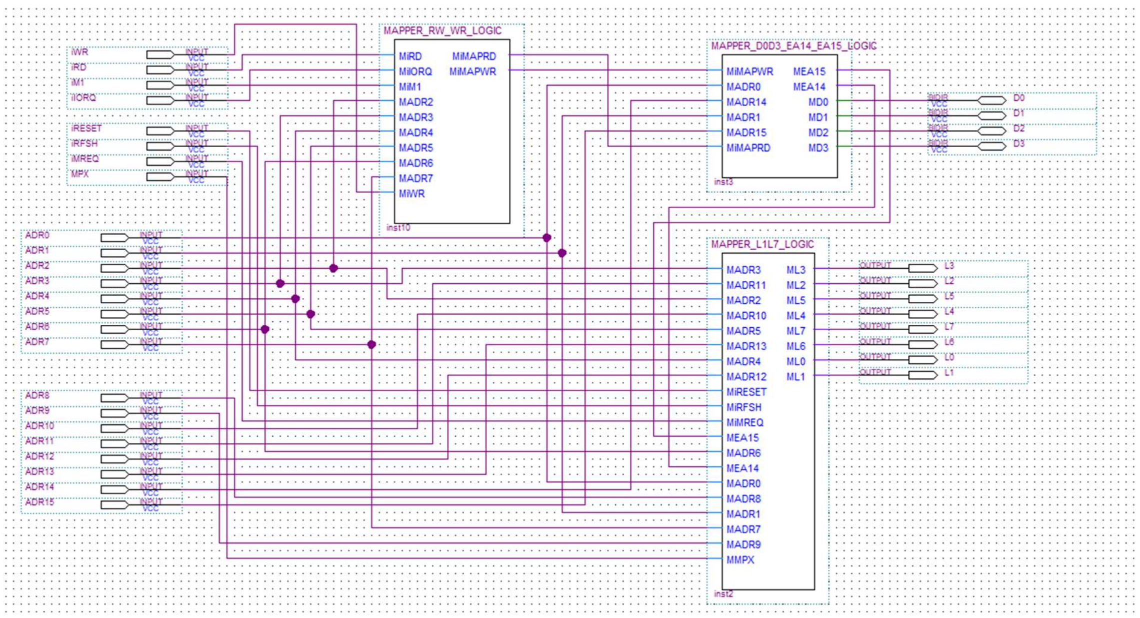
First Embryo of a Generic Memory mapper, limited to 256KB

In my search for the mapper to the SVI738 i came a cross different mapper, schematics and information how people have solved this, but my idea is to create a internal mapper and replace on-board 64KB with 256KB to have a full MSX2/MSX2+ after upgrade, issues I found was that the ULA that handle the SLT3-x is not correct, so that is part of the ULA project.

This mapper is very simple and can easy be implemented to other MSX computer.
So after doing some reverse engineering and looking at the internal memory mappers for the Philips NMS 8250, Yamaha YIS503, Expert2/3 schematics it landed into this design.
I also looked into the MSX Data Pack documentation, but the design there has no back annotation so it will not work in most European MSX-2(+) machines.

So to keep simple I divided the schematics into 3 blocks, and to keep it difficult 😉 I try to solve it using VHDL and CPLD chips, instead of traditional TTL LS chips, this way it will just be a EPM7064 and 2x44256 DRAM. But of course the design can be used implementing standard TTL LS chips. (bigger PCB board and more components is the backside)

So back to the design 😉 that is based on three build blocks, MAPPER_RW_WR_LOGIC, MAPPER _D0D3_EA14EA15_LOGIC and MAPPER_L0L7_LOGIC.



MAPPER_RW_WR_LOGIC is handling the READ/WRITE logic for the mapper.
MAPPER _D0D3_EA14EA15_LOGIC handles the D0-D3 lines with backward annotation + the EA14/EA15 address out the MAPPER_L0L7_LOGIC.
MAPPER_L0L7_LOGIC handles the address lines L0-L7 that are connected to the DRAM address lines.
NO additional /RAS, /CAS or MPX signal needed those are supply ULA, and by using 2 chips no additional split of /CAS is needed.

MAPPER_RW_RW_LOGIC

How does the RW/WR logic to the mapper works?

