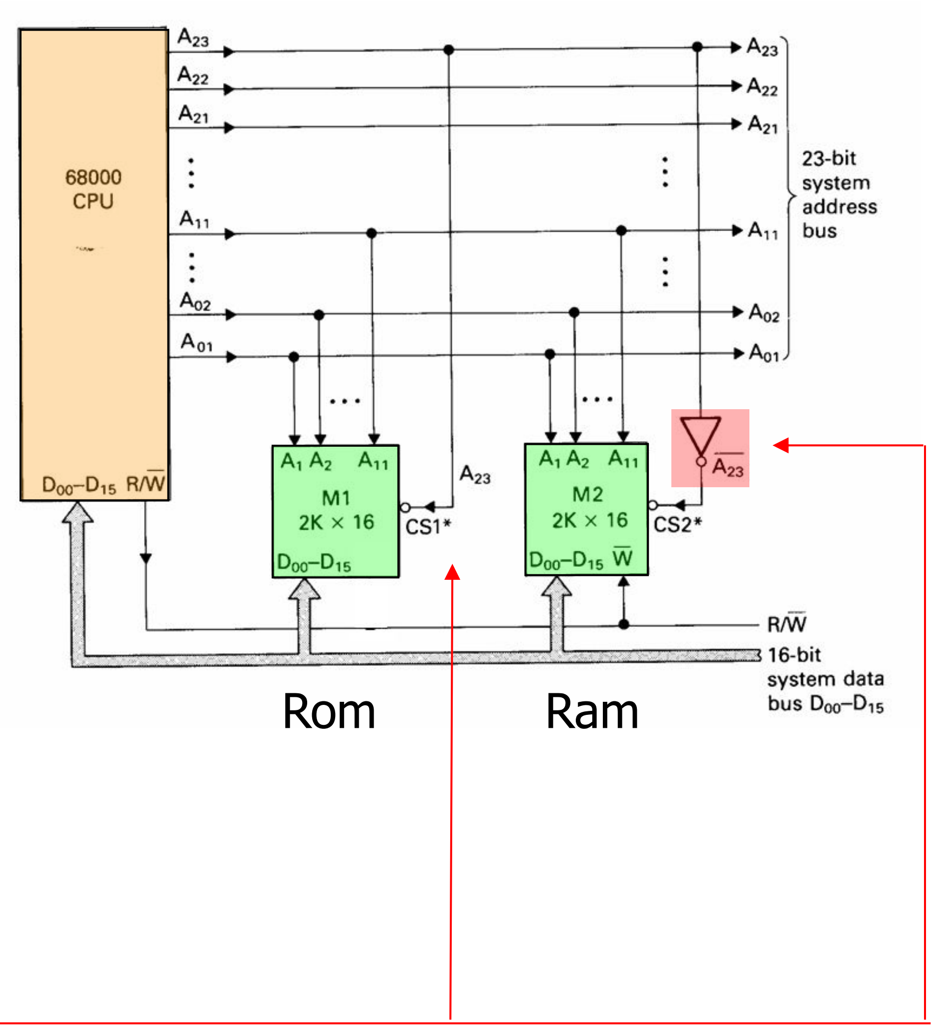
Address Decoding Techniques

Partial Address Decoding

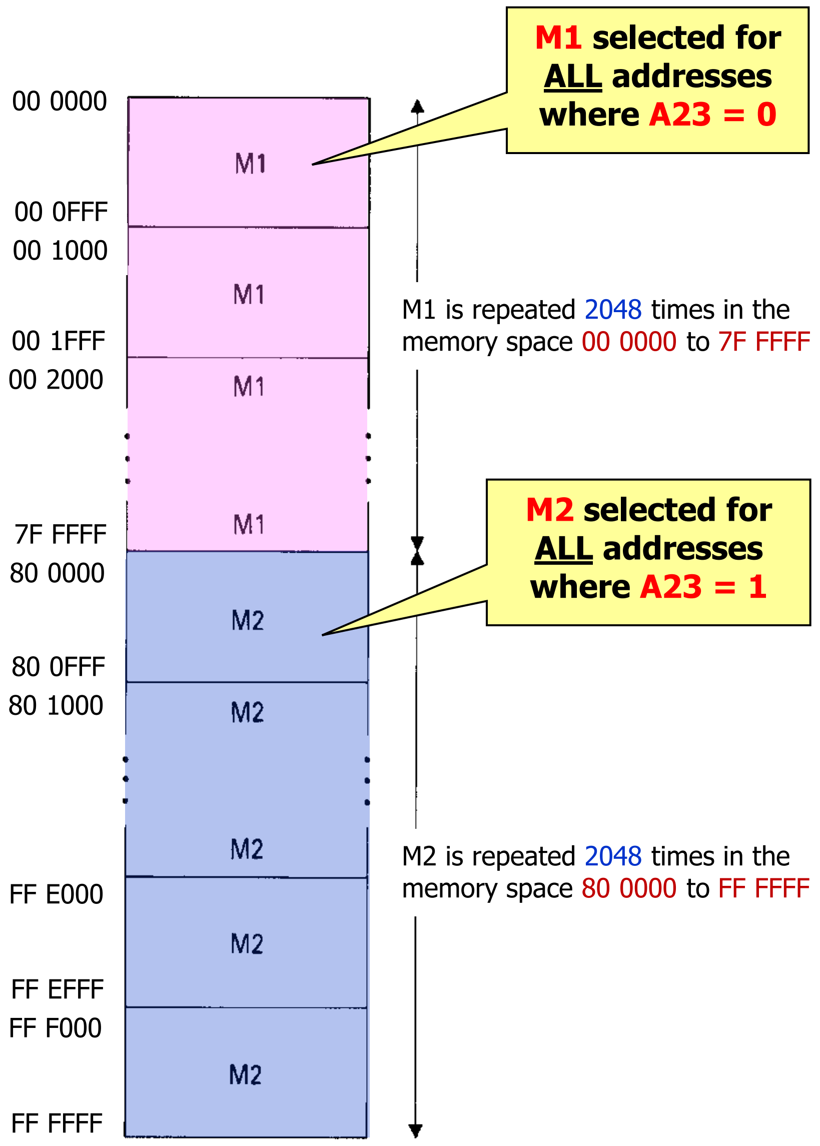
- Here only a sub-set of the unused address lines are employed by the decoder circuit.
- Partial decoding can lead to a reduction in
  - Complexity of decoder (fewer gates)
  - Propagation delay of the decoder
  - Power consumption (fewer gates)
  - PCB/chip "real-estate" and cost
- Partial address decoders are particularly attractive when used with CPUs having a large address space i.e. 32/64 bit address
- Going back to our very 1st example of a memory system, (Lecture 3) we could design a simple partial address decoder for a system with 2K words of Rom and Ram like this.
- With only two blocks to decode (Rom + Ram) we should only need to decode 1 address line e.g. A23 could be used



