Vectrex multicart

Date: 6/7/00

From: Ronen Habot

Subject: Menu driven multicart

Disclaimer

This document contains technical information regarding a general-purpose multicart concept. Under any circumstances, this information should not be used to mass manufacture and sell vectrex (or any other console) multicarts. All the rights to this multicart concept, electronic design and source are reserved. The writer of this document is not responsible for any console (and equipment) damage or, body injury caused by following the proposed instructions. To provide the writer more control regarding the distribution and use of this document, this document can not be printed, copied, or modified without a password. A password can be achieved individually by sending an email to the following address: webmaster@vgcollect.freehosting.net.

Introduction

As original game cartridges are becoming harder to find and therefore more expensive, as new games emerge every now and then and due to shortage in storage space for cartridges, I decided to look into the possibility of designing my own multicart. In addition, an easy to use and cheap to build were two of my goals when I first thought about it. The following paper describes in details what is the concept of the multicart and how does it work. I wrote this paper to let other videogame fans (with some technical understanding) the ability to better understand the concept and guts of this device.

Although, this paper refers mainly to the vectrex, the same concept can be identically adopted for every other console of the same type. In order to proof my concept I've built a prototype cartridge and wrote the menu code for the vectrex console. Since I have only one vectrex console, I can not guarantee that the following design would work on any other vectrex machine.

Background

A cartridge (usually) contains a single ROM device that is basically a peripheral device of the microprocessor's bus. The microprocessor is the main controller of the bus and in order to run a game, it executes predefined set of commands stored in that ROM device. This ROM is not the only peripheral device attached to the microprocessor's bus, in addition, another ROM (that resides in the console itself) stores the BIOS subroutines, the startup procedures and the built in Mine Storm game, as well as a RAM and other devices.

Anyway, to the cartridge edge connector only partial set of the microprocessor's bus are routed, which makes it a bit tougher to handle. Included in the provided signals are the signals that are essential for an external ROM to operate with couple of additional signals such as HALT and more. The actual signals that are provided to the ROM device are: Address bus A14 down-to A0 (enables 32K of ROM directly accessed, although the microprocessor has 16 address bits only 15 bits are provided to the cartridge), Data bus D7 down to D0, Read not (active low), Chip Enable not(active low). In addition a programmable I/O port is provided to the cartridge. This port can be used for instance, to generate a 16^{th} address bit for code larger than 32K. Although the vectrex cartridge connector provide the R/W~ signal, I decided to ignore its existence, due to my main goal of a general purpose, console independent multicart design.

One more aspect to keep in mind is the software. The software can be located within a 64K bytes address range and can jump to any location within this range. All vectrex programs start at address 0x0000. They all begin with a special identification header that is being detected by the BIOS start-up routines to determine whether there is a cartridge or not (and if not, the internal MINE STROM is launched).

Let's summarize what we know already:

- Each cartridge has (a straight forward) access to 32K bytes address space.
- Each cartridge has a Read, Chip enable signal but **no** Write signal (some console might have it but in this paper it is assumed not to be part of the provided bus to the cartridge).
- Address 0x0000 of any game contains a header detected by the console at power-up.

The multicart concept

To implement a multicart all what needs to be done is to cause the console to see any selected game at address 0x0000. In other words, if there is a way to cause the console to see at its address 0x0000 a different game at start-up the goal is completely achieved. This is the basic concept behind the "dip-switch" type of multicarts. To implement this concept, all what is needed is a large 8-bit memory (large enough to accommodate all the desired games) and a mechanism (dip-switch) to set a starting address for a selected game following the next guidelines:

Let's assume that the game size is 4K (4096) bytes. To access all the instructions for this game 12 (2^{12}= 4096) address bits are needed. If we place this game at location 0x0000 there is no problem at all. Now, let's add one more game into the same memory, but in a different location - at address 0x1000 (which is the 2^{nd} 4K bank of the same memory). With no external switching mechanism, every time we power up the console, obviously, the game residing at address 0x0000 will be played with no way to start playing the second game. On the other hand, if we have 3 dip-switches to replace address bits A14 down to A12, we could pre-define an offset (by setting them to the right value) for a different memory location (and thus a different game), plug it into the cartridge slot and turn the power on. Therefore, to play the 1^{st} game the dip-switch value is 000 and for the 2^{nd} game 001. Since the microprocessor doesn't "care" about addresses higher than 0x0FFF for each game (and therefore has address bits A14, A13 and A12 always 0) this concept works fine. I must admit that this method has some difficulties with a variation in game sizes (as the vectrex games do) but, this is the basic concept and not the whole solution.

Menu driven multicart

The concept in this case is the same. The major difference is that there is no physical dip-switch anymore but a simple memory element (D-LATCH) that latches the most significant bits (MSB) of the address based on the player selection for a specific game. Let's just keep in mind that the cartridge does not have access to the microprocessor's Write signal - which make it a bit tougher to handle.

The missing Write signal is the first obstacle in our way towards the end product. The solution for this is a bit tricky and here is how to solve it: The basic idea is to write (required game offset) through read only operation. This is achieved by predefining an address (one or more) that once the program read from, the required offset is written into the MSB latch. To implement that, an address decoder has to be designed to issue a latch enable once all the conditions (access to the predefined address and read active and chip select active) are met. The next question is what should the latch capture address or data? I think that latching address is simpler since the data has to be both meaningful for the microprocessor and also, to point to the correct offset. Address on the other-hand changes the execution location on the fly but in my opinion it is easier to handle and therefore this is the approach that was implemented.

The following section shows the whole memory map of the multicart. It assumes 512K bytes ROM (Actually Am29F040-PC120) to be used for this purpose. In general, the whole address space is divided into 4 banks of 128K each. The idea is to support a variety of game sizes (4K, 8K, 16K and 32K). Each bank can contain games smaller (or equal) in size to the specified size. The next table describes better the meaning of that idea:

Bank	Address range	Bank Type	Supported game sizes
0	0x00000	4K	4K
	0x1FFFFF		
1	0x20000	8K	4K / 8K
	0x2FFFFF		
2	0x40000	16K	4K / 8K / 16K
	0x5FFFFF		
3	0x60000	32K	4K / 8K / 16K / 32K
	0x7FFFFF		

A more detailed table with all the game locations, sizes and ranges is provided below:

		A[14:0] - 32K ROM	A[18:0] - 512K ROM	JMP	Content
0K	0000	000 0000 0000 0000	000 0000 0000 0000 0000	-	Menu Prg
	0FFF	000 1111 1111 1111	000 0000 1111 1111 1111		
4K	1000	001 0000 0000 0000	000 0001 0000 0000 0000	7F02	Armor Attack
	1FFF	001 1111 1111 1111	000 0001 1111 1111 1111		
8K	2000	010 0000 0000 0000	000 0010 0000 0000 0000	7F04	Art Master
	2FFF	010 1111 1111 1111	000 0010 1111 1111 1111		
12K	3000	011 0000 0000 0000	000 0011 0000 0000 0000	7F06	Bedlam
	3FFF	011 1111 1111 1111	000 0011 1111 1111 1111		
16K	4000	100 0000 0000 0000	000 0100 0000 0000 0000	7F08	Berzerk
	4FFF	100 1111 1111 1111	000 0100 1111 1111 1111		
20K	5000	101 0000 0000 0000	000 0101 0000 0000 0000	7F0A	4D Rotcub
	5FFF	101 1111 1111 1111	000 0101 1111 1111 1111		
24K	6000	110 0000 0000 0000	000 0110 0000 0000 0000	7F0C	Castle
	6FFF	110 1111 1111 1111	000 0110 1111 1111 1111		
28K	7000	111 0000 0000 0000	000 0111 0000 0000 0000	7F0E	ROM dump
	.	.	.		
	.	.	.		
	7F00	111 1111 0000 0000	000 0111 1111 0000 0000		*Jump table*
	7F01	111 1111 0000 0001	000 0111 1111 0000 0001		
	7F02	111 1111 0000 0010	000 0111 1111 0000 0010		Armor Attack
	7F03	111 1111 0000 0011	000 0111 1111 0000 0011		
	7F04	111 1111 0000 0100	000 0111 1111 0000 0100		Art Master
	.	.	.		
	.	.	.		
	.	.	.		
	.	.	.		
	7FEC	111 1111 1111 1100	000 0111 1111 1111 1100		
	7FFD	111 1111 1111 1101	000 0111 1111 1111 1101		
	7FFE	111 1111 1111 1110	000 0111 1111 1111 1110		
	7FFF	111 1111 1111 1111	000 0111 1111 1111 1111		
32K	8000	---Not Available ---	000 1000 0000 0000 0000	7F10	Chasm
			000 1000 1111 1111 1111		
36K	9000	---Not Available ---	000 1001 0000 0000 0000	7F12	hyper
			000 1001 1111 1111 1111		
40K	A000	---Not Available ---	000 1010 0000 0000 0000	7F14	mine2
			000 1010 1111 1111 1111		
44K	B000	---Not Available ---	000 1011 0000 0000 0000	7F16	ripoff

