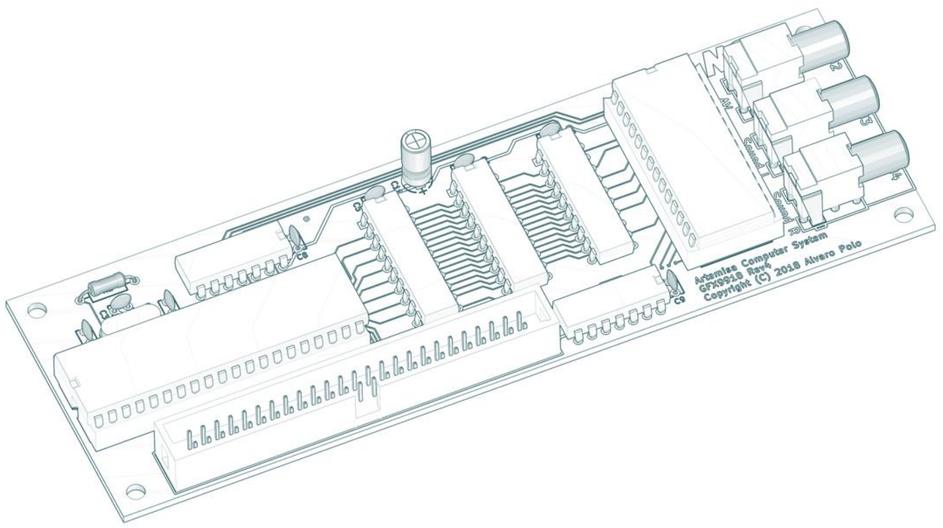
Artemisa Computer System Homebrew DIY (Model 101) Technical Handbook



Rev04

Table of Contents

"640K ought to be enough for anybody"

(Attributed to) Bill Gates, 1981

Introduction

Welcome to the early days of home computing.

Let me propose you a journey to the 80s of past century. During this decade, the computing industry experienced a big revolution. The lessons learned during the 70s talk about a future in which computers domain everything. A future in which computers are everywhere. Including our homes.

In these days, many companies produced home computers. This is the age of the Spectrum ZX, the Commodore 64, the Amstrad CPC, and of course the big family of MSX computers. Small machines with really limited capacity and resources had their place in many homes when we were just kids. We learned to play videogames with these early computers. We learned to code with them. We learned their potential. Their power to change the modern economies, to transform the World. And we learned all that was just the beginning.

Again, welcome to the early days of home computing. Welcome to Artemisa.

What is Artemisa

Artemisa is an 8-bit computer system that conforms the MSX-1 specification. There were many hardware vendors who produced MSX computers in the 80s. Sony, Canon, Casio, Sharp, Spectravideo, Panasonic, Philips, … Artemisa is just another member of this big family, with some particularities.

One of these particularities is that Artemisa design began in 2018. That is 35 years after the initial design of the first MSX computers. But this is not new. In 2006, ESE Artists' Factory designed the 1chipMSX. This is a FPGA-based computer system that implements the MSX-2 standard. In 2013, a Korean team of MSX enthusiasts known as Retroteam Neo created the Zemmix Neo as an evolution of the 1chipMSX design with inspiration from the classic Zemmix game console produced by Daewoo. Since then, many other teams along the globe had produced their own customizations of Zemmix Neo. Also, in 2020 a group of Spanish makers released the MSXVR. This is an enhanced MSX emulated by a Raspberry Pi with the physical appearance of a classic 8-bit computer.

So yes, Artemisa is just another member that joins the club of MSX computers designed in 21st century. But here comes another particularity. Artemisa is one of the few, along the Omega MSX, that do not use emulation techniques. Some other products as 1chipMSX or Zemmix Neo use FPGAs to emulate the whole MSX hardware. Some others as MSXVR or the Zemmix Mini uses software emulation on top of an ARM-based computer. In contrast, Artemisa (and Omega) use real integrated circuits and discrete logic. In other words, it is made using the same design techniques and materials used in the 80s, when the first MSX computers were also designed. For example, the heart of the Artemisa computer is a real Zilog Z80 microprocessor instead of a Z80 CPU reimplemented in an HDL language by reverse engineering and synthetized in an Altera Cyclone FPGA.

