Address Decoding Techniques

What is Address Decoding ?

- Address decoding is a simple process that attempts to ‘map’ a memory device to a particular range of CPU addresses, that is, the memory device will respond to the CPU if the CPU issues an address within the designated range for the device.

- Address decoding allows us to partition the memory space of the CPU into smaller block that can then be designated as areas for Sram, Dram, Flash etc. and even memory mapped IO devices that respond like memory.

- The same techniques allow us to break up a section of say Sram into smaller sections allocated to individual Sram chips.

- For example, in a 68000 system, which has an address space of 8M Words (16M bytes), address decoding might be used to create a 256k word Partition of Sram allocated to the address range [04 0000 – 07 FFFF].

  This is turn could be further broken down into 4 smaller blocks/banks of 64k words [04 0000 – 04 FFFF], …. [07 0000 – 07 FFFF] and then constructed from 4 x 64kword memory chips to occupy the ful 256k word address range.

- Designing an Address decoder is a straightforward combinatorial design problem, where the circuit produces **Chip Select** signals to enable individual memory devices whenever the CPU issues an address corresponding to that chosen for the device.

Address Decoding Techniques

- Decoding memory addresses is analogous to decoding telephone numbers

  - The telephone exchange is analogous to our address decoder.
  - Each telephone number must be unique (within its domain)
  - We use the telephone number to uniquely address individual phones.

- For example dialling the number 001 604 123 4567 will cause the telephone exchange to map the digits 001 to Canada, the digits 604 to say the city of Vancouver the digits 123 to a suburb of Vancouver and the digits 4567 to a unique phone within the suburb. In other words

  - The first 3 digits of the phone number partition the global phone space into a country.
  - The next 3 digits partition the country phone space into a city or region.
  - The next 3 digits partition the city/region phone space into a suburb.
  - Finally the remaining 4 digits partition the suburb space into unique phones with the numbers in the range 0000 - 9999.

- There are lots of design issues, trade offs and technologies that we can use to solve this problem.

Address Decoding Techniques

- In a computer system the same principles of separating/partitioning into domains occurs.