			000 1011 1111 1111 1111		
48K	C000	---Not Available ---	000 1100 0000 0000 0000 000 1100 1111 1111 1111	7F18	
52K	D000	---Not Available ---	000 1101 0000 0000 0000 000 1101 1111 1111 1111	7F1A	Scarmble
56K	E000	---Not Available ---	000 1110 0000 0000 0000 000 1110 1111 1111 1111	7F1C	Solar
60K	F000	---Not Available ---	000 1111 0000 0000 0000 000 1111 1111 1111 1111	7F1E	Space
64K	10000	---Not Available ---	001 0000 0000 0000 0000 001 0000 1111 1111 1111	7F20	Starhawk
68K	11000	---Not Available ---	001 0001 0000 0000 0000 001 0001 1111 1111 1111	7F22	Startrek
72K	12000	---Not Available ---	001 0010 0000 0000 0000 001 0010 1111 1111 1111	7F24	Sweep
76K	13000	---Not Available ---	001 0011 0000 0000 0000 001 0011 1111 1111 1111	7F26	test cart
80K	14000	---Not Available ---	001 0100 0000 0000 0000 001 0100 1111 1111 1111	7F28	Trek2
84K	15000	---Not Available ---	001 0101 0000 0000 0000 001 0101 1111 1111 1111	7F2A	Vectrace
88K	16000	---Not Available ---	001 0110 0000 0000 0000 001 0110 1111 1111 1111	7F2C	Vpong
92K	17000	---Not Available ---	001 0111 0000 0000 0000 001 0111 1111 1111 1111	7F2E	
96K	18000	---Not Available ---	001 1000 0000 0000 0000 001 1000 1111 1111 1111	7F30	
100K	19000	---Not Available ---	001 1001 0000 0000 0000 001 1001 1111 1111 1111	7F32	
104K	1A000	---Not Available ---	001 1010 0000 0000 0000 001 1010 1111 1111 1111	7F34	Engine
108K	1B000	---Not Available ---	001 1011 0000 0000 0000 001 1011 1111 1111 1111	7F36	
112K	1C000	---Not Available ---	001 1100 0000 0000 0000	7F38	

			001 1100 1111 1111 1111		
116K	1D000	---Not Available ---	001 1101 0000 0000 0000 001 1101 1111 1111 1111	7F3A	
120K	1E000	---Not Available ---	001 1110 0000 0000 0000 001 1110 1111 1111 1111	7F3C	
124K	1F000	---Not Available ---	001 1111 0000 0000 0000 001 1111 1111 1111 1111	7F3E	
From here 8K games area starts					
128K	20000	---Not Available ---	010 0000 0000 0000 0000 010 0001 1111 1111 1111	7F40	3d crazy cst
	22000	---Not Available ---	010 0010 0000 0000 0000 010 0011 1111 1111 1111	7F44	Blitz
	24000	---Not Available ---	010 0100 0000 0000 0000 010 0101 1111 1111 1111	7F48	Crazy
	26000	---Not Available ---	010 0110 0000 0000 0000 010 0111 1111 1111 1111	7F4C	Heads up
	28000	---Not Available ---	010 1000 0000 0000 0000 010 1001 1111 1111 1111	7F50	Melody
	2A000	---Not Available ---	010 1010 0000 0000 0000 010 1011 1111 1111 1111	7F54	3d mine
	2C000	---Not Available ---	010 1100 0000 0000 0000 010 1101 1111 1111 1111	7F58	3d narrow
	2E000	---Not Available ---	010 1110 0000 0000 0000 010 1111 1111 1111 1111	7F5C	Narzod
	30000	---Not Available ---	011 0000 0000 0000 0000 011 0001 1111 1111 1111	7F60	
	32000	---Not Available ---	011 0010 0000 0000 0000 011 0011 1111 1111 1111	7F64	Polar
	34000	---Not Available ---	011 0100 0000 0000 0000 011 0101 1111 1111 1111	7F68	Pole
	36000	---Not Available ---	011 0110 0000 0000 0000 011 0111 1111 1111 1111	7F6C	Spike
	38000	---Not Available ---	011 1000 0000 0000 0000 011 1001 1111 1111 1111	7F70	Spinball

	3A000	---Not Available ---	011 1010 0000 0000 0000 011 1011 1111 1111 1111	7F74	
	3C000	---Not Available ---	011 1100 0000 0000 0000 011 1101 1111 1111 1111	7F78	Webwars
	3E000	---Not Available ---	011 1110 0000 0000 0000 011 1111 1111 1111 1111	7F7C	
From here 16K games area starts					
256K	40000	---Not Available ---	100 0000 0000 0000 0000 100 0011 1111 1111 1111	7F80	Vaboom
	44000	---Not Available ---	100 0100 0000 0000 0000 100 0111 1111 1111 1111	7F88	
	48000	---Not Available ---	100 1000 0000 0000 0000 100 1011 1111 1111 1111	7F90	Darktower
	4C000	---Not Available ---	100 1100 0000 0000 0000 100 1111 1111 1111 1111	7F98	Spikesh
	50000	---Not Available ---	101 0000 0000 0000 0000 101 0011 1111 1111 1111	7FA0	
	54000	---Not Available ---	101 0100 0000 0000 0000 101 0111 1111 1111 1111	7FA8	Galaxian
	58000	---Not Available ---	101 1000 0000 0000 0000 101 1101 1111 1111 1111	7FB0	Frogger
	5C000	---Not Available ---	101 1100 0000 0000 0000 101 1111 1111 1111 1111	7FB8	
From here 32K games area starts					
384K	60000	---Not Available ---	110 0000 0000 0000 0000 110 0111 1111 1111 1111	7FC0	
	68000	---Not Available ---	110 1000 0000 0000 0000 110 1111 1111 1111 1111	7FD0	
	70000	---Not Available ---	111 0000 0000 0000 0000 111 0111 1111 1111 1111	7FE0	
	78000	---Not Available ---	111 1000 0000 0000 0000 111 1111 1111 1111 1111	7FF0	
512K	***END OF MEMORY - No more games***				

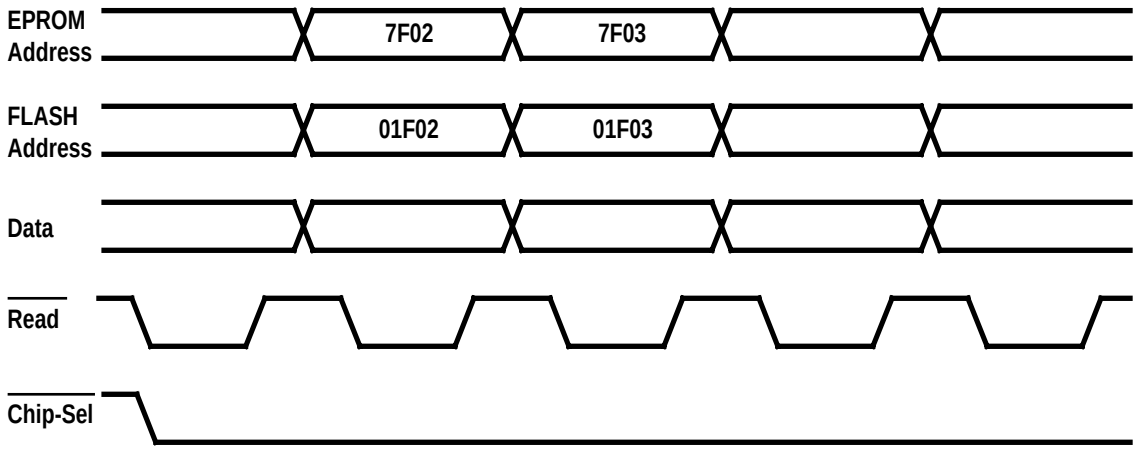
Implementation

The implementation is based mostly on address manipulation and not on data. The basic idea, is to define a way that when the PC of the microprocessor jumps to, will latch a desired offset for the MSB of the ROM in the latch (the electronic "dip-switch"). This is achieved by the following means - address space 0x7F00 to 0x7FFF (256Bytes) is reserved for that purpose. Now, when a game is selected through the selection menu, the program jumps to a predefined address (provided in the last table) and in that instance here is what happens:

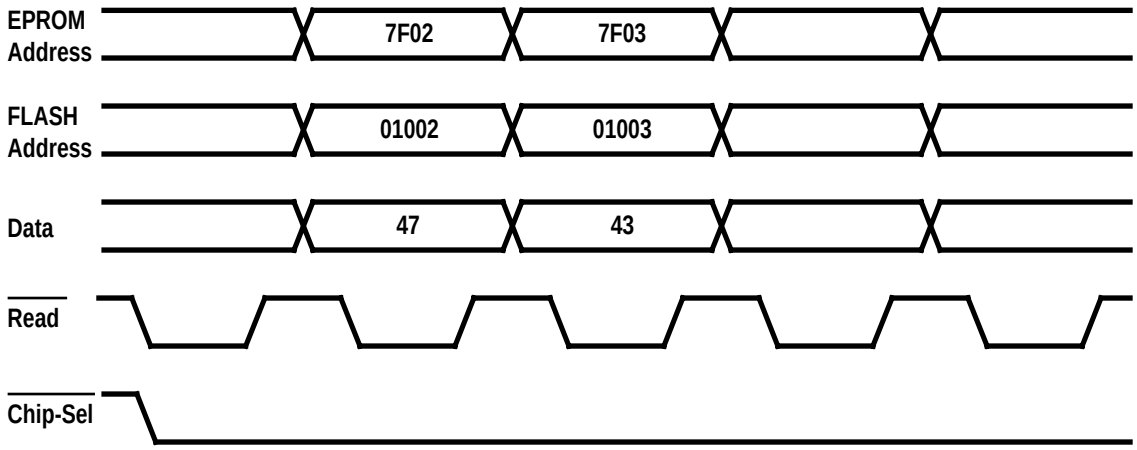
- The external address bus is logically divided into two sections: the "valid jump" address, which, in that case address bits A14 down-to A8 are all '1' (logic high) and the offset, which in this case is address bits A7 down-to A1.
- A decoding logic "detects" that the current address is a "valid jump" address and this is what enables the latch to store the offset - provided by address bits A7 down-to A1.
- The latch outputs are routed to the ROM most significant bits and remain unchanged for the whole game play. In that way, even though the game is stored in high address space, to the console it looks like as 0x0000 based game.
- At the end of the game, whenever the (BIOS routine) warm or cold start procedure is called by the original game, the PC goes through address 0x0000. This address is being decoded (as well) and a latch-enable is being generated (same as if it would be a "valid address") which in response latches all '0' as an offset. As a result, the menu program is being executed again (since it is located at address 0x0000). The only disadvantage of this approach is that the player has to brows through the menu and select a game every time a game ends, even if it the same game (this could be solved if there was a way to store the played game index in such a way that any played game wouldn't change - within the EPLD for instance).

To implement this logic, I've used an EPLD device that can electronically be erased and reprogrammed with a new content (design). The selected address space of 0x7FXX has been chosen due to the decoding ease - simply an AND gate that get all the relevant address bits as inputs.

Now, with all the above knowledge, let's have a short demonstration of how does the concept work by drawing the timing diagrams of the inputs and the outputs of the EPLD. For this example, let's assume that address bits A11 down-to A4 are connected directly to the Flash and are not passing through the EPLD.



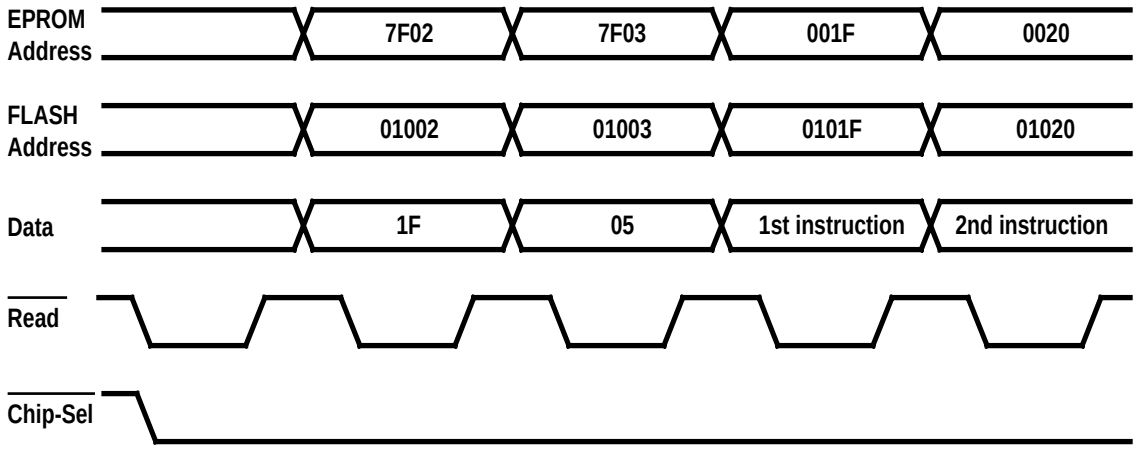
In this example, the player has selected ARMOR ATTACK, in response, the PC is modified to be 0x7F02. As a result, the value shown above in the Flash Address waveform indicates the location of the code that is going to be executed from the Flash. In that case, this location is towards the end of the game (where data structures are stored) which is not a meaningful code for the microprocessor to execute. Therefore, an additional logic has to be added to the EPLD to overcome this problem. The solution is as follows: these address bits (A11 down-to A4) should be masked (forced to be '0') while the "valid address" condition is met. Implementing this approach would provide the following timing diagram:



In this case, the address is correct, but, the data read from the Flash is part of the Vectrex header that must be present at the beginning of each game (in this case the letters GC that are part of the "© GCE ____" that shows up after turning the console on with a cartridge inserted) and not actual code. That can cause (and actually does) the microprocessor to execute invalid instructions and eventually the screen becomes blank. So, here is a new problem. One more thing to keep in mind is that each game has a different header length and therefore, the actual game starts in a different offset relatively to the starting address. The solution for this problem is as follows: In the main program, a table is predefined with 2 columns:

1. The physical address location of each game (as described in the previous table).
2. The first code address after the game's header (for "ARMOR ATTACK" the address is 0x1F).

So, just before the "jump" to the desired 0x7Fxx address is executed, register D (of the 6809) is loaded with the first code address of the selected game. And, since, the header of the game is no longer needed (all what we want is to start playing the game) the pointed addresses content is swapped (manually, through a binary editor) with the following op-codes: 0x1F followed by 0x05 - which translate to **TFR D,PC**. In that case, the microprocessor is executing a command that changes the PC to point to the beginning of the selected game, which causes the game to be immediately executed. Please note, that here since there is no 0x7Fxx involved, no new address is being latched and there is no problem with the offset. In that case the timing diagram (for "ARMOR ATTACK") looks as shown below:



In this case, the game is executed with no issues till it gets to its end. At this point, either warm or cold start routines is called from the BIOS, which causes the address to restart the cartridge detecting procedure again. This ensures that the address bus becomes 0x0000. In response, the decoding logic (within the EPLD) is latching the offset - 0x00 that points back to the main program - the menu selection. This decoding logic is also useful at power-up where the D-Latch comes with unknown values, and as a result 0x00 is latched to guarantee the proper start of the main menu program.

This description, would have come to its end if all the games were of the same size. As we all know, there are games that occupy larger amounts of memory. The following section describes the approach that was implemented to support this variety of game sizes on one hand, and to avoid any unused memory space on the other hand.