- We could arrange for M1 (Rom) to be enabled when A23 = 0 and M2 (Ram) to be enabled when A23 = 1 thus both devices can <u>never</u> be selected at the same time. It also means there is still ROM located at address 0 (a requirement for the 68000).

Address Decoding Techniques

- However, there are drawbacks to partial decoding, for example take the circuit just considered.
- Because only A23 was used by the address decoder, the other 11 address lines (A22 - A12) play no part in the decoding process and thus act as 'don't cares' to the decoder.
- The effect of this is that both 2k word memory blocks get '**mirrored**', in the memory map.
- That is, each block appears to the CPU to be **replicated** 2048 times and appears as 4M words. This is because A22 – A12 plays no part (i.e. 11 address lines) and 2^{11} = 2048
- This would certainly confuse a booting operating system searching for installed memory.
- Here the OS might conclude that the system has 8M bytes of Ram instead of the 4K bytes it really has which is *devastating* to say the least.


00 0000
00 0FFF
00 1000
00 1FFF
00 2000
7F FFFF
80 0000
80 0FFF
80 1000
FF E000
FF EFFF
FF F000
FF FFFF
M1
M1
M1
M1
M2
M2
M2
M2
M2
M1 selected for ALL addresses where A23 = 0
M2 selected for ALL addresses where A23 = 1
M1 is repeated 2048 times in the memory space 00 0000 to 7F FFFF
M2 is repeated 2048 times in the memory space 80 0000 to FF FFFF

Partial Decoding: A more practical example

- Let's go back to the problem of the simple Rom, Ram and Peripheral mapping problem from lecture 3. That system comprised of the following

  - 10K words of Rom arranged as
    - A block of 2K (Rom1) plus
    - A block of 8K (Rom2)
  - 2K words of Ram (Ram1)
  - 2 individual words for Peripheral 1 (Peri1)
  - 2 individual words for Peripheral 2 (Peri2)

- The address table for a **FULL** address decoder scheme is repeated below for the sake of comparison and uses many gates to fully decode all unused address lines. Let's look at a **partial** decoding scheme.

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

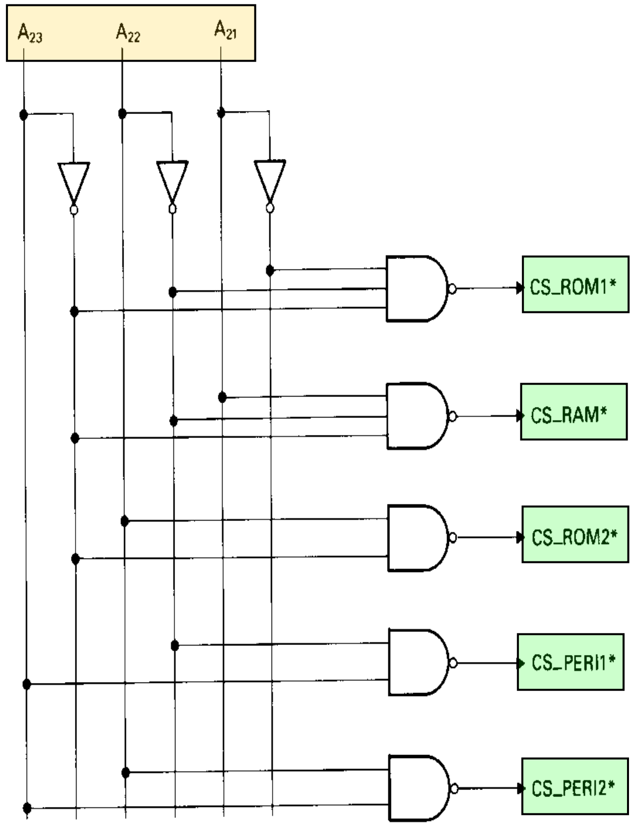
Solution using Partial Decoding

- With 5 chip select signals to produce, in *theory* only 3 address lines should be needed by our decoder ($2^3 > 5$).