Please note this does not mean Artemisa is better (or worse) than its FPGA-based or software-based counterparts. It is likely that all the things you can do with Artemisa can also be done in any other MSX computer system, emulated or not. You will not observe any remarkable additional feature, more speed or more stability. All they are functional MSX computers you can enjoy.

Why Artemisa

You might ask why you would need an Artemisa computer considering there are some other options out there. Good question!

One possible answer is that you might find Artemisa computer more authentic. Please note I am not saying *it is* more authentic. But just you might find so. There are some users that specially appreciate the real hardware. And they feel more comfortable using an MSX computer that uses the classic technologies, implemented with a real CPU, real VDP, real PSG, real memory, … A computer that is designed using the old-school techniques. If you are such a user, you will specially enjoy Artemisa.

In the case of Artemisa Homebrew DIY series, there is another reason for choosing it. You have the chance to **learn how an MSX computer is designed** from the top to the bottom. And I am not just saying you will have to assemble it. Thanks to this book, you will also have the chance to understand how every single chip works. You have access to the schematic designs of every single circuit and an educational guide about their operation. This is a complete lecture about the architecture and design of MSX computers. A book that will put in your hands the tools to understand how a microcomputer works under the hoods. If you are interested in computer architectures beyond the theory, you will find Artemisa was made for your needs.

In summary, the main reasons to choose Artemisa is because you **feel the real hardware is particularly sexy** and/or you want to **learn how an MSX computer works**. If you only want to have an MSX machine, probably other options are better. On one hand, a refurbished MSX or a new FPGA-based one are likely less expensive. On the other hand, some other homebrew designs such as the Omega are more powerful and MSX2-compliant, but they lack the extensive documentation to understand their design. The offer of Artemisa is educative. The **goal is to make you learn** computer architecture and design at the same time you build and assemble your own MSX computer.

About this document

As stated above, this handbook compiles all the technical knowledge about the Artemisa Homebrew DIY Computer series. You will find it useful to assemble the parts of the DIY kit. But also, to understand how an MSX computer works in detail. From the smaller chip to its most complex circuit.

Intended audience

If you have purchased an Artemisa Homebrew DIY Computer series, you are likely an electronic hobbyist or an MSX fan and you feel skilled enough to assemble a computer system by your own. This means soldering the components following the detailed instructions you will find in this book.

The Artemisa model 101 is built using electronic components with large through-hole packages. Its integrated circuits are DIP (Dual Inline Package), with a pad separation of 2.54mm. Other components such as resistors and capacitors are also large ones. This resembles the design style of early 80s computers. But also lowers the entrance barrier for those that do not trust their own soldering skills. The grade of difficulty of assembling an Artemisa computer model 101 is low.

Thus, only a **basic knowledge in soldering** is required. If unsure, there are tons of guides and YouTube tutorials available online to learn from. Assembling this kind of electronic components is easy. Just forget your fears and enjoy the experience!

Also, if you want to take advantage of the educational opportunities of this book, **some basic knowledge of computing and digital systems** is required. The user is assumed to understand basic concepts such as bit, byte, memory address, bus, CPU, peripheral, logic gate, voltage level, etc. A degree in Computer Science is useful but not necessary.

How to Read This Book

There are some parts of the book that have a special meaning or must be read in some specific way. They are described in this section.

Theory of operation blocks

These blocks will be represented as follows:

Theory of Operation

Here goes a full detailed description of how a part of the circuit works.

The theory of operation blocks will describe the theory behind a part of the system. A basic knowledge on computing and digital systems is required in order to fully understand these blocks. However, do not worry if you lack this knowledge or find it difficult to understand. This information is not essential to assemble the system. You can still build your Artemisa computer without fully understanding what these text blocks tell. This is just for educational purposes.

Warning blocks

Some operations require special attention to things that, when unnoticed, could cause damage in the circuit. In these cases, a warning text block will indicate you have to pay special attention, like this one:

⚠ This is a warning message that requires your attention.