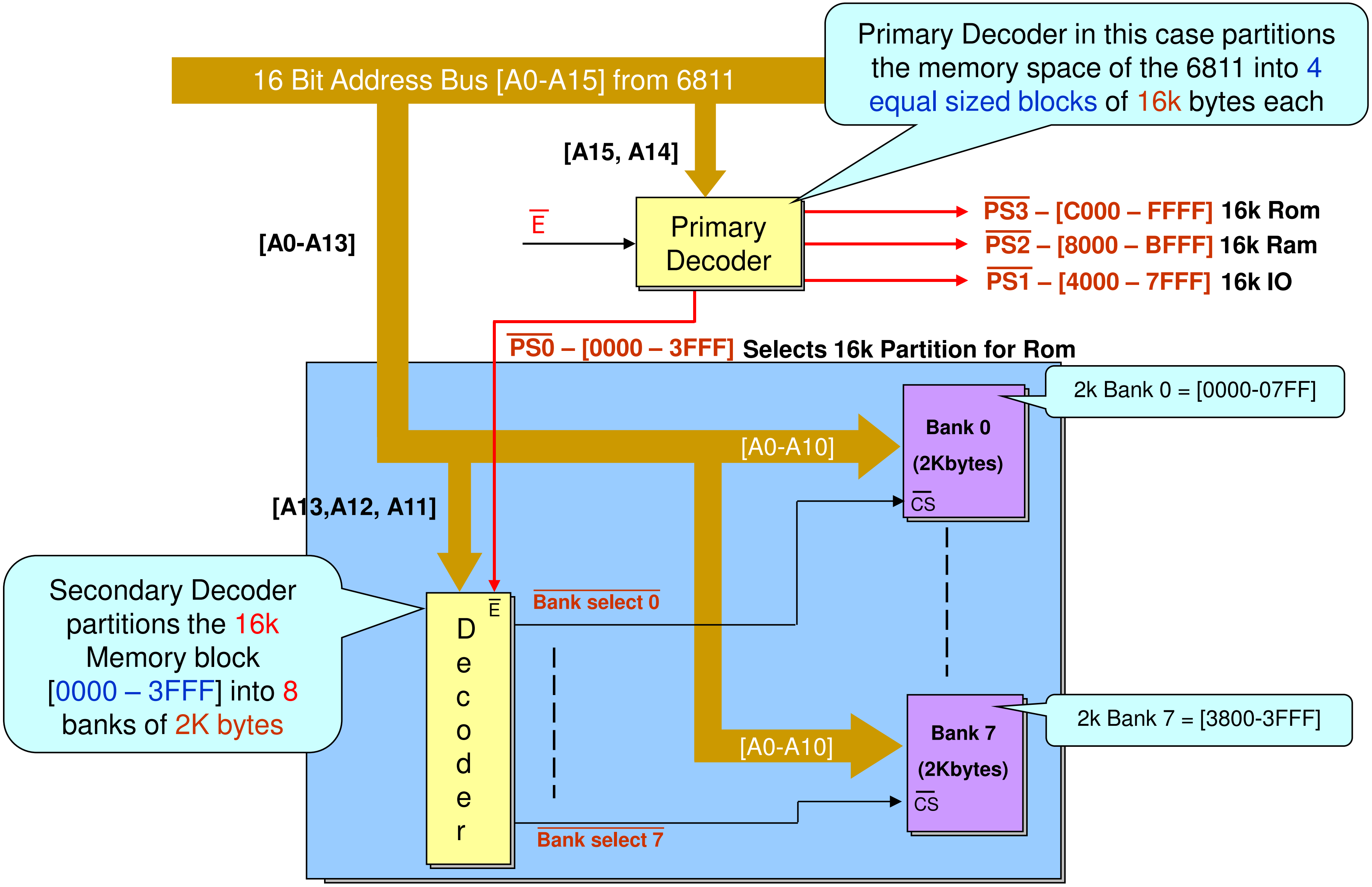
- In particular, we decode/partition the CPUs total address space such that

  - Part of the address issued by the CPU (*usually signified by the higher order address lines*) is used to partition the total memory space of the CPU into areas of say Sram, Dram, Flash or Memory Mapped Peripherals.

  - Within each partition, an additional sub-set of the *remaining* address lines may be further decoded to select individual Banks of memory chips within that partition, e.g. to partition the Sram area into 4 blocks of 64k Words each.

  - Finally whatever address bits are then left are decoded internally by the memory chip itself to select memory locations or registers within the chip.

Example Partitioning of 6811's 64k Memory Space



Primary Decoder in this case partitions the memory space of the 6811 into 4 equal sized blocks of 16k bytes each

Secondary Decoder partitions the 16k Memory block [0000 – 3FFF] into 8 banks of 2K bytes

16k Partition Needs 14 Address Lines [A0-A13]