- One possible address table for a simple memory map is illustrated below where only the 3 highest address lines are decoded (*it's common to use the highest address lines unless a specific base address is req'd*).

  (***Note****: this address map is different to the original full address decoder from lecture 3 and has been chosen to maximise the advantages of partial address decoding, however Rom1 is still at address 0*).

- What is address range and size (*including mirroring*) of each block?



Logic Equations
Rom1 CS* = !(!A23 . !A22 . !A21)
Ram1 CS* = !(!A23 . !A22 . A21)
Rom2 CS* = !(!A23 . A22)
Peri1 CS* = !(A23 . !A22)
Peri1 CS* = !(A23 . A22)

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
	0	0	0	ignored									x	x	x	x	x	x	x	x	x	x	x	Rom1
	0	0	1	ignored									x	x	x	x	x	x	x	x	x	x	x	Ram
	0	1	x	ignored							x	x	x	x	x	x	x	x	x	x	x	x	x	Rom2
	1	0	x	ignored																			x	Peri1
	1	1	x	ignored																			x	Peri2

Address Decoding Techniques

Exercise 11

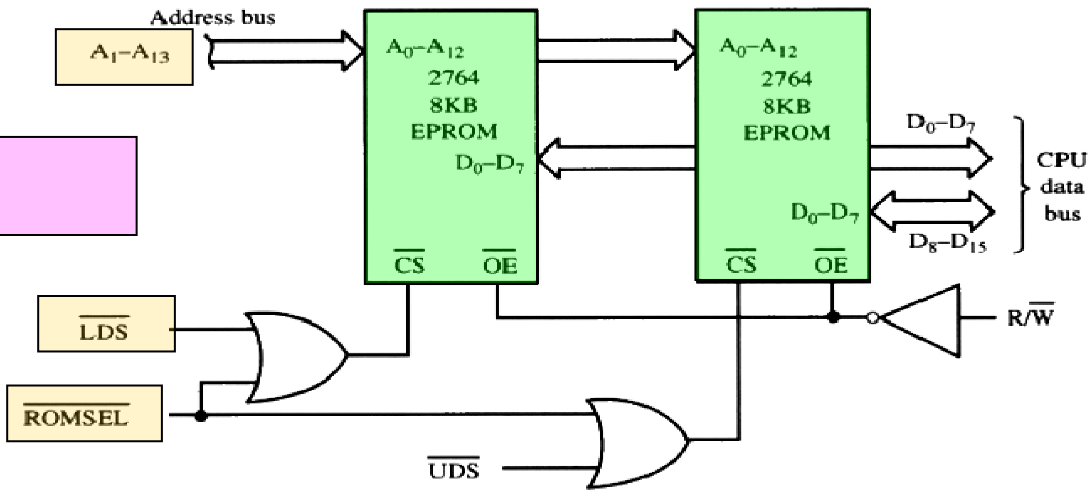
- Design a Partial address decoder for a 68000 based system with 8K Words of EPROM space, so that it responds to a base address of $00 8000, using 8K byte memory chips.

Solution 11

- Two chips are needed to give 8k x 16. Each chip will use 13 address lines, (2^{13} = 8192) a0-a12 connected to A1-A13 on the CPU leaving potentially A14-A23 to feed the decoder.
- A Base address of $00 8000 = binary [0000,0000,10xx,xxxx,xxxx,xxxx], i.e. when A15 = logic 1
- An address table is given below, along with a sample circuit.
- As a minimum solution to this problem, A15 could be used as our $\overline{\text{ROMSEL}}$ signal as this would mean the device *definitely* responds to address $00 8000 but also to a lot of other addresses too.
- We could reduce the extent of mirroring and free up more space in the memory map for future expansion etc. if we just included A23-A20 = 0 into the decoding logic. The ROM would still respond to the base address required but it might not "start" or be limited to that address.

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Rom

Logic Equations
ROMSEL = !(!A23 . !A22 . !A21 . !A20)



Address Decoding Techniques

Exercise 12

- **Design a partial address decoder for a 68000 based system that contains**
  - 2M bytes of Eprom starting at $00 0000 using 512k byte chips.
  - 2M bytes of Ram starting at $20 0000 using 256k byte chips.
  - 64k bytes of I/O space starting at $80 0000.

- **Solution 12**
  - For the EPROM we will need 4, 512k byte chips, organized as 2 banks of 512k words (512k = 19 address lines) or 1M byte.
  - For the RAM we will need 8, 256k byte chips, organized as 4 banks of 256k words: RAM1 to RAM4 (256k = 18 address lines).
  - The 64k IO Space = 32k words at $80 0000 = 15 lines
  - The address table is given below, while the logic equations for a simple partial decoder circuit is given alongside.
  - The equations have been carefully chosen to fix the devices at the addresses specified and to ensure no overlap and to make banks contiguous within a block.