Schematic diagrams

Along this document, the different parts of the circuit are explained using schematic diagrams. They are not difficult to understand. But if they are totally unfamiliar to you, a brief introduction might be useful.

Most of the elements you will find in a schematic diagram, shown in **¡Error! No se encuentra el origen de la referencia.**, are described as follows:

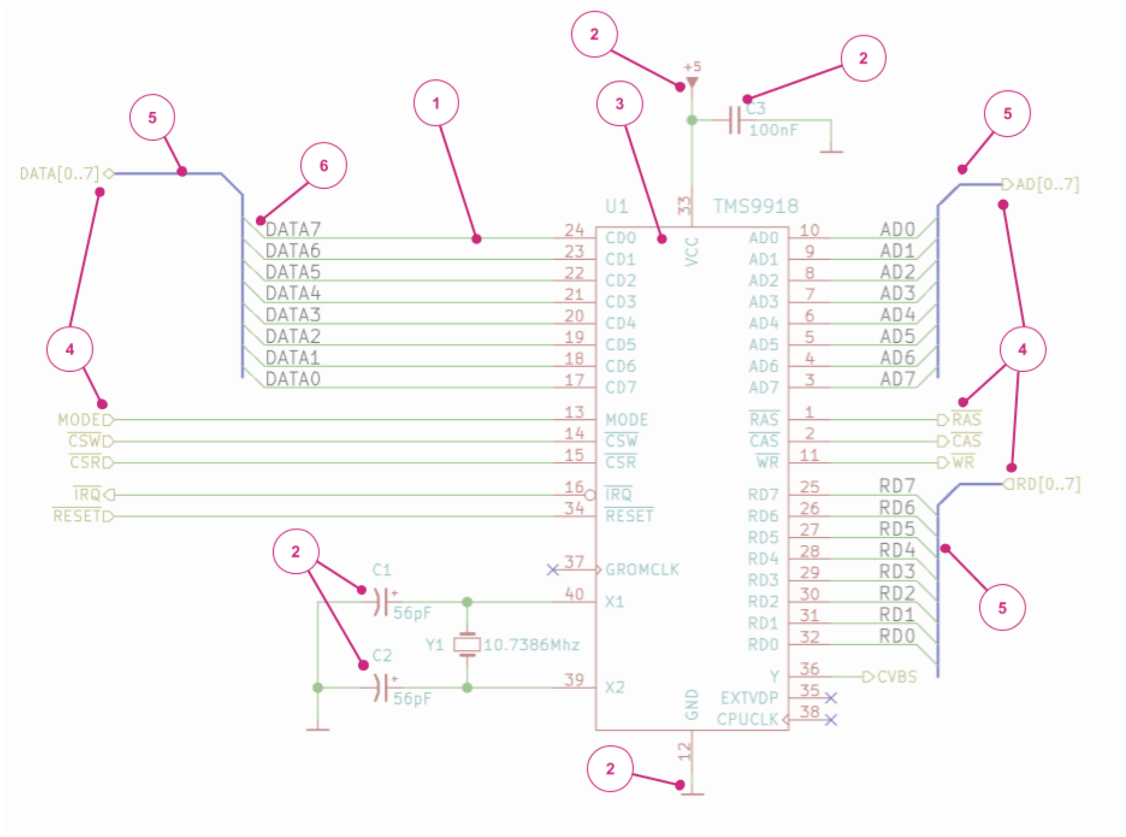


Figure 1: Elements of a schematic diagram

1. Wires. This is the most building block of a schematic diagram. It connects two points electrically. This means there will be a copper path between them that will let the current flow.
2. Basic components. Elements that have one or more pins where wires can be connected. These are power supply terminals, capacitors, resistors, oscillators, logic gates, etc. The different symbols used in this manual are described in Table 1.
3. Integrated circuits. Components that contain a whole and complex circuit inside it. These are processors, memories, flip-flops, decoders, etc.