Address Decoding Techniques

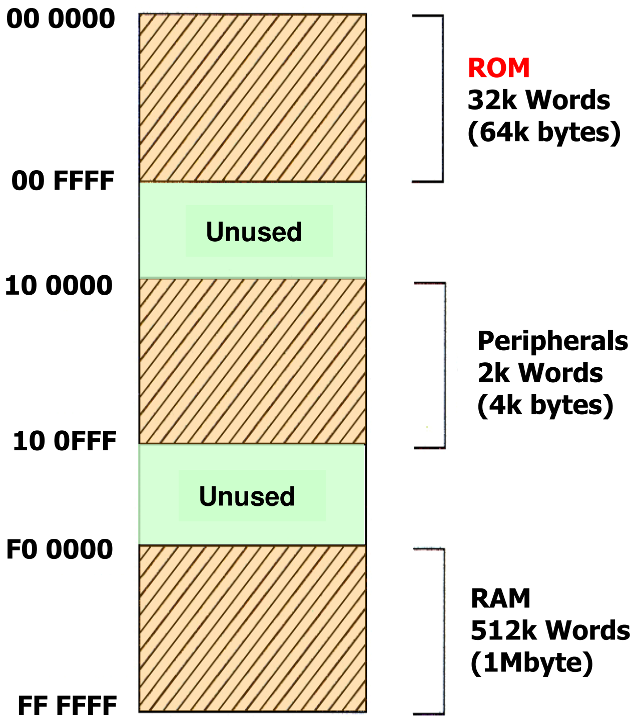
Designing a Memory Map

- Memory systems are usually partitioned into blocks of Rom, Ram and Peripherals. (*see opposite*)

- The only constraint that most CPUs impose on the designer is that there should be some Rom at say location 0 upwards to hold the reset exception vector table and Boot code

- Everything else is 'user definable'. No single memory map is necessarily any better than any other, they are just different.

- Some maps are designed to have holes in them to allow for expansion, some are designed with economy in mind, i.e. to use the minimum amount of logic etc. and some are just random !!



68000 Memory Map Partitioned into 3 Blocks or Rom, Ram, and Peripherals

Address Decoding Techniques

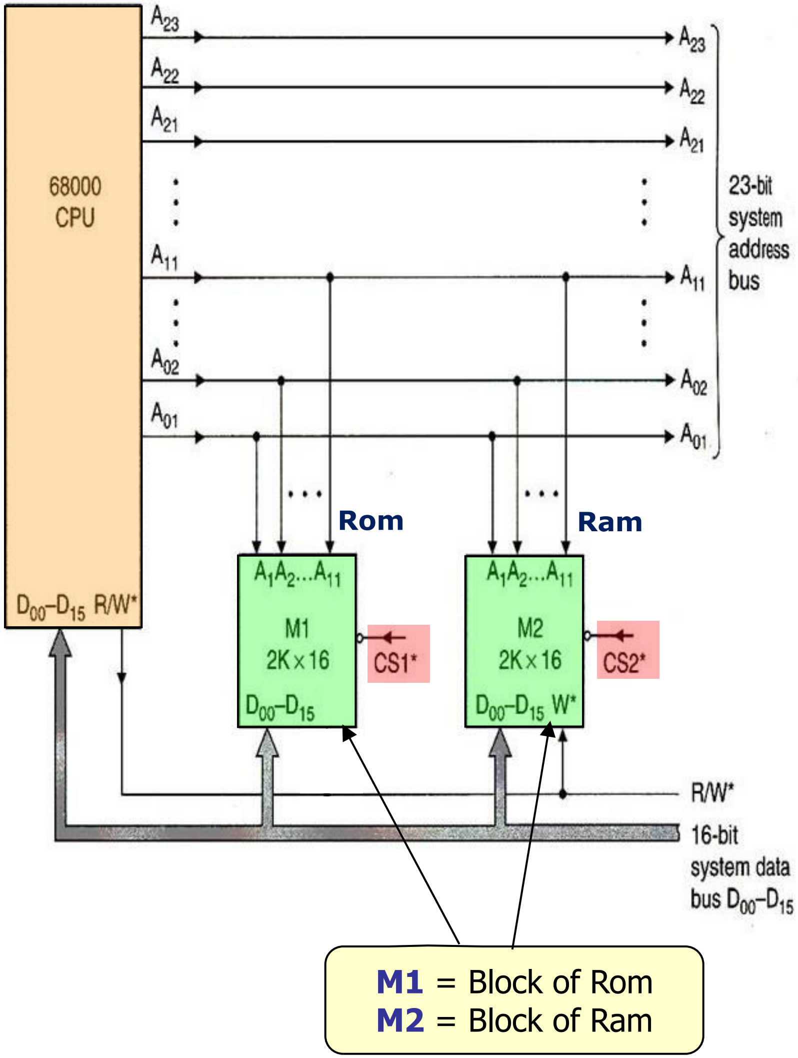
- Design an address decoder for a 68000 system with memory devices arranged as follows:

  - 2K Word Block of Rom: M1
  - 2K Word Block of Ram: M2

- Each block will need 11 address lines connected to the lowest address lines issued by the CPU: **A1-A11**.