Logic Equations

Eprom1 = !(!A23 . !A21 . !A20)
Eprom2 = !(!A23 . !A21 . A20)

Ram1 = !(!A23 . A21 . !A20 . !A19)
Ram2 = !(!A23 . A21 . !A20 . A19)
Ram3 = !(!A23 . A21 . A20 . !A19)
Ram4 = !(!A23 . A21 . A20 . A19)

IO = !(A23)

Address	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Eprom (Bank1)
	0	x	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Eprom (Bank2)
	0	x	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Ram1 (Bank1)
	0	x	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Ram2 (Bank2)
	0	x	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Ram3 (Bank3)
	0	x	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Ram4 (Bank4)
	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	IO

Address Decoding Techniques

Block Address Decoding

- Block decoding partitions the CPU address space into equal sized blocks and can be used with both full and partial address decoding schemes.

- The resultant block size is dependent upon those address lines used by the block decoder.

- For example we could partition an address space of 64K into

  - 4 blocks of 16k, or
  - 8 blocks of 8k or
  - 16 blocks of 4k etc.

- Once memory has been partitioned into blocks, full, partial or additional block decoders can subsequently be employed to refine and resolve the address map into smaller and smaller blocks.

- For example, in the case of a 64k system partitioned into 4 blocks of 16k, we could use additional block decoders to partition this into

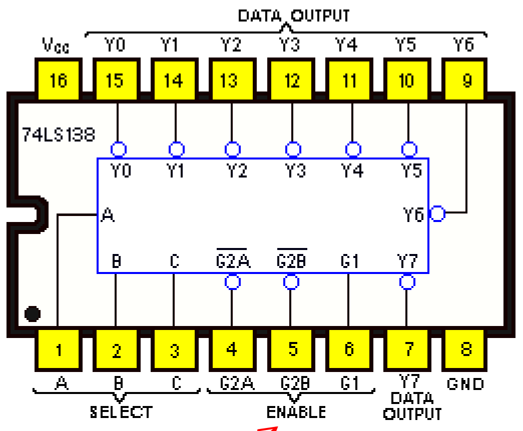
  - 4 blocks of 4k, or
  - 8 blocks of 2k or
  - 16 blocks of 1k etc.

Address Decoding Techniques

Block Decoders

- Fortunately for us, devices exist, often in the form of a single chip, to make block decoding solutions easy for us, or we can use VHDL for programmable logic.
- For example the 74138 is a simple 3 to 8 line decoder, giving us the ability to partition memory into 8 equal sized blocks.
- A Pin Out and Truth Table for this device is illustrated below. It has 3 inputs (which could be fed to address lines), 8 outputs for 8 individual equal sized block select signals and 3 enable signal (*for further refinement of address range*).

INPUTS						OUTPUTS							
G_1	$\overline{G_{2A}}$	$\overline{G_{2B}}$	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0



2 active low plus 1 active high enable

Example VHDL code for a simple 3-8 Decoder

```
LIBRARY       IEEE ;
USE           IEEE . STD_LOGIC_1164 . ALL ;
------------------------------------------------------------------------------------------------------------------------------------
-- 3 TO 8 Line Decoder
------------------------------------------------------------------------------------------------------------------------------------
ENTITY  Decoder3to8Line  IS
      PORT(   CBA           : IN  STD_LOGIC_VECTOR ( 2  downto  0 ) ;          -- 3 bit input select bus
              G1, G2A, G2B  : IN  STD_LOGIC ;                                  -- 3 enable inputs
              Y7_Y0         : OUT STD_LOGIC_VECTOR ( 7  downto  0 )            -- 8 active low outputs as a bus
      ) ;
END ;
------------------------------------------------------------------------------------------------------------------------------------
ARCHITECTURE  Behavioural  OF  Decoder3to8Line  IS
BEGIN
   PROCESS( CBA ,  G1 ,  G2A ,  G2B)
   BEGIN
         IF ( ( G1 = '1' ) AND  ( G2A = '0' )  AND  ( G2B  =  '0' ) ) THEN       -- if device enabled
              IF     ( CBA = "000" )  THEN  Y7_Y0  <=  "11111110" ;              -- if  CBA = 000 set y0 to 0
              ELSIF ( CBA = "001" )  THEN  Y7_Y0  <=  "11111101" ;               -- else if CBA = 001 set y1 to 0
              ELSIF ( CBA = "010" )  THEN  Y7_Y0  <=  "11111011" ;               -- else if CBA = 010 set y2 to 0
              ELSIF ( CBA = "011" )  THEN  Y7_Y0  <=  "11110111" ;               -- else if CBA = 011 set y3 to 0
              ELSIF ( CBA = "100" )  THEN  Y7_Y0  <=  "11101111" ;               -- else if CBA = 100 set y4 to 0
              ELSIF ( CBA = "101" )  THEN  Y7_Y0  <=  "11011111" ;               -- else if CBA = 101 set y5 to 0
              ELSIF ( CBA = "110" )  THEN  Y7_Y0  <=  "10111111" ;               -- else if CBA = 110 set y6 to 0
              ELSE                         Y7_Y0  <=  "01111111" ;               -- else if CBA = 111 set y7 to 0
              END IF;
         ELSE                                                                    -- if not enabled
              Y7_Y0 <= "11111111" ;
         END IF ;
   END PROCESS;
END ;
```