4. Connections from/to other schematic diagrams. The pointing edge indicates the direction of the signal. If it has two pointed edges, it is bidirectional.
5. Buses. Collections of parallel wires that transmit a multibit data.
6. Wire connections from/to buses. A label indicates what is the wire that is obtained from the bus.

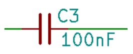
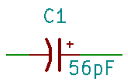
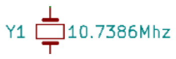
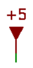
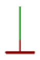
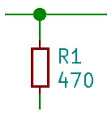
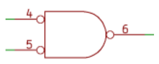
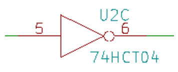
	Ceramic capacitor
	Electrolytic capacitor
	Clock oscillator
	5 volts power supply
	Ground power supply
	Resistor
	AND logic gate
	NOT logic gate

Table 1: Component symbols in schematic diagrams

If you still do not know what a bus, a capacitor or a logic gate is, do not panic. Most of these concepts will be elaborated in the chapter Artemisa Computer System.

Number notations

Some numerical values are expressed along this manual. The prefix formats shown in Table 2 are used to distinguish between different integer bases.

Prefix	Description
0b	Binary number format. Example: 0b01100011.
0x	Hexadecimal number format. Example: 0x4A0129FE.

Table 2: Number format prefixes

Artemisa Computer System Architecture

Overview of an MSX computer

We can start this book providing a bird's sight of how an MSX computer is made.

Most 8-bit microcomputers are amazingly simple. They include a **CPU**, a few tens of kilobytes of **memory** and some **peripherals**. All of them connected through a **system bus**, as shown in Figure 2.

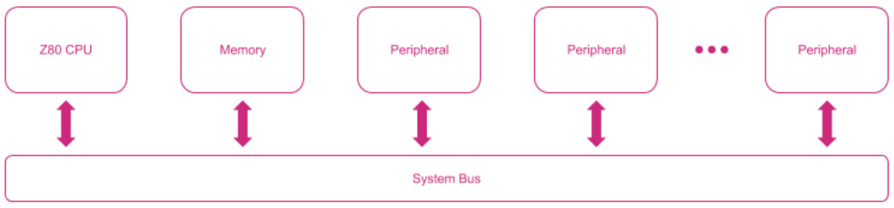


Figure 2: High level architecture of an MSX computer

Electronic buses

The basic operation of any computer is to move information around. The information is physically represented by electrical signals that may have or not voltage and current. We could simplify this idea by saying that the presence of electrical current and voltage represents the digital number 1, while the absence of it represents the 0. Hence, every single wire can transfer one bit of information at any time.

A bus in electronics is just a group of wires that have some common purpose. There are some buses that tie the wires together to represent multi-bit magnitudes. For example, if we put 8 wires together, we will have the ability to represent 8-bit numbers. This is a value from 0 to 255 (or -127 to 128). With 16 wires we will represent values from 0 to 65535 (or -32768 to 32767). And so on. Some other times we put wires in the same bus because they represent some information that is closely related. For example, some control signals can be grouped together in the same bus to communicate control actions from one device to another, such as read, write, reset, interrupt, etc.

If we make an analogy between a computer and the human body, the system bus is the nervous system. In our bodies, it is the medium to transfer the signals from the brain to the muscles to coordinate our actions. But it is also the channel to carry the sensitive information from our five senses to perceive the world. In our computers, the CPU is the brain and the muscles and senses are the peripherals. And all them use the system bus as a kind of nervous system to transfer information in both directions.

The microprocessor

The CPU of an MSX computer is a Zilog Z80 microprocessor running at 3.58Mhz. The Z80 was introduced by Zilog in 1976 as an enhancement and direct competitor of the popular Intel 8080. It was binary compatible with the Intel CPU, which means it can execute programs originally created for the 8080. Thanks to this and its extended instruction set, it was positioned as one of the most successful CPUs of all the times. It was chosen by Sinclair to make most of their computers, including the top selling ZX Spectrum. It was also the CPU behind the Amstrad CPC series, and some other game consoles such as the ColecoVision and the Sega Master System. And, of course, it is the brain of any MSX computer system.