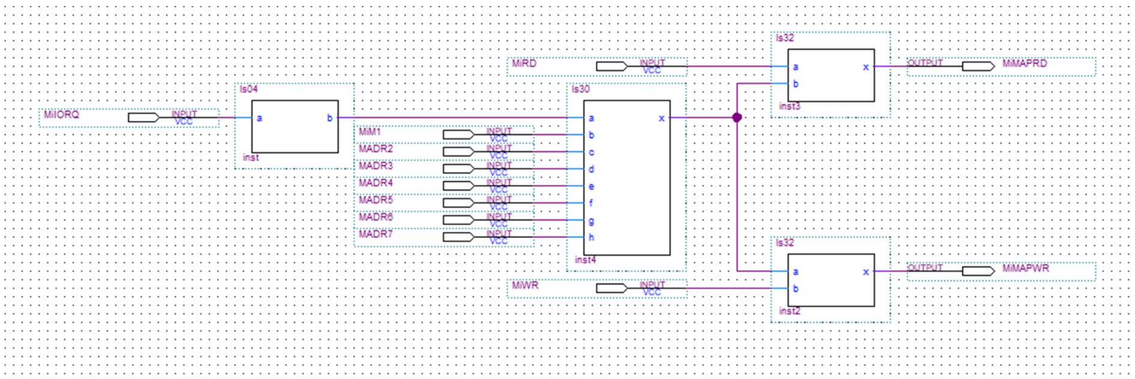
If all input signal to 8 input NAND are H output will be L otherwise combination it will be H, inputs are the lower address line A2 to A7 plus the /IORQ and /M1 signal from the CPU.
/IORQ signal indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. (Observe that the /IORQ is inverted before going into port a)

/M1 and /IORQ are both active, then that indicates an interrupt acknowledge cycle (an interrupt response vector can be placed on the data bus)

Now that output signal is the used together with the /RD and /WR using the OR gate to create the /MAPRD and /MAPWR signal.

/MAPRD and /MAPWR signal will only be L when /RD or /WR are L and the output from 8NAND is also L else it will be H. (this handle by LS32 NOR logic)

/RD Indicates that data is ready to be read from a memory or I/O device to the CPU.
/WR indicates that data is going to be written from the CPU data bus to a memory or I/O device.



MAPPER_D0D1_EA14EA15_LOGIC

This part of the design that handles the EA14 and EA15 line out, and the D0-D3 (this are BIDIR lines) data lines with backward annotation support.

Input lines to this module are /MiMAPRD and /MiMAPWR (see previous module) plus the MADR0, MADR1, MADR14 and MADR15. D0-D3 are also used as both input and output depending of state.

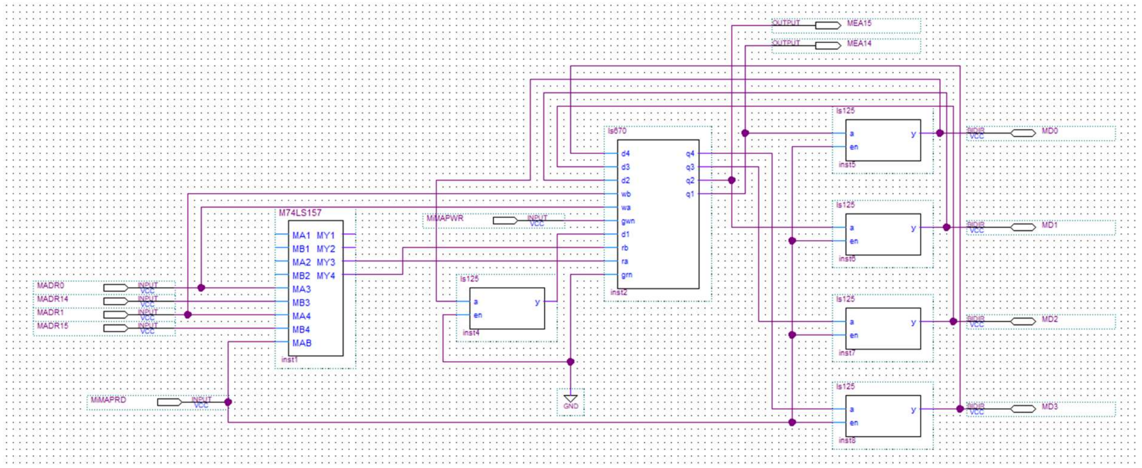
Start by looking at the LS157 chip, which is multiplexer and controlled by the /MAPiRD that goes into the port MAB. When MAB is L the MADR0 an MADRA1 are sent out to Y3 and Y4 and if H the MADR14 and MADR15 are sent out to Y3 and Y4 instead.