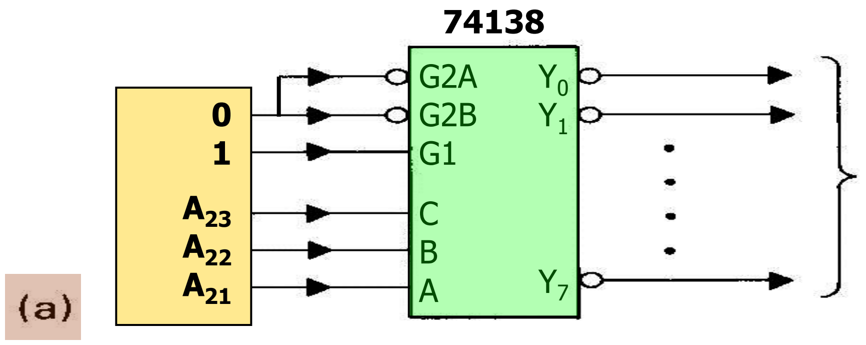
Note: Y7-Y0 assigned a value for each path in the process above. Result: combinatorial logic with <u>no</u> storage/latches

Address Decoding Techniques

Simple Block decoder schemes using a 74138

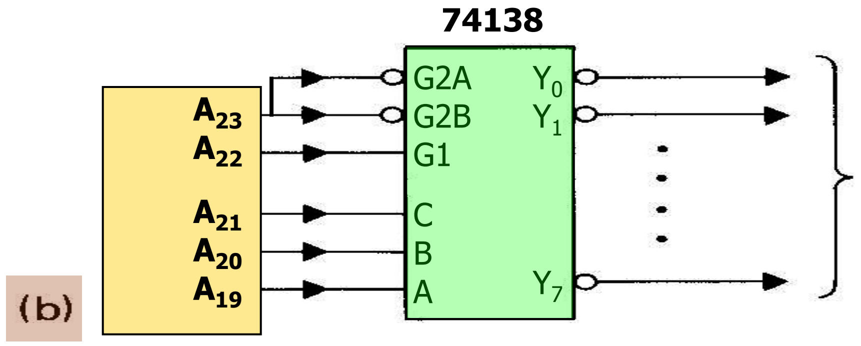
- Circuit **a)** below is used to decode the upper 3 address lines of the 68000 address bus to provide 8 blocks of 2M bytes, i.e. address blocks in the range hex 00 0000 – 1F FFFF, 20 000 - 3F FFFF etc.

- Circuit **b)** makes use of the enable lines to assist in the decoding of 8 blocks in the lower quarter of the address space (A23,A22 = [0,1]).



8 Blocks of 2M bytes in range
Y_0 range = [00 0000 – 1F FFFF]
Y_1 range = [20 0000 – 3F FFFF]
Y_2 range = [40 0000 – 5F FFFF]
Y_3 range = [60 0000 – 7F FFFF]

Y_4 range = [80 0000 – 9F FFFF]
Y_5 range = [A0 0000 – BF FFFF]
Y_6 range = [C0 0000 – DF FFFF]
Y_7 range = [E0 0000 – FF FFFF]



8 Blocks of 512k bytes in range
Y_0 range = [40 0000 – 47 FFFF]
Y_1 range = [48 0000 – 4F FFFF]
Y_2 range = [50 0000 – 57 FFFF]
Y_3 range = [58 0000 – 5F FFFF]

Y_4 range = [60 0000 – 67 FFFF]
Y_5 range = [68 0000 – 6F FFFF]
Y_6 range = [70 0000 – 77 FFFF]
Y_7 range = [78 0000 – 7F FFFF]

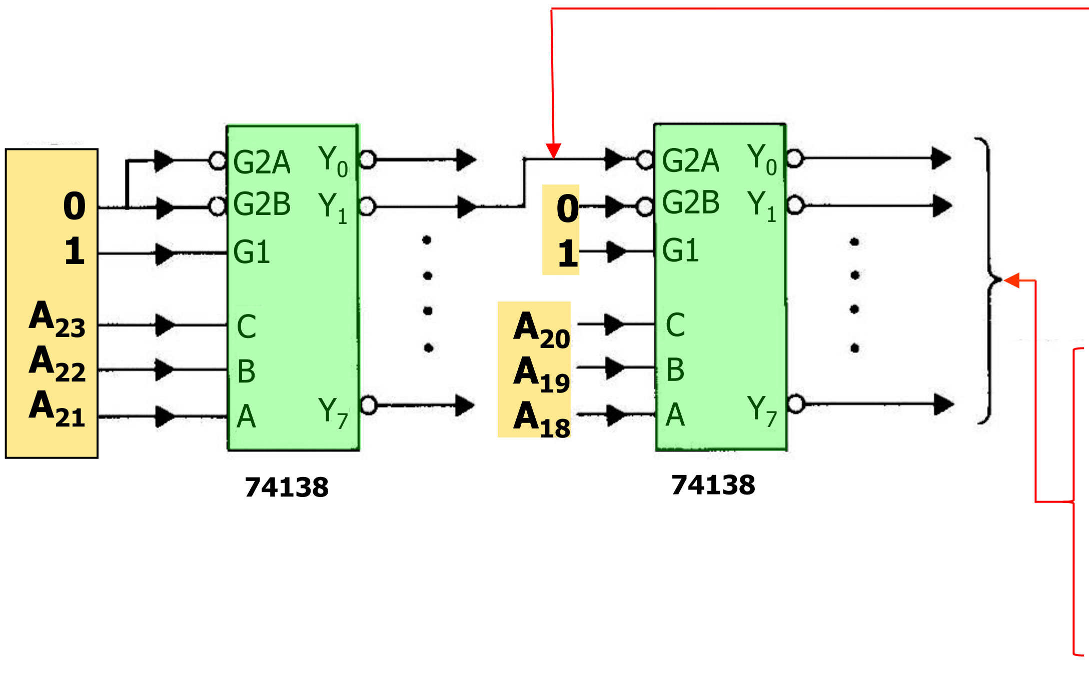
Address Decoding Techniques

More Block decoder schemes using 74138 (cont…)

- This uses two cascaded 74138 devices to provide more refined block decoding. A23-A21 feed into the 1st device which provides block outputs of 2M bytes (as before).