- Our task is to design the decoder that will generate the chip selects (CS1* and CS2*) for each block so that only one block responds to each address.

- We get to chose where to position the block in the memory map.That is, what range of CPU addresses the block responds to.



M1 = Block of Rom
M2 = Block of Ram

Full Address Decoding

Address Decoding Techniques

- A system employs Full address decoding, when the decoder circuit utilises all the unused address lines that are not directly connected to the memory chips.

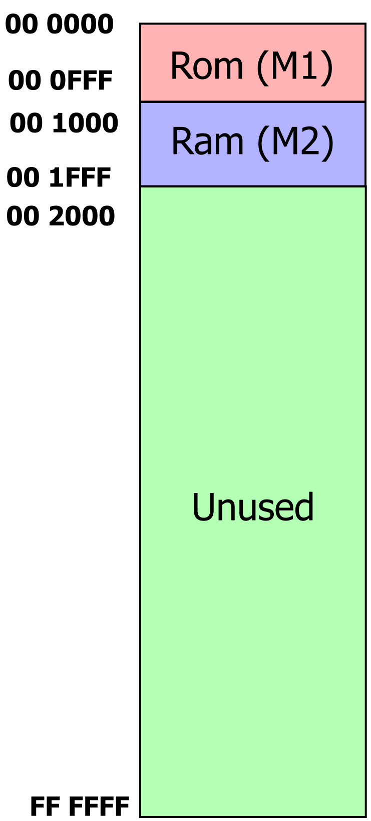
- In our previous example, a full address decoder would decode A12-A23 (*for original 68000, A12-A31 for soft core 68000 used in 465*).

- Let's design a decoder to place :-

  - Rom (M1) at address [00 0000 – 00 0FFF]
  - Ram (M2) at address [00 1000 – 00 1FFF]

- This arrangement will give us a total of 4KWords of contiguous memory with no gaps and no overlaps and thus the memory map will look like that shown opposite.

Memory Map



Address Decoding Techniques

How do we design the circuit to realise this map?

- **Step1:** Create an address table with all CPU address lines listed as columns. In the case of the (*original*) 68000 this will be A23-A1.

- **Step 2:** Create a Row entry for each memory block that you are designing a chip-select signal for: M1 and M2.

- **Step 3:** Decide how many of the CPU address lines will connect directly to the memory blocks and place an 'X' meaning "don't care" under those address lines in the table because these address lines will be decoded by the memory chips/blocks themselves.

  Because we are using 2K word addresses, each Rom/Ram block will utilise 11 addresses lines ($2^{11} = 2048$), this leaves A12-A23 to be decoded.

Note no A0

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
M1													x	x	x	x	x	x	x	x	x	x	x	Rom[000000-000FFF]
M2													x	x	x	x	x	x	x	x	x	x	x	Ram[001000-001FFF]

Address Decoding Techniques

- **Step 4:** Use the remaining address bits in the table to chose the combination of 0's and 1's on the address lines that will select the device at the locations required, taking care to make sure no two devices ever respond to the same address.

- Our full address decoder design thus decodes [A12-A23]

- Notice how a difference in A12 is what effectively differentiates an address intended for M1, from M2. Thus
M1 is addressed when [A23 - A13] = 0 and [A12] = 0
M2 is addressed when [A23 - A13] = 0 and [A12] = 1

- These combinations of address lines are mutually exclusive conditions thus only one device can be selected at a time.