Now if we look into the part with LS125 chips, this are 3 state buffer, and the logic is very simple if EN are L then A=Y, and if EN are H the Y=Hi-Z. (Hi-Z is to effectively remove the output logic)
EN is handle by the / MiMAPRD line and to make a long story short if that is L then the output from LS670 chip is transfer to the D0-D3 lines, and if H its blocked.

Now to the more complex part, LS670 chip is 4 Word x 4 Bit register file logic and use following inputs D0-D3 as input to port D1-D4. The D0 is buffered by the LS125 (always passed because EN is connected to GND)

GWN is the Write Enable port and is handled by the /MiMAPWR and the WA and WB are mapped to MADR0 and MADR1. WA and WB are Write Address Inputs.
GWN is L then it enabled and disabled when H (depending of the state of /MiMAPWR)

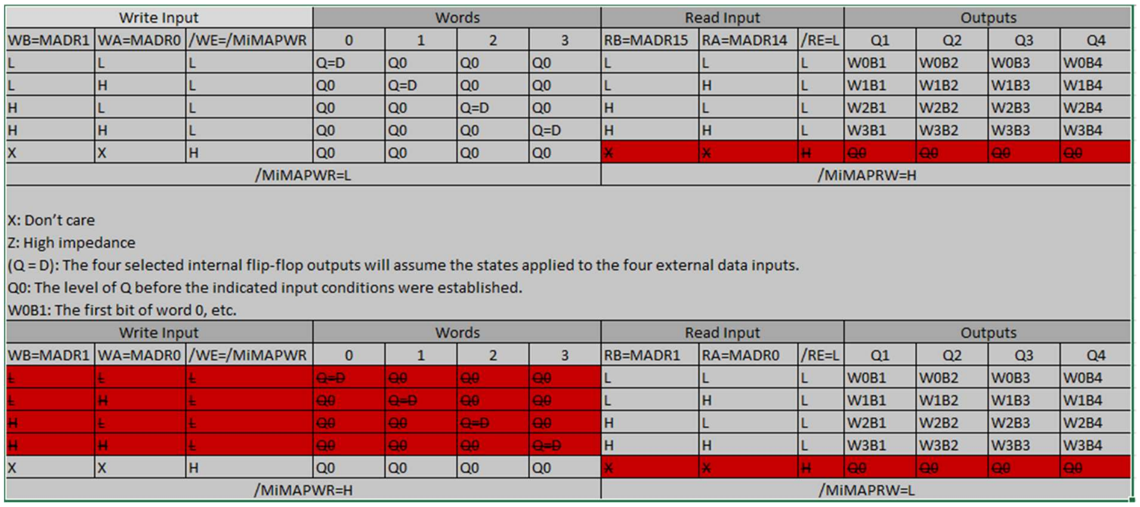
GRN is the Read Enable port is always L (mapped to GND) and the Read Address Inputs are handled by the RA and RB ports that are connected to MADR0-MADR1 (/MiMAPRD is L) or MADR14-MADR15 (/MiMAPRD is H).



The EA14 and EA15 output are mapped to Q1 and Q2, and always the WnB1 and WnB2 depending of RA and RB address that the LS157 passed through. (if /MiMAPRD is L or H)

Q1-Q4 is mapped to D0-D3 as output if /MiMAPRD is L (passed through the LS125 logic)

See table and Schematics to try to understand logic (its not easy - I think I got it right 😉)



Write Input			Words				Read Input			Outputs			
WB=MADR1	WA=MADR0	/WE=/MiMAPWR	0	1	2	3	RB=MADR15	RA=MADR14	/RE=L	Q1	Q2	Q3	Q4
L	L	L	Q=D	Q0	Q0	Q0	L	L	L	W0B1	W0B2	W0B3	W0B4
L	H	L	Q0	Q=D	Q0	Q0	L	H	L	W1B1	W1B2	W1B3	W1B4
H	L	L	Q0	Q0	Q=D	Q0	H	L	L	W2B1	W2B2	W2B3	W2B4
H	H	L	Q0	Q0	Q0	Q=D	H	H	L	W3B1	W3B2	W3B3	W3B4
X	X	H	Q0	Q0	Q0	Q0	~~X~~	~~X~~	~~H~~	~~Q0~~	~~Q0~~	~~Q0~~	~~Q0~~
/MiMAPWR=L							/MiMAPRW=H						

X: Don't care
Z: High impedance
(Q = D): The four selected internal flip-flop outputs will assume the states applied to the four external data inputs.
Q0: The level of Q before the indicated input conditions were established.
W0B1: The first bit of word 0, etc.