The memory

The memory system of an MSX system is comprised by RAM (typically 64KB) and ROM (typically 32KB) memory. The ROM memory is a read-only memory whose content is written during the manufacturing process. It is used to store the MSX

BIOS and the MSX Basic interpreter. The RAM memory, in contrast, can be read and write. It is used to store the programs that are loaded from storage devices such as the cassette tape or the floppy drive, and to store the data used by these programs. The RAM memory is volatile, which means its contents will be lost after power off or reset. In contrast, the ROM memory is persistent. Once written during the manufacturing process, its contents will persist until the end of the days or until it is destroyed. The thing that happens first.

However, the memory system of the MSX is quite complex. It has a sophisticated bank switching mechanism to break the limit of 64KB of memory space of the Z80 CPU, as we will see later in detail in page 23. Hence, it uses a not trivial decoding and slot selection logic to interpret the signals coming from the system bus and the Parallel Peripheral Interface (PPI) in order to identify the memory device when it is addressed, as shown in Figure 3. Do not worry about PPI for now, we will cover this later.

The MSX BIOS

BIOS is the acronym of Basic Input/Output System. This is an essential software program present in any computer system that serves two purposes.

First, it is the first software the CPU will execute upon boot. Thus, it is responsible of preparing the system before the first program can run. Here preparing the system means initializing the hardware into a consistent state, perform a few sanity-checks to ensure the hardware is not malfunctioning and locate the first program that will run. This first program is typically an operating system. In our case, the MSX Basic or the MSX DOS.

The other purpose is to provide a software abstraction layer over the hardware. The BIOS is a sort of software library that provides utility routines the programmer can use to interact with the hardware. This not only simplifies the life of the programmers, but also ensures the portability of their programs. The BIOS hides the divergencies of the hardware behind an API (Application Programming Interface). So in case of a computer having a different hardware, it is possible to reprogram its BIOS to deal with the differences so that the user software can also run on it without any change.

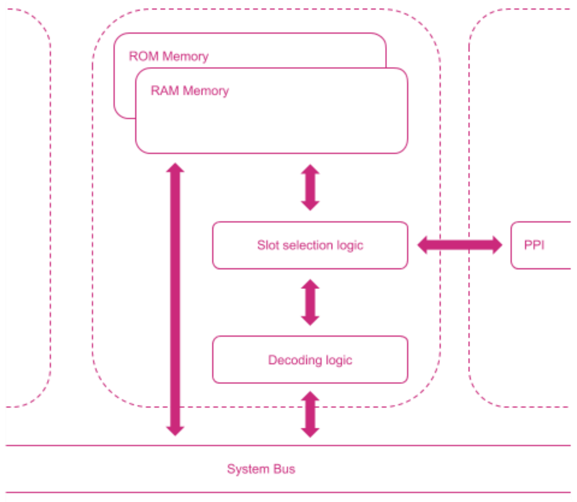


Figure 3: High level architecture of MSX Memory

The Video Display Processor

Unfortunately, a computer made of a CPU and memory is a useless junk that only serves to dissipate heat. It needs some peripherals to interact with the users, receiving data and programs as inputs and showing the result of these programs as outputs.

Perhaps the most important peripheral is the Video Display Processor (VDP). This is a circuit responsible of producing the video display in the computer. In the MSX standard, this is done by a chip of the Texas Instruments TMS9918 family, as shown in Figure 4. This chip encapsulates all the logic necessary to display images in a TV set or suitable monitor. It is