  The 2nd output from this device (Y1) which is enabled in the address range [20 0000-3F FFFF] is then used to enable a 2nd cascaded device fed with address lines A20-18.

  This 2nd device decodes the address 20 0000-3F FFFF into 8 smaller blocks of 256k words in the range 20 0000-23 FFFF, 24 0000-27 FFFF etc.



8 Blocks of 2M bytes in range

Y_0 range = [00 0000 – 1F FFFF]
Y_1 range = [20 0000 – 3F FFFF]
Y_2 range = [40 0000 – 5F FFFF]
Y_3 range = [60 0000 – 7F FFFF]

Y_4 range = [80 0000 – 9F FFFF]
Y_5 range = [A0 0000 – BF FFFF]
Y_6 range = [C0 0000 – DF FFFF]
Y_7 range = [E0 0000 – FF FFFF]

8 Blocks of 256k bytes in range

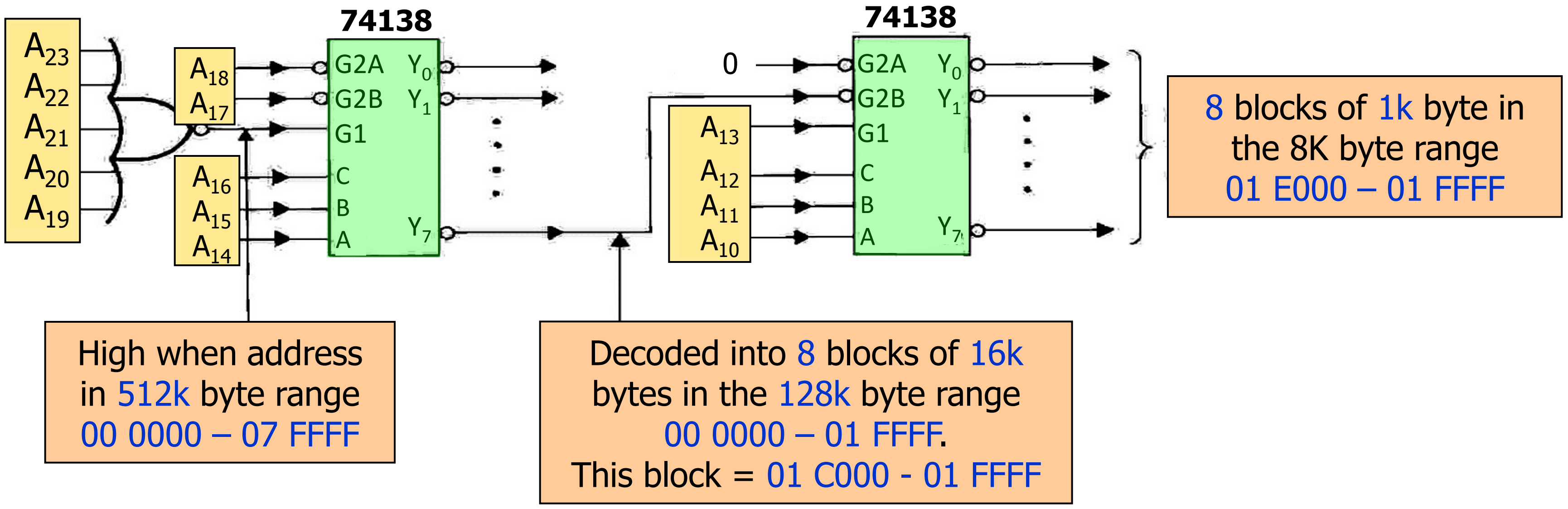
Y_0 range = [20 0000 – 23 FFFF]
Y_1 range = [24 0000 – 27 FFFF]
Y_2 range = [28 0000 – 2B FFFF]
Y_3 range = [2C 0000 – 2F FFFF]

Y_4 range = [30 0000 – 33 FFFF]
Y_5 range = [34 0000 – 37 FFFF]
Y_6 range = [38 0000 – 3B FFFF]
Y_7 range = [3C 0000 – 3F FFFF]

Address Decoding Techniques

Sample Block decoder schemes using 74138 (cont…)

- Finally, the circuit below, provides very fine grained address decoder resolution to provide 8 blocks of 1K byte in the address range 01 E000 – 01 FFFF.
- Such a scheme might be employed to select 1 of 8 possible small I/O devices within the system and limit their address space to a very small part of the overall memory map of the system.