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
M1	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	Rom[000000-000FFF]
M2	0	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	Ram[001000-001FFF]

Note no A0

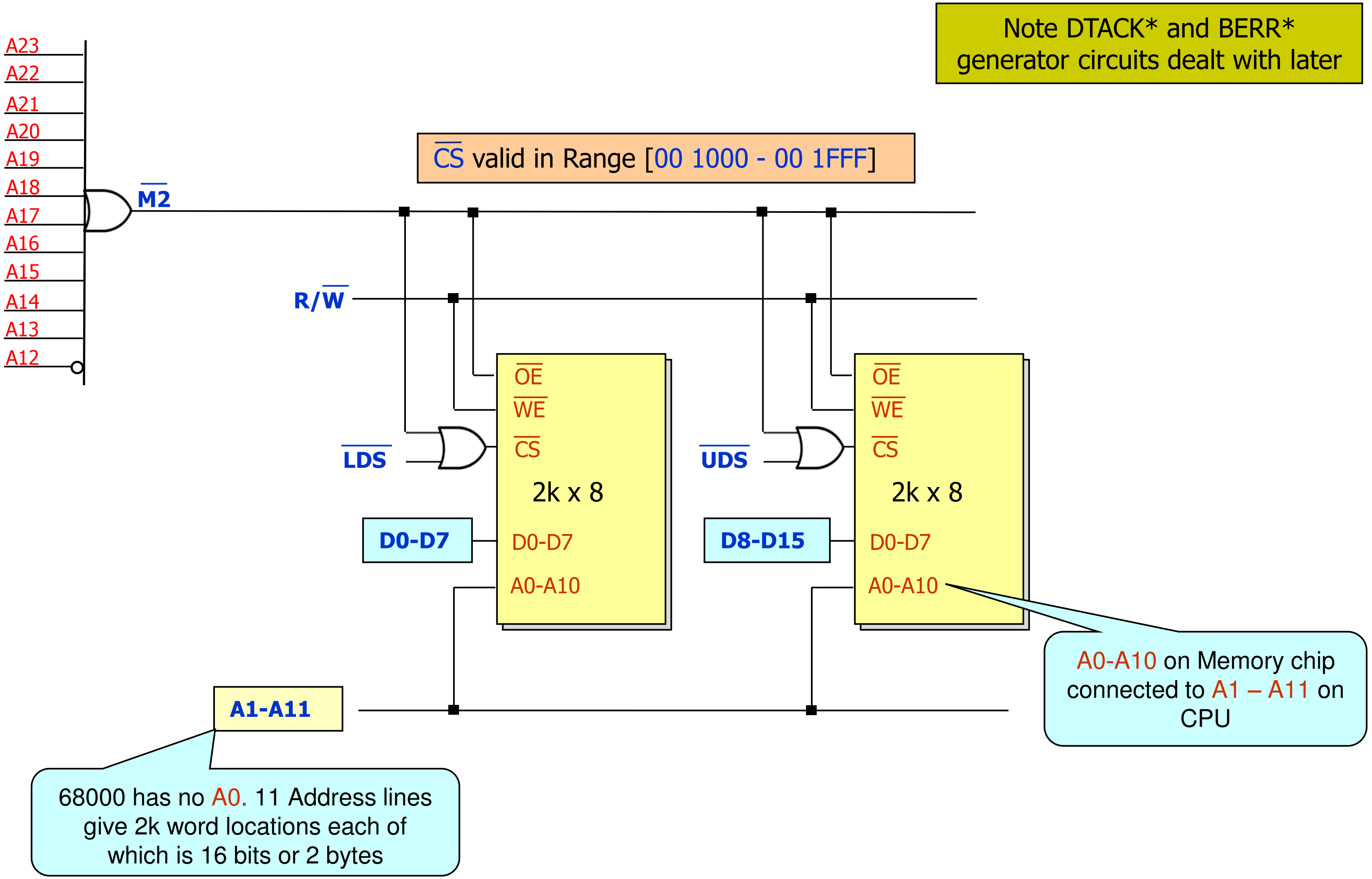
M1 = A23 + A22 + A21 + …. + A12
M2 = A23 + A22 + A21 + …. + (!A12)