The most straight-forward approach is to divide the memory into equal size segments, of the same size as the largest supported game (e.g., 32K byte). The disadvantage of this method is, obviously, the waist of memory space for the smaller sized games (28K bytes are waisted for each 4K bytes game). In order to avoid this type of memory utilization, I divided the whole memory space into 4 (equal size) segments of 128Kbytes each. The 1st segment is identified as the 4K games segment, the 2nd segment as the 8K games, the 3rd as the 16K and the 4th segment is identified as the 32K games. Now, let's keep in mind that for 4K games, 12 address bits are required and therefore, the rest of the Flash's address bits (A18 down-to A12) can be latched and remain unchanged for the whole game play (as long as the game is smaller than 4K bytes). Since the microprocessor actually doesn't activate any address bit beyond A11. As 8K games go, the microprocessor won't toggle any address bit beyond A12 - but A12 has to be provided by the microprocessor towards the Flash. Same approach is applied for larger game sizes: for 16K games the microprocessor won't change ant address bit beyond A13 and for 32K games A14 is still controlled by the microprocessor. The segmentation is defined by the 2 MSB of the address space - A18 and A17. An additional decoding logic has been added to decode the address segmentation as follows:

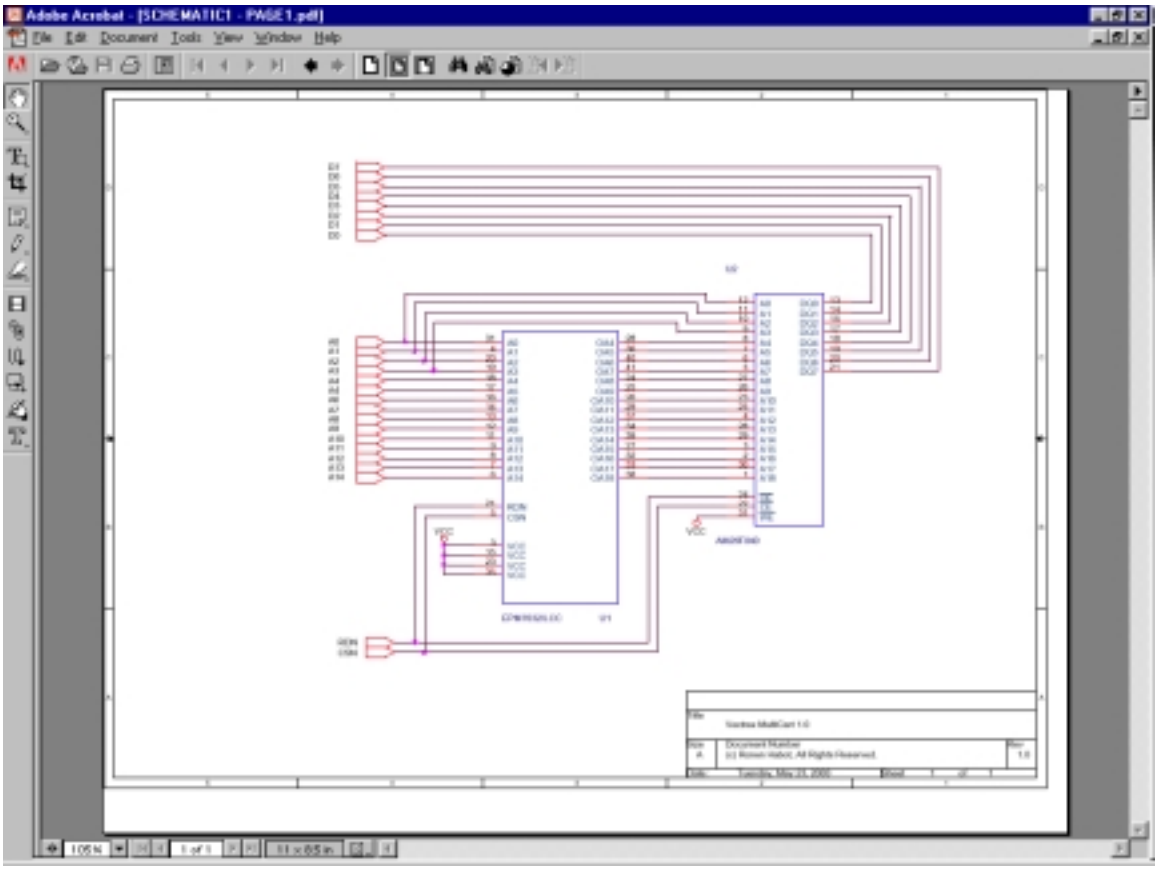
A[18:17]	Segment
00	4K
01	8K
10	16K
11	32K

There are 3 bits that are relevant for the game size as far as the Flash is concerned - A[14:A12]. The output of the decoding logic is basically a multiplexer selection for each one of these 3 bits as described below:

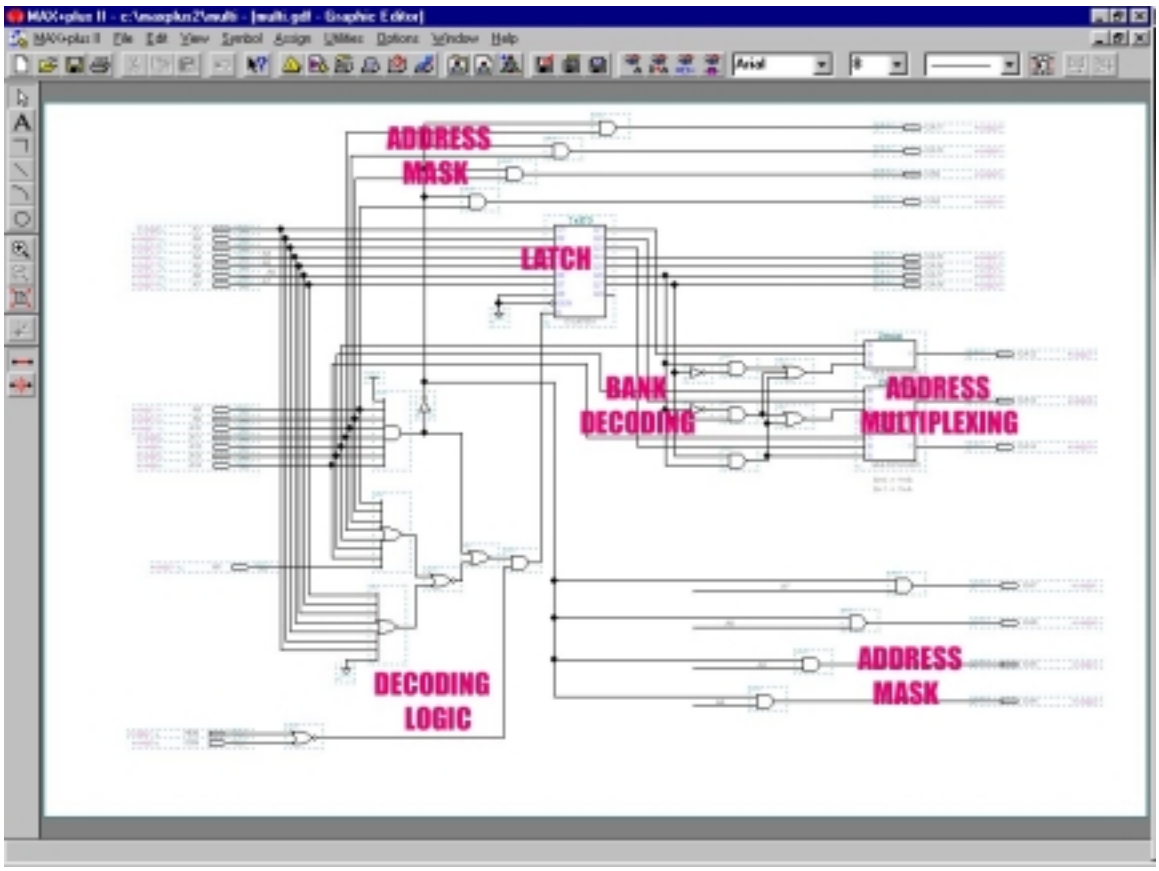
- For 4K games, A[14:12] are latched and steady through the whole game. The multiplexer selects the D-LATCH outputs and routes them towards the Flash.
- For 8K games, A[14:13] are latched and steady and A12 (that changes trough the game play) is provided - unlatched - to the Flash.
- For 16K games, A14 is latched and A[13:12] are provided - unlatched - to the Flash.
- For 32K games, A[14:12] are provided - unlatched - to the Flash.

Under these conditions, a game, smaller in size than the segment's name can fit into it. The disadvantage is the overhead penalty but if there is no other choice, it is better than nothing…

The next drawing depicts the cartridge (or EPORM replacement) schematics. The required components are a Flash (or similar sized) EPROM and the EPLD device. Please note that there are pin differences between different Flash devices.



The following schematic represents the content of the EPLD device.