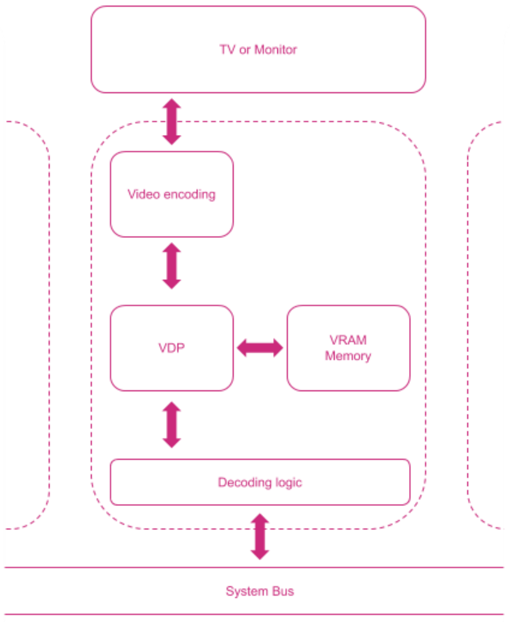


Figure 4: High level architecture of MSX Video Display Processor

connected to the system bus through a decoding logic to adapt the signals. One of its main particularities is that it uses its own separated video RAM memory to store the images and the sprites it will display in the screen.

It is worth to remark this distinction of the VDP in an MSX computer. Almost any other system uses a video display that shares the RAM memory with the CPU. The computer programs must write the data of the images they want to display into a specific section of the main RAM memory. And the display processor will read from that memory area to draw the images into the screen.

As you might think, sharing the memory between two different devices that can potentially access to the same addresses at the same time is not easy. This is known as memory contention. And it is full of tradeoffs. Consider that the video display cannot stop doing its job because the memory is being used by the CPU. If it does, we would observe artifacts in the screen such as flickering images caused by the interruption of the video stream. In some cases, this is addressed by finding the perfect timing on the accesses to the system bus, as Steve Wozniak did for the Apple][. In some other cases, the CPU is forced to wait when memory contention occurs, losing precious CPU cycles and impacting the system performance. This is the case of the ZX Spectrum and the Amstrad CPC computers.

In contrast, the MSX computers strictly separates the system RAM from the video RAM. The programs only access the VRAM indirectly by sending I/O requests to the VDP to perform read or write operations on the VRAM. The VDP coordinates the access to the memory for both: read the bitmaps to display the screen and process the read or write requests coming from the CPU. The result is a simpler computer architecture without memory contention that do not sacrifice CPU cycles.

I/O Requests

The Z80 microprocessor, as many others, includes the concept of I/O operations in its design. I/O is just an acronym of Input/Output. The CPUs that embrace this concept implement a specific way to communicate to and from system peripherals by means of an I/O request.

An I/O request is a communication initiated by the CPU whose destination is a specific device attached to the system bus. The request can be one of two types: a read operation to transmit a byte from the device to the CPU, or a write operation to transmit a byte from the CPU to the device. The device that must respond to the request is identified by an 8-bit number known as the *I/O port*. The CPU uses the signals of the system bus to indicate the start of an I/O request, the port of the device that must respond to it, and whether it is a read or a write operation. The byte that is exchanged between the CPU and the device is also transferred throw the system bus.

For example, if the CPU wants to write the byte 0xAB to the VRAM address 0x1234, it will have to follow this sequence of I/O requests:

- I/O request to port 0x99 to write the byte 0x34. This will tell the VDP the least significant bits of the VRAM address that will be written.
- I/O request to port 0x99 to write the byte 0x52. This is the six most significant bits of the VRAM address (0x12) that will be written, prefixed with 0b01. These two bits prefix indicate to the VDP must be prepared for a VRAM write operation next.
- I/O request to port 0x98 to write the byte 0xAB. This is the byte the VDP will write to VRAM address 0x1234.

Some other CPUs, such as the MOS 6502 or the Motorolla 68000 do not implement the concept of I/O request. These CPUs uses memory access instructions equally to access the memory or access other bus devices. This means part of the memory address space must be reserved for I/O operations.

The parallel peripherals

Once we can display the results of the programs our MSX computer executes, it is time to think about receiving some inputs. One of the most important pieces used in the MSX architecture to achieve this is the Parallel Peripheral Interface (PPI), shown in Figure 5.