Active low 'memory select' signals

Address Decoding Techniques

Note DTACK* and BERR* generator circuits dealt with later

$\overline{CS}$ valid in Range [00 1000 - 00 1FFF]



A0-A10 on Memory chip connected to A1 – A11 on CPU

68000 has no A0. 11 Address lines give 2k word locations each of which is 16 bits or 2 bytes

Two 2k byte devices working in parallel to give 2k words (or 4K bytes)

Address Decoding Techniques

A More Practical Example

- A 68000 system requires the following arrangement of Rom, Ram and Peripheral chips:

> - 10K word block of Rom arranged as
>   - 2K words (Rom1)
>   - 8K words (Rom2)
> - 2K word block of Ram (Ram1)
> - 2 Individual words for Peripheral 1 (Peri1)
> - 2 Individual words for Peripheral 2 (Peri2)

- You are free to assign the blocks any addresses you like with the exception of Rom1 which must reside at a base addresses of 0.

- Design a full address decoder to realise this memory map.

Address Decoding Techniques

- **Step 1:** Map each chip to an address space in the memory map of the system. The only constraint here is that Rom1 should reside at base address 0. The allocation of other block addresses is fairly arbitrary so the memory map below has been chosen.

Rom1 =	[00 0000 - 00 0FFF]	2 K Words	(4 Kbyte range)
Ram1 =	[00 1000 - 00 1FFF]	2 K Words	(4 Kbyte range)
Rom2 =	[00 4000 - 00 7FFF]	8 K Words	(16 Kbyte range)
Peri1 =	[00 8000 - 00 8003]	2 Words	(4 byte range)
Peri2 =	[00 8004 - 00 8007]	2 Words	(4 byte range)

- These addresses were carefully chosen so that there is no overlap or sharing of addresses between the blocks. Many other memory maps could have been chosen.

- For example, Rom2 could have been chosen to be **contiguous** with Rom 1 i.e. from address [00 1000 - 00 8FFF] but it wasn't because the decoder for an 8k block is easier to design if we assign the block a start address corresponding to that of a natural 8k boundary in the memory (*we would need 4 separate 2k decoders OR'd together if we made* Rom2 *sit in a 2k boundary*). This is why Rom2 was chosen to start at address [00 4000].

- **Step 2:** Decide how many Address lines each block will require governed by its size

Rom1 is a 2K word block requiring 11 address lines (i.e. $2^{11} = 2048$).

Ram1 is also a 2K word block also requiring 11 address lines.

Rom2 is an 8K word block requiring 13 address lines (i.e. $2^{13} = 8192$).

Peri1 is a 2 word peripheral requiring 1 address line.

Peri2 is a 2 word peripheral requiring 1 address line.

Address Decoding Techniques

- **Step 3:** Create an address table for your memory map. Place an ‘x’ in the row entry for each chip beneath the address lines that connect directly to the chip.

Note no A0, LDS* and UDS* select individual bytes within a word

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
000000-000FFF	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	Rom1
001000-001FFF	0	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	Ram1
004000-007FFF	0	0	0	0	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	Rom2
008000-008003	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	x	Peri1
008004-008007	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	x	Peri2

- From this address table we determine which combination of 0’s and 1’s on the unused address lines for each chip are used by the decoder to provide a chip select signal.

- We can thus see that the Boolean expression for the active low block select signals is given as

Write VHDL for these decoders

Rom1 = A23 + A22 + A21 + …. + A12
Ram1 = A23 + A22 + A21 + …. + (!A12)
Rom2 = A23 + A22 + A21 + …. + A15 + (!A14)
Peri1 = A23 + A22 + A21 + …. + A16 + (!A15) + A14 + A13 + …. + A3 + A2
Peri2 = A23 + A22 + A21 + …. + A16 + (!A15) + A14 + A13 + …. + A3 + (!A2)