The code

The ROM content is assembled in several steps as described in the following section. The required steps are as follows:

- Menu program
- Game conversion (and modification)
- Bank compilation
- BIN generation

The 1st step is to write the code for the menu of the game selection. Since printing the whole game list at once on the Vectrex screen is too flickering, I wrote a scrolling menu. The player selects a game by pressing controller #1's buttons 1/2 for up/down single step, or 3 & 1/2 for multiple steps browsing through the complete game list. To start a selected game, button 4 has to be pressed.

Next, the 2nd step involves with converting the game BIN files into a form that is easy to recompile and modified as needed. For that purpose I've used a couple of (UNIX) scripts that convert the BIN files into bytes in hex format, then add the assembler command - **FCB** and in that way, each BIN file has been converted to a file with the .inc extension. In that phase of preparation, I also modified the right header location as described in the ***implementation*** section. An example for this type of file is provided below:

```
FCB $67,$20,$47,$43,$45,$20,$31,$39,$1F,$05,$80,$fd,$1d,$f8,$50,$20
```

```
       FCB $d0,$42,$45,$52,$5a,$45,$52,$4b,$80,$00,$cc,$02,$00,$bd,$f7,$a9
       FCB $96,$79,$47,$97,$82,$0f,$21,$0f,$22,$bd,$f5,$33,$0f,$67,$bd,$f1
       …
```

The 3rd step involves with preparing the 64K banks of the code to be later merged into a single BIN file. For this purpose additional 8 assembler files were created with the following content:

- **ORG** command for the game location relative to the bank beginning.
- **INCLUDE** command for the .inc file of the desired game.

This file is compiled and the BIN file is used in the final (4th) step.

An example for such a file is shown below:

```
;will be placed at 0x20000
;Where 8K games start

       ORG  0x0000
       INCLUDE "src/multi/games/3dczycst.inc"

       ORG  0x2000
       INCLUDE "src/multi/games/blitz.inc"

       …
```

Then, the final step is to merge the menu program and the 7 additional 64K banks into a single BIN file. This was done through a BIN editor (available freely on the WWW) and then saved as a new BIN file. This new 512K BIN file was then programmed into the Flash memory. Please note that the 1st 64K bank has the menu program in it. It also contains the **ORG** & **INCLUDE** commands for rest of the games located in this bank.