Write Input			Words				Read Input			Outputs			
WB=MADR1	WA=MADR0	/WE=/MiMAPWR	0	1	2	3	RB=MADR1	RA=MADR0	/RE=L	Q1	Q2	Q3	Q4
~~L~~	~~L~~	~~L~~	~~Q=D~~	~~Q0~~	~~Q0~~	~~Q0~~	L	L	L	W0B1	W0B2	W0B3	W0B4
~~L~~	~~H~~	~~L~~	~~Q0~~	~~Q=D~~	~~Q0~~	~~Q0~~	L	H	L	W1B1	W1B2	W1B3	W1B4
~~H~~	~~L~~	~~L~~	~~Q0~~	~~Q0~~	~~Q=D~~	~~Q0~~	H	L	L	W2B1	W2B2	W2B3	W2B4
~~H~~	~~H~~	~~L~~	~~Q0~~	~~Q0~~	~~Q0~~	~~Q=D~~	H	H	L	W3B1	W3B2	W3B3	W3B4
X	X	H	Q0	Q0	Q0	Q0	~~X~~	~~X~~	~~H~~	~~Q0~~	~~Q0~~	~~Q0~~	~~Q0~~
/MiMAPWR=H							/MiMAPRW=L						

MAPPER_L1L7_LOGIC

This module will handle the L1-L7 out to the DRAM, we will also handle the extra A7 refresh line that in PHILIPS design is called 'A7. (do not get confused 'A7 is not the A7)

Input ports are ADR0 to ADR7, ADR8 to ADR13 plus EA14 and EA15 output from the previous module MAPPER_D0D1_EA14EA15_LOGIC. Also the /RFSH, /MREQ, MPX and /RESET is used in the design. (in the schematic picture the are prefix with M or Mi)

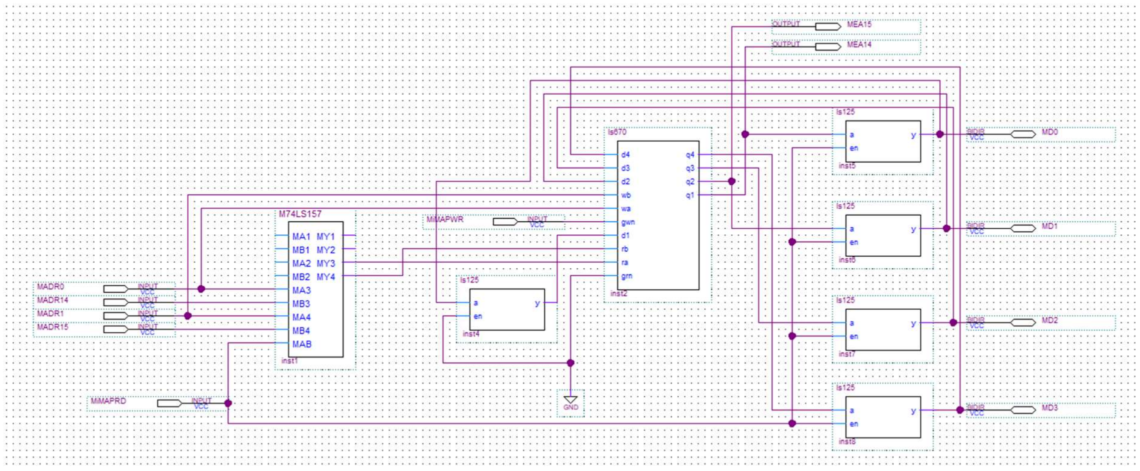
The MPX port (in some design called MUX) is the signal that toggle the two multiplexer chips LS157 when L it transfers ADR0 to ADR6 to the L0-L6 and when H it uses ADR8-ADR13 plus EA14 to the L0-L6 instead. (L7 with the ADR7 or EA15 is handle by the LS513 chip)