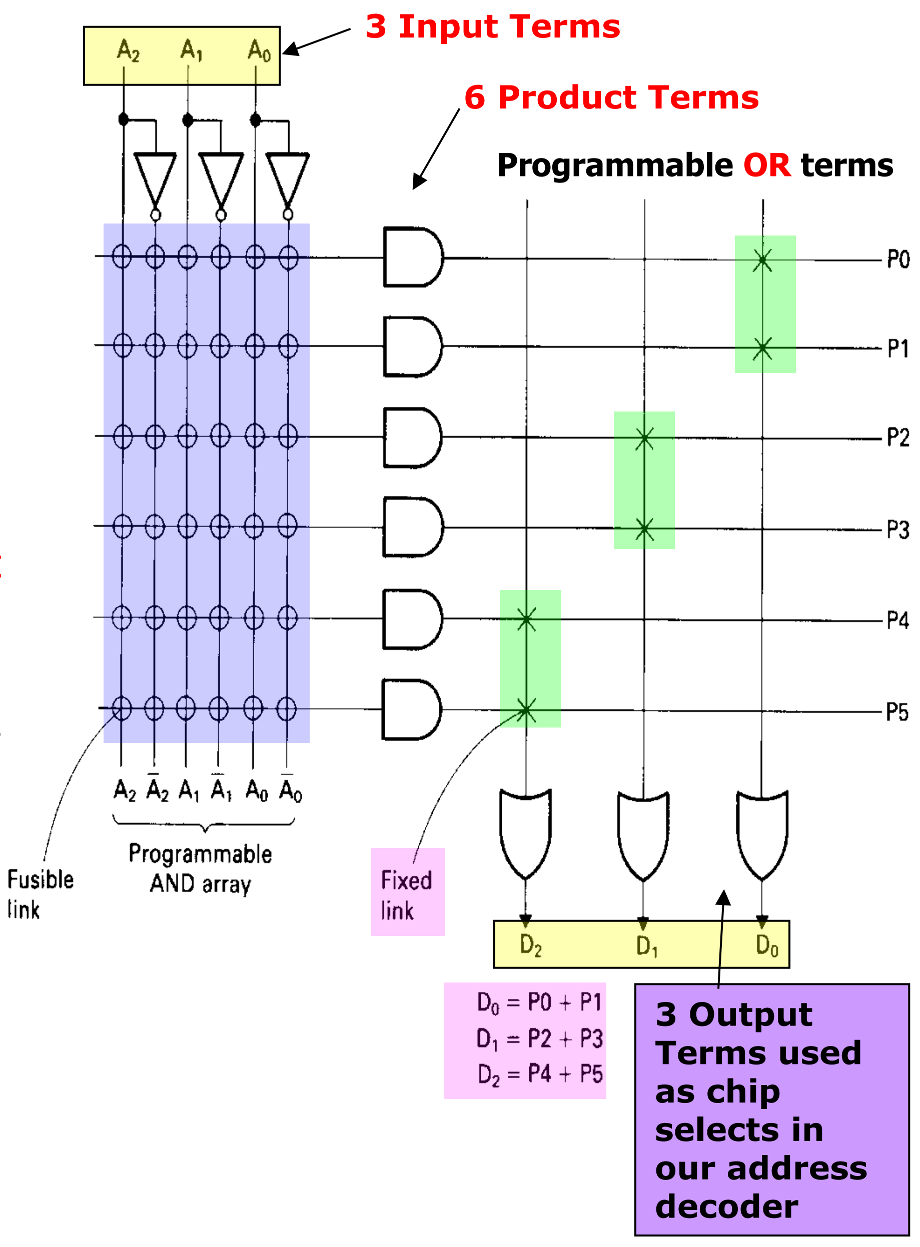


74138
74138
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
G2A
G2B
G1
C
B
A
Y0
Y1
Y7
0
A13
A12
A11
A10
High when address in 512k byte range 00 0000 – 07 FFFF
Decoded into 8 blocks of 16k bytes in the 128k byte range 00 0000 – 01 FFFF. This block = 01 C000 - 01 FFFF
8 blocks of 1k byte in the 8K byte range 01 E000 – 01 FFFF

Address Decoding Techniques

Address Decoding Using Programmable logic

- Simple programmable logic has an architecture ideally suited to implementing a sum-of-products (SOP) expression.
- Given that any decoder can be expressed in SOP form it follows that we can program a GAL to do any address decoding function we like provided it has enough inputs and outputs
- The illustration here demonstrates the principle with a simplified section of a device.
- This device has 3 input terms, [a0,a1,a2] which are available in their normal and inverted forms.
- The diagram illustrates a device with 6 product terms, i.e. 6 AND gates connected via 'fuses' to any combination of the 3 input terms, thus each AND gate has 6 inputs and 6 fuses.
- Designing address decoders with PALs involves generating the Boolean logic equations from the address table and 'generating a fuse map' (JEDEC file) which can be downloaded to a Prom programmer to 'blow' the device.