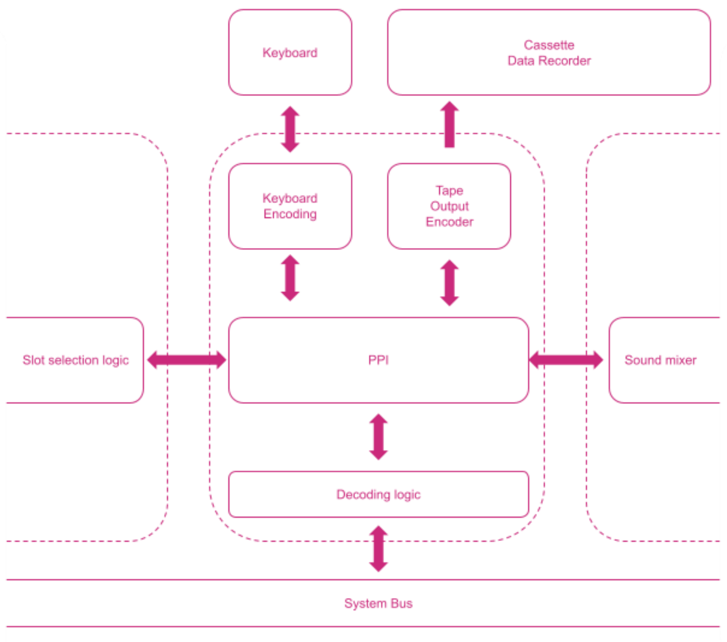


Figure 5: High level architecture of parallel peripherals

The PPI is an Intel 8255 integrated circuit that will act as hardware controller for some peripherals and a few control lines of the system. This is the same chip included in the IBM PC computer, still present in the chipsets of most of modern PC designs. The 8255 has three 8-bit ports: A, B and C. Each port can be configured by software as input or output. The MSX BIOS will configure each port at convenience on system boot.

The system peripherals that are attached to the system through the PPI are connected by wires to some lines of its three ports, as shown in Figure 6. The mission of this chip in the MSX is to serve as an interface adapter for the peripherals that

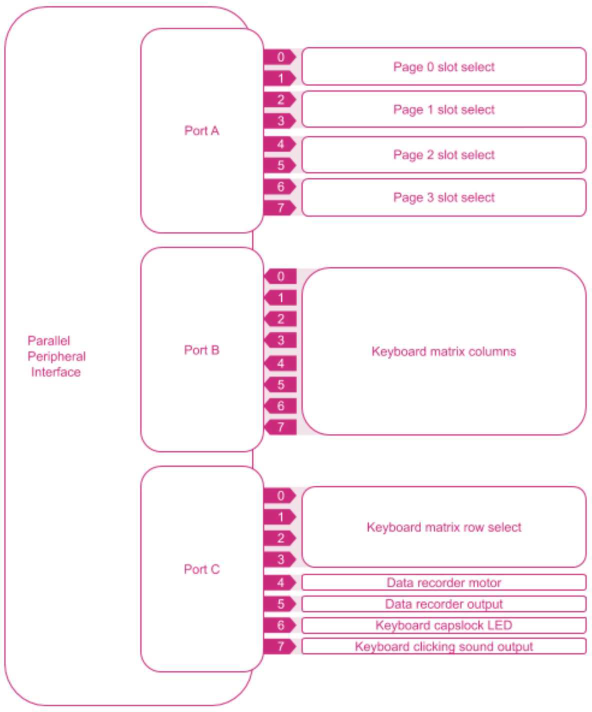


Figure 6: PPI Ports

cannot be directly connected to the system bus, such as the keyboard and the cassette data recorder. Using I/O requests, the CPU can read from the input ports and write to the output ports of the PPI. Which is analogous to read or write from or to the peripherals attached to the ports.

The port A of the PPI is configured by the BIOS as output during the system boot. This port is responsible of choosing the memory slot to be used for every memory page. Each pair of bits determine the primary memory slot of a memory page. Since two bits are used per memory page, we can choose among 4 different primary memory slots. Do not panic if you still do not understand a word about this. It is a complex subject we will cover in detail later.

The port B of the PPI is configured by the BIOS as input. All eight bits are connected to the columns of the keyboard matrix. The port C is configured as ouput. The four less significant bits are used to select one row of the keyboard during the scanning process. Bits 4 and 5 are used to control the data recorder. Bit 6 is used to drive the LED of the capslock key of the keyboard and bit 7 is used to produce clicking sounds when keys are typed.