So if we simplify the logic the first LS157 handles lines L0 to L3 lines, and second L157 handles L4 to L6, the LS153 handle L7 where output is 'A7, ADR7, ADR15 and L32+2xLS74 chips handle the additional 'A7 refresh signal 😉

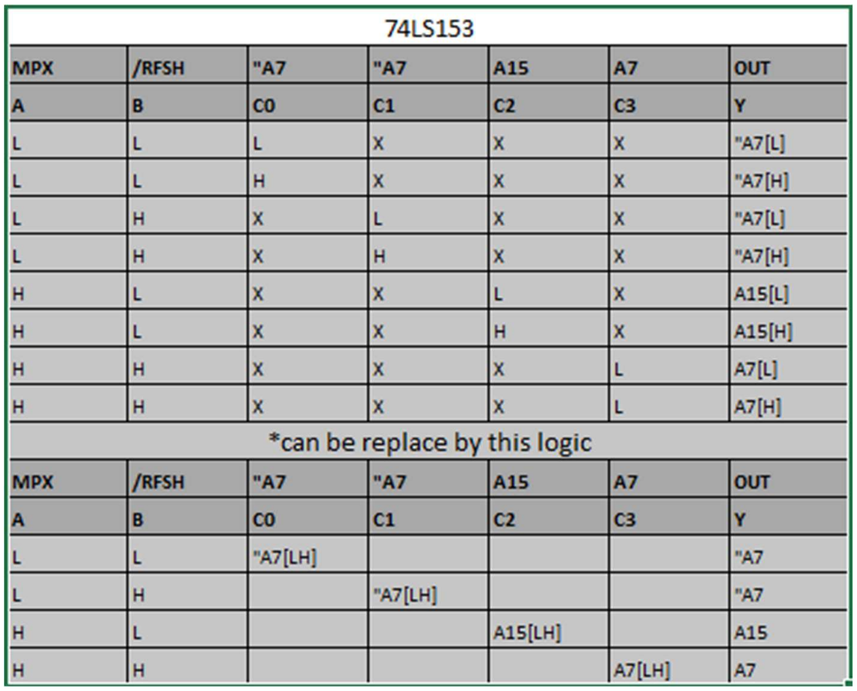
In MSX DATA PACK schematics for the memory logic they call this ROW\COL where ADR0-7 is ROW and ADR8-ADR15 is COL, this is how the Z80 CPU will be available to read or write to a specific memory cell when the D0-D7 bus contains read or write data from/to the DRAM.

Because the Z80 internal memory refresh (/RFSH) operation (increment lower 7 bits) only handles ADR0 to ADR6, when need to add an external logic that handles the ADR7 refresh.

In the design we used the ADR6 to create an additional refresh cycle for the ADR7 using the MiRFSH (/RFSH signal) and MiMREQ (/MREQ) and this is handled by the LS32 chip and the two LS74 chips only difference is that we added the MiRESET signal (/RESET) so when CPU reset its also do the 'A7 reset. 😊



In the original PHILIPS schematic they use 4 NAND gates to handle 'A7 and ADR7 lines, this is too complex to follow so I replace the logic using a LS153 chip (an one-bit wide 4 to 1 multiplexer) instead as design in the MSX DATA PACK (same logic vs simple logic 😊) and EXPERT2/3 schematics. when 'A7, ADR15 and ADR7 is passed to L7 depends on the MPX and MiRFSH (/RFSH) signal.



74LS153

MPX	/RFSH	"A7	"A7	A15	A7	OUT
A	B	C0	C1	C2	C3	Y
L	L	L	X	X	X	"A7[L]
L	L	H	X	X	X	"A7[H]
L	H	X	L	X	X	"A7[L]
L	H	X	H	X	X	"A7[H]
H	L	X	X	L	X	A15[L]
H	L	X	X	H	X	A15[H]
H	H	X	X	X	L	A7[L]
H	H	X	X	X	L	A7[H]

*can be replace by this logic

MPX	/RFSH	"A7	"A7	A15	A7	OUT
A	B	C0	C1	C2	C3	Y
L	L	"A7[LH]				"A7
L	H		"A7[LH]			"A7
H	L			A15[LH]		A15
H	H				A7[LH]	A7