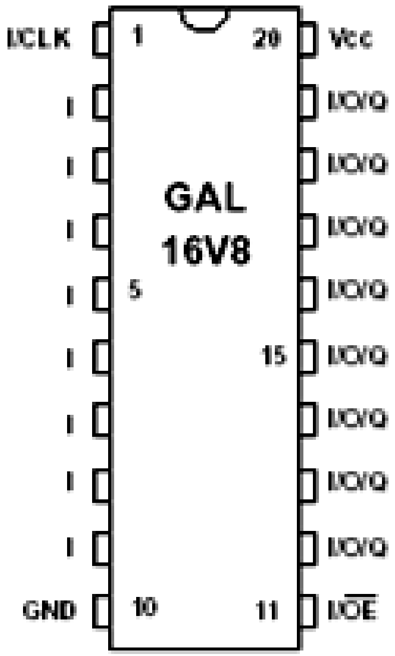


$D_0 = P0 + P1$
$D_1 = P2 + P3$
$D_2 = P4 + P5$

Address Decoding Techniques

GAL Example: GAL16V8

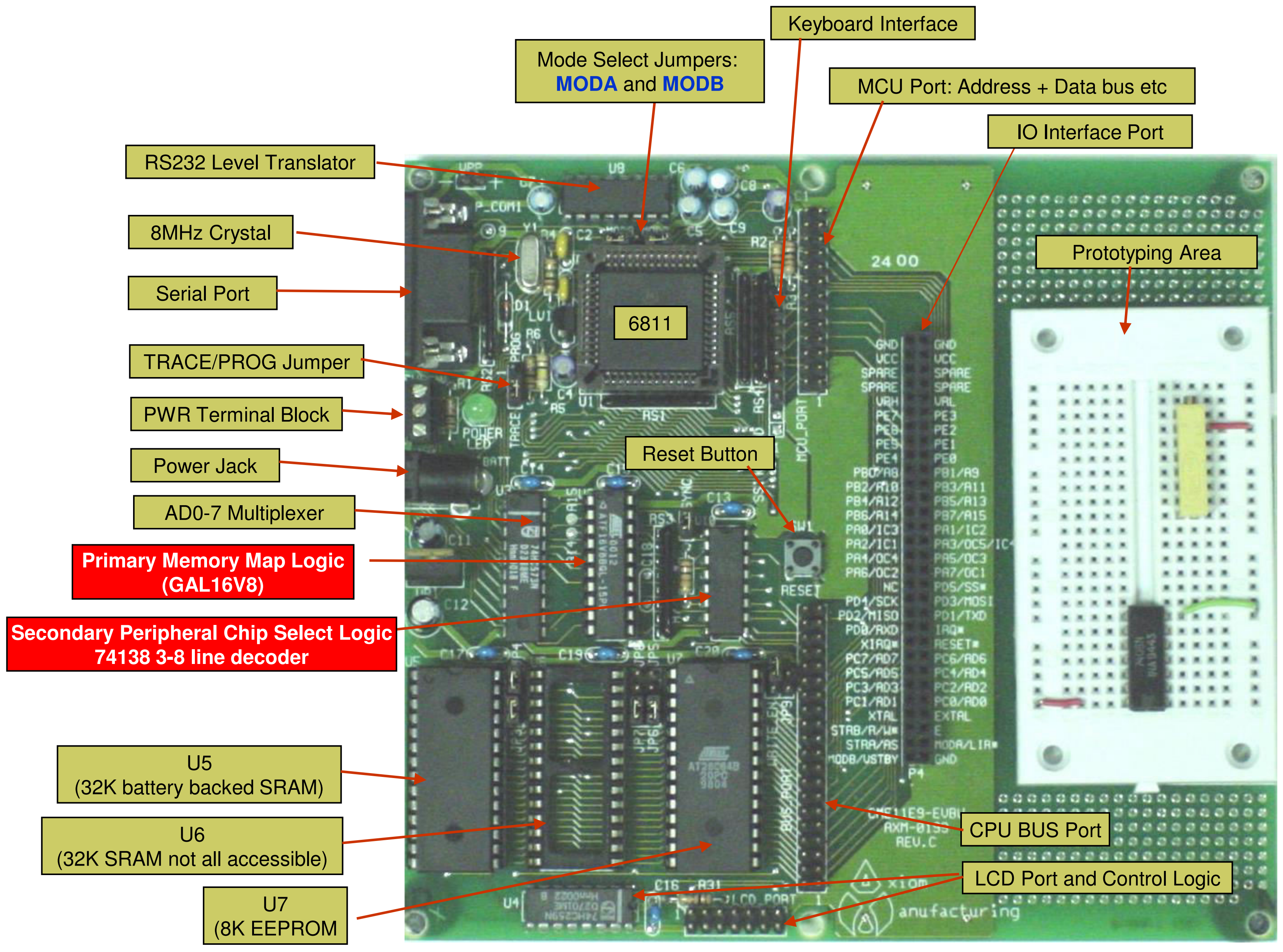
- The Pin out for the Lattice GAL16V8 is shown below:



The pins can be grouped as follows:

Pins	Signal Names
1	Clock input (not used for decoding)
2-9	Inputs: Address line from CPU
10	Ground (0V)
11	Output Enable
12-19	Outputs: Chips selects
20	Power (5V)

- The GAL16V8 is a fast PAL with a 3.5nsec decode time.
- It uses an erasable process technology, with a 100msec erase time.
- This device performed address decoding duties on the 6811 boards some that we used a few years ago in ECE 259.


Keyboard Interface
Mode Select Jumpers: MODA and MODB
MCU Port: Address + Data bus etc
IO Interface Port
RS232 Level Translator
8MHz Crystal
Prototyping Area
Serial Port
6811
TRACE/PROG Jumper
PWR Terminal Block
Power Jack
Reset Button
AD0-7 Multiplexer
Primary Memory Map Logic (GAL16V8)
Secondary Peripheral Chip Select Logic 74138 3-8 line decoder
U5 (32K battery backed SRAM)
U6 (32K SRAM not all accessible)
CPU BUS Port
LCD Port and Control Logic
U7 (8K EEPROM

Axiom 68HC11 - Microcomputer System Board