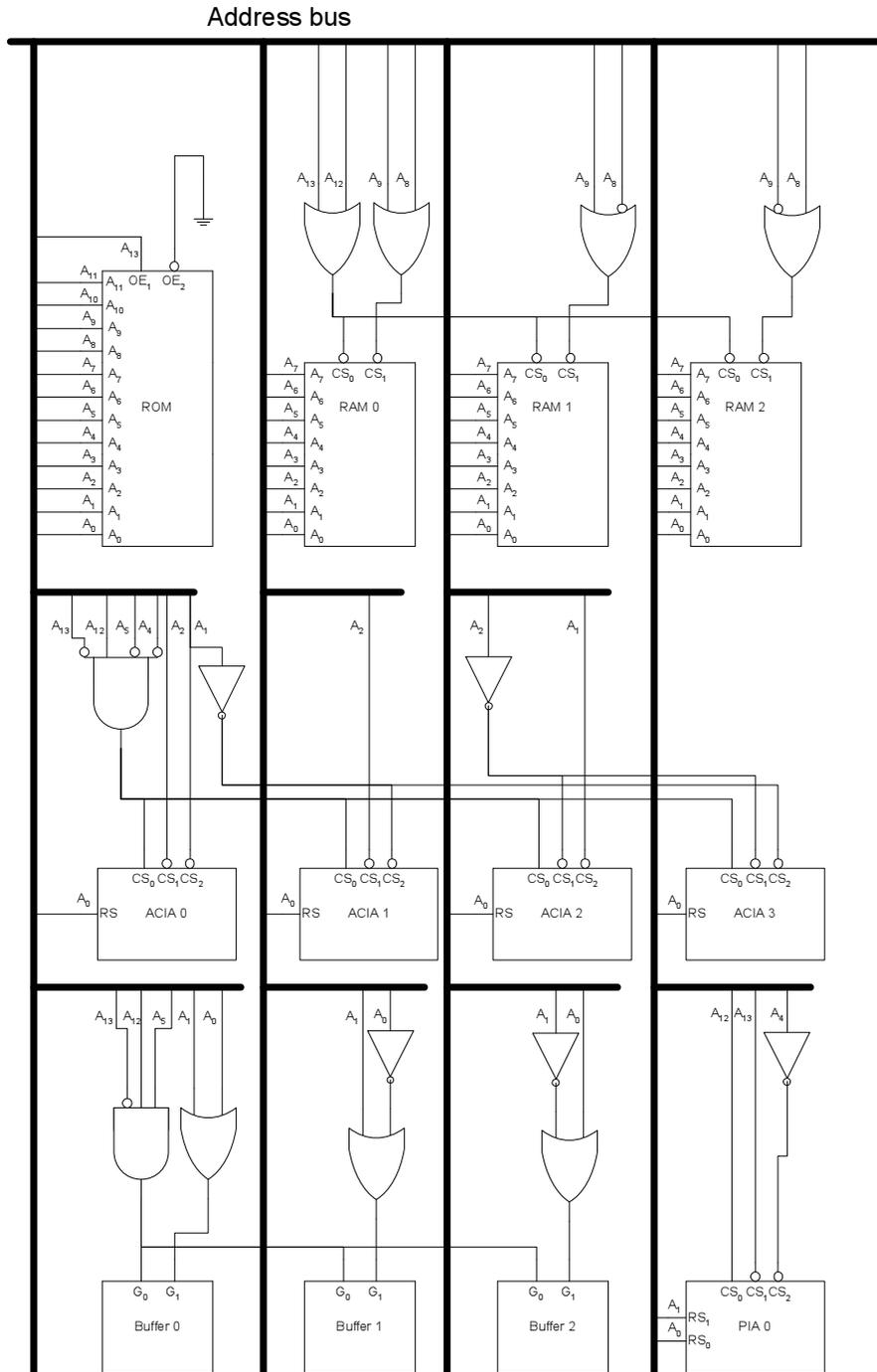


Figure 9: Schematic for the complete decoder