The source code of the menu program is listed below. Please note that not all the released Vectrex are present - mostly to the fact that I did not receive an explicit permission from their writers to include them here. The end of the menu code contains the **INCLUDE** commands for the rest 60K of the 1st 64K bank.

```
;***************************************************************************
;              VECTREX MULTICART - MAIN MENU SELECTION PROGRAM
;                      Written by Ronen Habot, May-2000
;                            All rights reserved
;***************************************************************************

       INCLUDE "src/multi/whole/vectrex.inc"

;***************************************************************************
; General constants
;***************************************************************************
NUMBER_OF_GAMES      EQU    39
NUMBER_OF_GAMES_ON_SCRN EQU $05
MENU_LINE_SIZE       EQU    $15
MENU_LINE_SPACE      EQU    $10
MENU_Y_POS           EQU    $40
MENU_X_POS           EQU    $E5


;***************************************************************************
; Table definition of games in the memory
;***************************************************************************
```

```
;4K games start here...
ARMOR_JMP             EQU    $7F02
ART_JMP               EQU    $7F04
BEDLAM_JMP            EQU    $7F06
BERZERK_JMP           EQU    $7F08
D_ROTOCU_JMP          EQU    $7F0A
SCASTLE_JMP           EQU    $7F0C
ROM_JMP               EQU    $7F0E
CHASM_JMP             EQU    $7F10
HYPER_JMP             EQU    $7F12
MINE2_JMP             EQU    $7F14
RIPOFF_JMP            EQU    $7F16
EMPTY_4K_0_JMP        EQU    $7F18 ;;
SCRAMBLE_JMP          EQU    $7F1A
SOLAR_JMP             EQU    $7F1C
SPACEWAR_JMP          EQU    $7F1E
SHAWK_JMP             EQU    $7F20
STREK_JMP             EQU    $7F22
SWEEP_JMP             EQU    $7F24
TEST_JMP              EQU    $7F26
STREK2_JMP            EQU    $7F28
VRACE_JMP             EQU    $7F2A
VPONG_JMP             EQU    $7F2C
EMPTY_4K_1_JMP        EQU    $7F2E ;;
POP_JMP               EQU    $7F30
EMPTY_4K_2_JMP        EQU    $7F32 ;;
ENGINE_JMP            EQU    $7F34
EMPTY_4K_3_JMP        EQU    $7F36
EMPTY_4K_4_JMP        EQU    $7F38
EMPTY_4K_5_JMP        EQU    $7F3A
EMPTY_4K_6_JMP        EQU    $7F3C
EMPTY_4K_7_JMP        EQU    $7F3E
;8K games start here...
D_CRAZY_JMP           EQU    $7F40
BLITZ_JMP             EQU    $7F44
STARDEMO_JMP          EQU    $7F48
HEADSUP_JMP           EQU    $7F4C
MELODY_JMP            EQU    $7F50
D_MINESTORM_JMP       EQU    $7F54
D_NARROW_JMP          EQU    $7F58
FNARZOD_JMP           EQU    $7F5C
EMPTY_8K_0_JMP        EQU    $7F60   ;;
POLAR_JMP             EQU    $7F64
POLE_JMP              EQU    $7F68
SPIKE_JMP             EQU    $7F6C
SPINB_JMP             EQU    $7F70
EMPTY_8K_1_JMP        EQU    $7F74   ;;
WEBWAR_JMP            EQU    $7F78
EMPTY_8K_2_JMP        EQU    $7F7C
;16K games start here...
VABM_JMP              EQU    $7F80
SPIKEHOP_JMP          EQU    $7F88
DARKT_JMP             EQU    $7F90
SPIKESH_JMP           EQU    $7F98
EMPTY_16K_0_JMP       EQU    $7FA0  ;;
GALAXIAN_JMP          EQU    $7FA8
FROGGER_JMP           EQU    $7FB0
EMPTY_16K_1_JMP       EQU    $7FB8  ;;
;32K games start here...
EMPTY_32K_0_JMP       EQU    $7FC0   ;;
EMPTY_32K_1_JMP       EQU    $7FD0   ;;
EMPTY_32K_2_JMP       EQU    $7FE0   ;;
```

```
EMPTY_32K_3_JMP      EQU    $7FF0   ;;


;***************************************************************************
;Address of 1st opcode after the game header
;***************************************************************************
D_CRAZY_START        EQU    $0020  ;v?
D_MINESTORM_START    EQU    $0020  ;v
D_NARROW_START       EQU    $0024  ;v
D_ROTOCU_START       EQU    $000E  ;v
AGT_START            EQU    $0038  ;v?
ARMOR_START          EQU    $001E  ;v
ART_START            EQU    $0021  ;v
BEDLAM_START         EQU    $001E  ;v
BERZERK_START        EQU    $001A  ;v
BLITZ_START          EQU    $0019  ;v
SWEEP_START          EQU    $001E  ;v
CHASM_START          EQU    $001F  ;v
DARKT_START          EQU    $0021  ;v
ENGINE_START         EQU    $0022  ;v?
FNARZOD_START        EQU    $002D  ;v
FROGGER_START        EQU    $003A  ;v?
GALAXIAN_START       EQU    $0048  ;v?
ROCKS_START          EQU    $0023   ;v?
HEADSUP_START        EQU    $001B  ;v
HYPER_START          EQU    $001D  ;v
MELODY_START         EQU    $0024  ;v
MINE2_START          EQU    $0016  ;???
MOON_START           EQU    $001E  ;v?
OMEGA16K_START       EQU    $001E  ;v?
PATRIOT_START        EQU    $0041  ;v
POLAR_START          EQU    $001F  ;v
POLE_START           EQU    $004F  ;v
RIPOFF_START         EQU    $0022  ;v
ROM_START            EQU    $001B  ;v
SCRAMBLE_START       EQU    $001B  ;v
SOLAR_START          EQU    $001E  ;v
SPACEWAR_START       EQU    $001D  ;v
SPIKE_START          EQU    $0018  ;v
SPIKEHOP_START       EQU    $0014  ;v?
SPINB_START          EQU    $001B  ;v
SCASTLE_START        EQU    $001E  ;v
STARDEMO_START       EQU    $0036  ;v?
SHAWK_START          EQU    $001C  ;v
STREK_START          EQU    $0027  ;v
STREK2_START         EQU    $0027  ;v?
SWB_ANA_START        EQU    $0038  ;v?
TEST_START           EQU    $001D  ;v
VABM_START           EQU    $001A  ;v
VADERS_START         EQU    $0080  ;v
VRACE_START          EQU    $001B  ;v
VM_BNK1_START        EQU    $002C  ;v?
VM_BNK2_START        EQU    $002C  ;v?
VPONG_START          EQU    $0039  ;v
WEBWAR_START         EQU    $001D  ;v

;***************************************************************************
; Temporary variables that are going to be overwritten by the selected game
; when started.
;***************************************************************************
TempByte             EQU    $C880
GameIndex            EQU    $C881
```

```
FrameYpos            EQU    $C882
GameAddress          EQU    $C884
Vec_Text_Width_neg   EQU    $C886
game_cur_list        EQU    $CA00


;***************************************************************************
; Begin of the menu code:
;***************************************************************************
                           CODE
        ORG     $0000


        FCC     "g GCE 2000"
        FCB     $80
        FDB     $FD81
        FDB     $f850
        FDB     $00B0
        FCC     "VECTREX MULTI CART"
        FCB     $80

        FDB     $FA40
        FDB     $A6C0
        FCC     "g RONEN HABOT, REV 01"
        FCB     $80

        FDB     $FA40
        FDB     $90C0
        FCC     "ALL RIGHTS RESERVED"
        FCB     $80
        FCB     $0

       LDA   #$00                       ;Clear required parameters
       STA   GameIndex
       STA   TempByte

        LDD   #$FC20                    ;Set the H and W of the text
        STD   Vec_Text_HW

       JSR   update_menu_list           ;Initialize menu in RAM


;***************************************************************************
; Main program starts here
;***************************************************************************
menustart:
        JSR   Wait_Recal                ;reset the crt
        LDA   #$7f                      ;get the inte
        STA   VIA_t1_cnt_lo             ;Set the VIA_t1_lo register
        JSR   Intensity_to_a            ;set intensity
        JSR   print_names               ;Print the names on the screen
        JSR   print_frame               ;Draw the frame at the center
        JSR   menu_check_btns           ;Check the buttons of controller #1
        JSR   update_menu_list          ;Update the list to be printed
        BRA   menustart                 ;end of main loop


;***************************************************************************
; This procedure is the main idea behind the whole multicart concept:
; Based on the GameIndex the program jumps to a predefined location that
; will be captured by the ALTERA (EPLD device) and then 2 things will happen:
; 1. The offset will be stored in the ALTERA's latch and be constant for the
```

```
;    whole gameplay.
; 2. The PC will get the offset required to start the selected game.
; In case of games greater than 4K, the MSB of the target address has to be set.
;*****************************************************************************
start_selected_game:
       LDU   #start_loc_tbl
       LDB   GameIndex
       CLRA
       ADDD  TempByte
       ADDD  TempByte
       ADDD  TempByte
       ADDD  #$02                  ;A=(4*GameIndex)+2
       LDX   D,U                   ;X <- JMP address
       SUBD  #$02                  ;A=(4*GameIndex)
       LDD   D,U                   ;D <- Value for PC right after JMP
       TFR   X,PC                  ;Actually jump to X


;*****************************************************************************
; Gets the GameIndex and modifies the printed portion of the menu accordingly.
;*****************************************************************************
update_menu_list:
       LDA   #MENU_LINE_SIZE               ;
       LDB   GameIndex                     ;
       MUL                                 ;A*B
       LDX   #games_list                   ;X <- ptr to games_list
       LEAX  d,x                           ;X <- X+D
       LDU   #game_cur_list                ;U <- ptr to game_cur_list
       LDA   #NUMBER_OF_GAMES_ON_SCRN      ;A <- No. of entries in the menu
mov_rom2ram:
       LDB   ,x+                           ;B <- source data from ROM
       STB   ,u+                           ;B -> destination in RAM
       CMPB  #$80                          ;Search for the end-of-string
       BNE   mov_rom2ram                   ;If not found, keep copying...
       DECA                                ;Dec. No. of lines to copy
       BNE   mov_rom2ram                   ;Check if all menu lines copied

       LDX   #game_cur_list                ;X <- ptr to game_cur_list in RAM
       LDA   #NUMBER_OF_GAMES_ON_SCRN      ;A <- No. of lines in the menu
       LDB   #MENU_Y_POS                   ;B <- Ypos of the menu
update_cur_loc:
       STB   ,x                            ;B -> *X
       SUBB  #MENU_LINE_SPACE              ;B <- B-space between lines
       LEAX  MENU_LINE_SIZE,x              ;X <- X+CONST to point to next line
       DECA                                ;A <- A - 1
       BNE   update_cur_loc                ;Check if end of menu,if not keep update
       RTS                                 ;Return to main program


;*****************************************************************************
; Print the games list on the screen and return to main menu.
; The names to print are stroed in the RAM by this point in time.
;*****************************************************************************
print_names:
       LDU   #game_cur_list                ;U <- RAM location of the menu
       LDA   #NUMBER_OF_GAMES_ON_SCRN      ;A <- No. of entries in the menu
print_cur_line:
       PSHS  a,u                           ;Store A,U in the stack
       JSR   Print_Str_xy                  ;Print the current entry of the menu
       PULS  a,u                           ;Restore A and U from the stack
       LEAU  MENU_LINE_SIZE,u              ;U <- U+line size
       DECA                                ;A <- A-1
```

```
       BNE   print_cur_line              ;check if end of menu,if not keep update

       LDU   #menu_inst_text             ;U <- ptr to instruction line
       JSR   Print_Str_xy                ;Print on screen the instruction line
       RTS                               ;Return to main program


;***************************************************************************
; Draws a box arround the center of the menu to indicate selected game
;***************************************************************************
print_frame:
       JSR   Reset0Ref_D0                ;Move beam to center
       LDA   #(MENU_Y_POS-5*MENU_LINE_SPACE/2-2) ; Calc. Ypos of box
       LDB   #(MENU_X_POS)               ; Calc. Xpos of box
       JSR   Moveto_d                    ;Move beam to the Y,X pos
       LDX   #text_frame                 ;X <- ptr to the box
       JSR   Draw_VLc                    ;Draw the box
       RTS                               ;Return to main program



;***************************************************************************
; This is the ROM portion of the menu. The game list is the complete list.
; According to the GameIndex, 5 lines are copied to the RAM to be displayed.
;***************************************************************************
games_list:
        DB $00,$E5,"                ", $80
        DB $00,$E5,"                ", $80
        DB $00,$E5," 3D CRAZY CLIMBER ", $80     ;
        DB $00,$E5,"  3D MINE STORM   ", $80     ;
        DB $00,$E5," 3D NARROW ESCAPE ", $80     ;
        DB $00,$E5,"   4D ROTOCUBE    ", $80     ;
        DB $00,$E5,"  ARMOR ATTACK    ", $80     ;
        DB $00,$E5,"   ART MASTER     ", $80     ;
        DB $00,$E5,"     BEDLAM       ", $80     ;
        DB $00,$E5,"     BERZERK      ", $80     ;
        DB $00,$E5,"      BLITZ!      ", $80     ;
        DB $00,$E5,"   CLEAN SWEEP    ", $80     ;
        DB $00,$E5,"   COSMIC CHASM   ", $80     ;
        DB $00,$E5,"    DARK TOWER    ", $80     ;
        DB $00,$E5," ENGINE ANALYZER  ", $80     ;
        DB $00,$E5,"FORTRESS OF NAZROD", $80     ;
        DB $00,$E5,"     FROGGER      ", $80     ;
        DB $00,$E5,"     GALAXIAN     ", $80     ;
        DB $00,$E5,"     HEADS UP     ", $80     ;
        DB $00,$E5,"    HYPERCHASE    ", $80     ;
        DB $00,$E5,"   MELODY MASTER  ", $80     ;
        DB $00,$E5,"   MINE STORM 2   ", $80     ;
        DB $00,$E5,"   POLAR RESCUE   ", $80     ;
        DB $00,$E5,"  POLE POSITION   ", $80     ;
        DB $00,$E5,"     RIP OFF      ", $80     ;
        DB $00,$E5,"     ROM DUMP     ", $80     ;
        DB $00,$E5,"     SCRAMBLE     ", $80     ;
        DB $00,$E5,"   SOLAR RESCUE   ", $80     ;
        DB $00,$E5,"    SPACE WARS    ", $80     ;
        DB $00,$E5,"      SPIKE       ", $80     ;
        DB $00,$E5,"    SPINBALL      ", $80     ;
        DB $00,$E5,"   STAR CASTLE    ", $80     ;
        DB $00,$E5,"    STAR DEMO     ", $80     ;
        DB $00,$E5,"    STAR HAWK     ", $80     ;
        DB $00,$E5,"    STAR TREK     ", $80     ;
        DB $00,$E5,"    STAR TREK 2   ", $80     ;
```

```
        DB $00,$E5,"    TEST CART     ", $80     ;
        DB $00,$E5,"     VABOOM!      ", $80     ;
        DB $00,$E5,"    VECTRACE      ", $80     ;
        DB $00,$E5,"      VPONG       ", $80     ;
        DB $00,$E5,"     WEBWARS      ", $80     ;
        DB $00,$E5,"                  ", $80
        DB $00,$E5,"                  ", $80


;***************************************************************************
; Instruction line to be printed below the menu...
;***************************************************************************
menu_inst_text:
        DB $95,$C5,"SELECT GAME AND PRESS 4 TO START", $80


;***************************************************************************
; To identify the selected game...
;***************************************************************************
text_frame:
       FCB   3
       FCB   14,0
       FCB   0,85
       FCB   -14,0
       FCB   0,-85

;***************************************************************************
; check_btns - poll controller1 buttons 1 - 4
;***************************************************************************
menu_check_btns:
       JSR   Read_Btns                  ;Get Buttons status
       CMPA  #$00                       ;Check if a button was pressed
       BEQ   menu_return_back           ;If not, return
menu_check_btn1_1:
       BITA  #$01                       ;Check if btn1_1 was pressed
       BEQ   menu_check_btn1_2          ;If not, check btn1_2
       LDA   Vec_Prev_Btns              ;Check if btn 3 was pressed, if not,
       BITA  $04                        ;scroll 1 up, else, scroll 4 up
       BEQ   scroll_1_up
       ;;;Scroll up 4 games
       LDA   GameIndex                  ;Same idea as before but +4 instead of
       CMPA  #$04                       ;+1
       BLE   menu_return_back
       SUBA  #$04
       STA   GameIndex
       RTS                              ;Return
scroll_1_up:
       ;;;scroll up one game:
       LDA   GameIndex                  ;A <- GameIndex
       BEQ   menu_return_back           ;If A=0 no GameIndex change
       DEC   GameIndex                  ;otherwise, point to previous game
       RTS                              ;Return to main program
menu_check_btn1_2:
       BITA  #$02                       ;Check if, btn1_2 was pressed
       BEQ   menu_check_btn1_4          ;If not, check btn1_4
       LDA   Vec_Prev_Btns              ;Check if btn 3 was pressed, if not,
       BITA  $04                        ;scroll 1 down, else, scroll 4 down
       BEQ   scroll_1_down
       ;;;Scroll down 4 games
       LDA   GameIndex                  ;Same idea as before but -4 instead of
       CMPA  #(NUMBER_OF_GAMES-4)       ;-1
       BGE   menu_return_back
```

```
       ADDA  #$04
       STA   GameIndex
       RTS
scroll_1_down:
       ;;;scroll down one game:
       LDA   GameIndex                  ;A <- GameIndex
       CMPA  #(NUMBER_OF_GAMES-1)       ;Check if exeeded the last game in the
       BGE   menu_return_back           ;list. If not, point to the next game.
       INC   GameIndex                  ;otherwise, return to main program
       RTS                              ;Return to main program
menu_check_btn1_4:
       BITA  #$08                       ;Check if, btn1_4 was pressed
       BEQ   menu_return_back           ;If not, return
       JSR   start_selected_game
menu_return_back:
       RTS                              ;Return to main loop

;***************************************************************************
; This table contains the offset address within the game to start (i.e., the
; address right after the "magic init" section of each game.
; The 2nd entry is the location of the game in the big memory.
;***************************************************************************
start_loc_tbl:
       DW    D_CRAZY_START, D_CRAZY_JMP          ;Menu item #00
       DW    D_MINESTORM_START, D_MINESTORM_JMP  ;Menu item #01
       DW    D_NARROW_START, D_NARROW_START      ;Menu item #02
       DW    D_ROTOCU_START, D_ROTOCU_JMP        ;Menu item #03
       DW    ARMOR_START, ARMOR_JMP              ;Menu item #04
       DW    ART_START, ART_JMP                  ;Menu item #05
       DW    BEDLAM_START, BEDLAM_JMP            ;Menu item #06
       DW    BERZERK_START, BERZERK_JMP          ;Menu item #07
       DW    BLITZ_START, BLITZ_JMP              ;Menu item #08
       DW    SWEEP_START, SWEEP_JMP              ;Menu item #09
       DW    CHASM_START, CHASM_JMP              ;Menu item #10
       DW    DARKT_START, DARKT_JMP              ;Menu item #11
       DW    ENGINE_START, ENGINE_JMP            ;Menu item #12
       DW    FNARZOD_START, FNARZOD_JMP          ;Menu item #13
       DW    FROGGER_START, FROGGER_JMP          ;Menu item #14
       DW    GALAXIAN_START, GALAXIAN_JMP        ;Menu item #15
       DW    HEADSUP_START, HEADSUP_JMP          ;Menu item #16
       DW    HYPER_START, HYPER_JMP              ;Menu item #17
       DW    MELODY_START, MELODY_JMP            ;Menu item #18
       DW    MINE2_START, MINE2_JMP              ;Menu item #19
       DW    POLAR_START, POLAR_JMP              ;Menu item #20
       DW    POLE_START, POLE_JMP                ;Menu item #21
       DW    RIPOFF_START, RIPOFF_JMP            ;Menu item #22
       DW    ROM_START, ROM_JMP                  ;Menu item #23
       DW    SCRAMBLE_START, SCRAMBLE_JMP        ;Menu item #24
       DW    SOLAR_START, SOLAR_JMP              ;Menu item #25
       DW    SPACEWAR_START, SPACEWAR_JMP        ;Menu item #26
       DW    SPIKE_START, SPIKE_JMP              ;Menu item #27
       DW    SPINB_START, SPINB_JMP              ;Menu item #28
       DW    SCASTLE_START, SCASTLE_JMP          ;Menu item #29
       DW    STARDEMO_START, STARDEMO_JMP        ;Menu item #30
       DW    SHAWK_START, SHAWK_JMP              ;Menu item #31
       DW    STREK_START, STREK_JMP              ;Menu item #32
       DW    STREK2_START, STREK2_JMP            ;Menu item #33
       DW    TEST_START, TEST_JMP                ;Menu item #34
       DW    VABM_START, VABM_JMP                ;Menu item #35
       DW    VRACE_START, VRACE_JMP              ;Menu item #36
       DW    VPONG_START, VPONG_JMP              ;Menu item #37
       DW    WEBWAR_START, WEBWAR_JMP            ;Menu item #38
```

```
;*****************************************************************************
; The following section puts the predefined games into their designated memory
; location.The .inc file is basically the BIN file, converted to FCB format +
; the 1F 05 modification.
; This is done till address 0xFFFF (64K) since the compiler doesn't support
; more than that...
;*****************************************************************************
       ORG  0x1000
       INCLUDE "src/multi/games/armor.inc"

       ORG  0x2000
       INCLUDE "src/multi/games/art.inc"

       ORG  0x3000
       INCLUDE "src/multi/games/bedlam.inc"

       ORG  0x4000
       INCLUDE "src/multi/games/berzerk.inc"

       ORG  0x5000
       INCLUDE "src/multi/games/rotcub.inc"

       ORG  0x6000
       INCLUDE "src/multi/games/castle.inc"

       ORG  0x7000
       INCLUDE "src/multi/games/rom.inc"

; Fill the jump table with NOP instruction - no real use but just "cleaner"
       ORG 0x7f00
jmp_table:
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12
       FCB $12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12,$12

       ORG  0x8000
       INCLUDE "src/multi/games/chasm.inc"

       ORG  0x9000
       INCLUDE "src/multi/games/hyper.inc"

       ORG  0xA000
       INCLUDE "src/multi/games/mine.inc"

       ORG  0xB000
       INCLUDE "src/multi/games/ripoff.inc"
```

```
        ORG  0xC000                ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0xD000
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0xE000
        INCLUDE "src/multi/games/solar.inc"

        ORG  0xF000
        INCLUDE "src/multi/games/space.inc"
```

The following section includes the source code for the following 64K segments - all the way up to 512K:

Bank2.asm

```
;1st 64K bank, will be placed at 0x10000
;Where 4K games continue

        ORG  0x00000
        INCLUDE "src/multi/games/starhawk.inc"

        ORG  0x01000
        INCLUDE "src/multi/games/startrek.inc"

        ORG  0x02000
        INCLUDE "src/multi/games/sweep.inc"

        ORG  0x03000
        INCLUDE "src/multi/games/test.inc"

        ORG  0x04000
        INCLUDE "src/multi/games/trek2.inc"

        ORG  0x05000
        INCLUDE "src/multi/games/vectrace.inc"

        ORG  0x06000
        INCLUDE "src/multi/games/vpong.inc"

        ORG  0x07000      ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0x08000      ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0x09000      ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0x0A000
        INCLUDE "src/multi/games/engine.inc"

        ORG  0x0B000      ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0x0C000      ;PLACE HOLDER
        INCLUDE "src/multi/games/scramble.inc"

        ORG  0x0D000      ;PLACE HOLDER
```

```
      INCLUDE "src/multi/games/scramble.inc"

      ORG  0x0E000      ;PLACE HOLDER
      INCLUDE "src/multi/games/scramble.inc"

      ORG  0x0F000      ;PLACE HOLDER
      INCLUDE "src/multi/games/scramble.inc"
```

Bank3.asm

```
;will be placed at 0x20000
;Where 8K games start

      ORG  0x0000

      INCLUDE "src/multi/games/3dczycst.inc"

      ORG  0x2000
      INCLUDE "src/multi/games/blitz.inc"

      ORG  0x4000
      INCLUDE "src/multi/games/stardemo.inc"

      ORG  0x6000
      INCLUDE "src/multi/games/headsup.inc"

      ORG  0x8000
      INCLUDE "src/multi/games/melody.inc"

      ORG  0xA000
      INCLUDE "src/multi/games/mine3.inc"

      ORG  0xC000
      INCLUDE "src/multi/games/narrow.inc"

      ORG  0xE000
      INCLUDE "src/multi/games/nazrod.inc"
```

Bank4.asm

```
;will be placed at 0x30000
;Where 8K games continue


      ORG  0x0000             ;PLACE HOLDER
      INCLUDE "src/multi/games/polar.inc"

      ORG  0x2000
      INCLUDE "src/multi/games/polar.inc"

      ORG  0x4000
      INCLUDE "src/multi/games/pole.inc"

      ORG  0x6000
      INCLUDE "src/multi/games/spike.inc"
```

```
      ORG  0x8000
      INCLUDE "src/multi/games/spinball.inc"

      ORG  0xA000             ;PLACE HOLDER
      INCLUDE "src/multi/games/webwars.inc"

      ORG  0xC000
      INCLUDE "src/multi/games/webwars.inc"

      ORG  0xE000             ;PLACE HOLDER
      INCLUDE "src/multi/games/webwars.inc"
```

Bank5.asm

```
;will be placed at 0x40000
;Where 16K games start

      ORG  0x0000
      INCLUDE "src/multi/games/vaboom.inc"

      ORG  0x4000             ;PLACE HOLDER
      INCLUDE "src/multi/games/vaboom.inc"

      ORG  0x8000
      INCLUDE "src/multi/games/darktowr.inc"

      ORG  0xC000
      INCLUDE "src/multi/games/spikesh.inc"
```

Bank6.asm

```
;will be placed at 0x50000
;Where 16K games continue

      ORG  0x0000             ;PLACE HOLDER
      INCLUDE "src/multi/games/frogger.inc"

      ORG  0x4000
      INCLUDE "src/multi/games/galaxian.inc"

      ORG  0x8000
      INCLUDE "src/multi/games/frogger.inc"

      ORG  0xC000             ;PLACE HOLDER
      INCLUDE "src/multi/games/frogger.inc"
```

Bank7.asm

```
;will be placed at 0x60000
;Where 32K games start

      ORG  0x0000       ;PLACE HOLDER
      INCLUDE "src/multi/games/spectrum.inc"

      ORG  0x8000       ;PLACE HOLDER
```

```
      INCLUDE "src/multi/games/spectrum.inc"
```

Bank8.asm

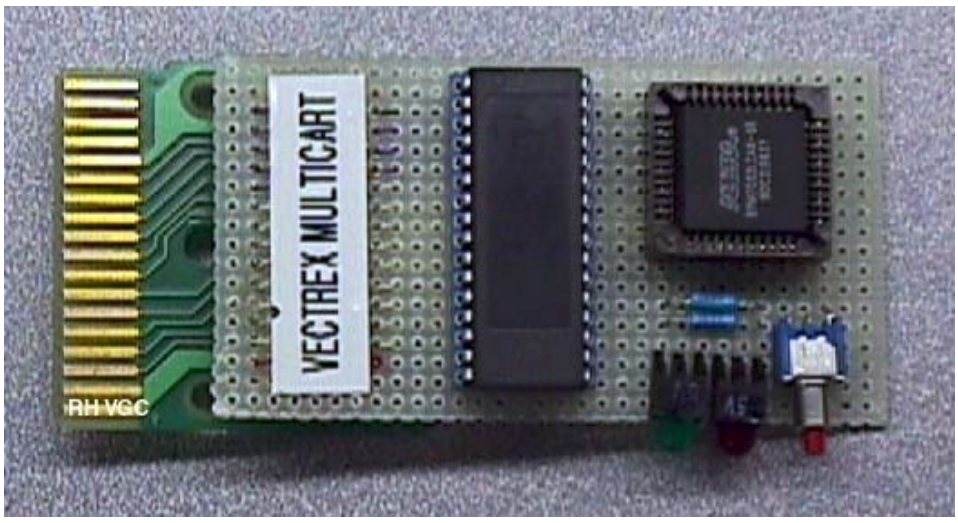
```
;will be placed at 0x70000
;Where 32K games continue

      ORG  0x0000       ;PLACE HOLDER
      INCLUDE "src/multi/games/spectrum.inc"

      ORG  0x8000       ;PLACE HOLDER
      INCLUDE "src/multi/games/spectrum.inc"
```

The last step involved in making the final BIN is merging all the 8 banks together into a single 512K file. Once that is achieved an EPROM (or Flash) can be programmed and the "brains" of the multicart is pretty much ready.

Building a prototype

In order to build my prototype I've used an existing Vectrex cartridge PCB as is with no modifications. The EPROM socket was replaced by a small board that had EPROM-like pins on one side and the rest of the components on its other side. The EPROM's signals were routed by soldering wire-wrap wires between the EPROM, Flash and EPLD as described in the previous sections.

Top view of the prototype is shown below (please ignore the switch and the LEDS - the switch was designed to be a PAUSE buts due to lack of available I/O on the EPLD it has been left unconnected) :



Price

The total price of the components is as follows:
1 x EPM7032-LC44-10 - $1.75
1 x AM29F040-PC120 - $13.1 - for development only, or, 1 x 27C040-PC120 - $7.5 - for final product
Misc. (wires/sockets/board) - no more than $5

The total comes to $19.85 for development cart or $14.25 for end product (This doesn't include case and dedicated PCB for this project, which I am not going to make) - In my opinion, at least the goal of "cheap to build" has been achieved…

Summary

This is a demonstration of one possible way to put together a Vectrex multicart. I'm sure that there is more than one way to achieve the same goal - maybe even in a simpler fashion. However, this is my attempt that is proofed to be working (on the single console I've tried it on). I'm sure that this document didn't cover all the aspects of the hardware and software design but I'm confident that with sufficient background and with enough time spent understanding this concept everybody can benefit from it.
In addition, reading between the lines of this document should reveal that 32K games might have a problem when executed in this version of the multicart (EPLD). The reason is that a 32K game can access address 0x7Fxx - which is a valid address in such a large game. In that case the EPLD gets confused and latches wrong address and the whole operation gets corrupted. This is not the case, if the game is smaller than 32K-256 Bytes. The way to solve that is to add a tiny state-machine in the EPLD that will allow the latch-enable generation only when a specific sequence of addresses has been accessed. That was also implemented but is not shown here.
And, one last thing: As stated before, my intentions are to demonstrate a feasible way to implement a Vectrex multicart. As such, I'm not going to sell or provide any related material for this type of project. Also, the .inc files and source code are not going to be available in any form other then the way they have been presented here.

Thanks for reading that far,
Keep on gaming and keep the Vectrex alive,
Ronen Habot,
June 2000.