As you can see, the most important peripheral attached to the PPI is the keyboard. This is the time to tell how it works in an MSX computer and what is the meaning of these rows and columns thing. The keyboard of the MSX is implemented as a X-Y matrix of 8 columns and up to 12 rows, as shown in Figure 7. The total number of rows depends on the keyboard distribution. There is a key switch that connects every row to every column of the matrix. When a key is pressed, the row and the column connected by the switch form a closed circuit. When this happens, any current driven through the wire of the row will propagate to the column. Otherwise, if the key is not pressed, we will have an open circuit and the column will have no current.

In order to make it work, the MSX uses the PPI to drive the row signals of the matrix and read the status of the columns. The four less significant bits of output port C of the PPI are used to encode the row that will drive the current. And the whole eight bits of input port B are connected to the matrix columns. The system software, typically the BIOS, will scan the matrix periodically. On each scan, it will loop from the first to the last row. In each loop iteration, it will select the i-th matrix row and read the status of the 8 columns. The row can be selected by writing its 4-bit number in the corresponding bits of the port C of the PPI through an I/O request. The columns can be checked by reading the port B of the PPI through an I/O request. If that read of port C reveals the current is driven in the j-th column, it means the key located at i-th row and j-th column in the matrix is pressed. Otherwise, it is depressed.

The bit 6 of PPI port C is also involved in the keyboard operation. It drives the LED of the capslock key. As you might think, this is controlled by software. It can be turned on and off by setting 0 and 1 in that bit of port C, respectively.

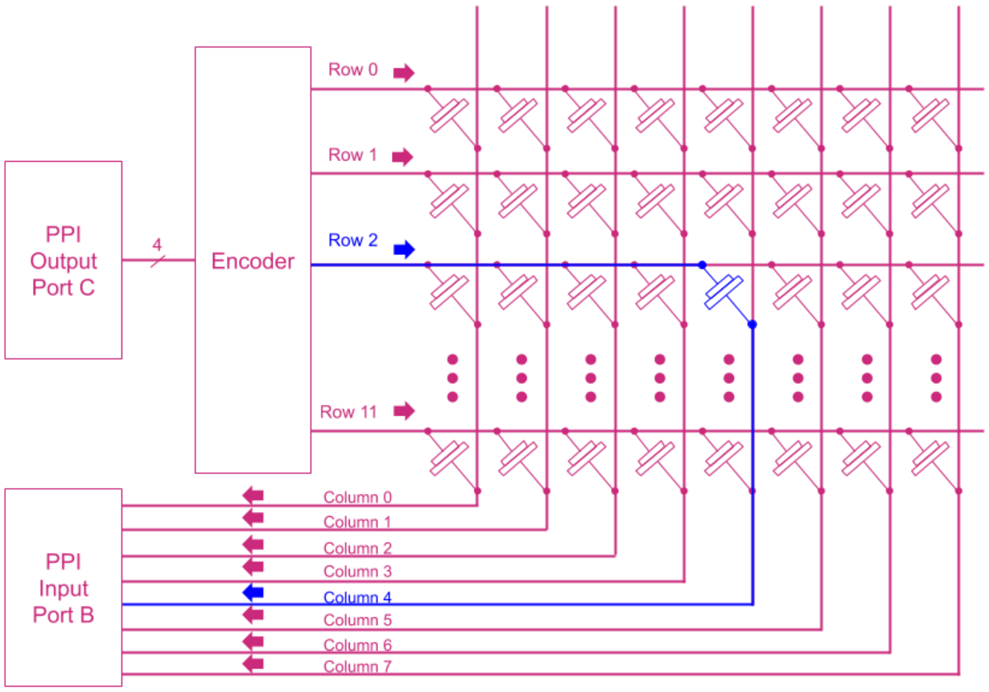


Figure 7: Keyboard matrix

The bits 4 and 5 of the PPI port C are connected to the cassette data recorder. The data recorder is an unexpensive storage device that uses cassette tapes to transmit information to and from the computer. It is a cassette player that reproduces the sounds from the tape when reading and records the sound signal on the tape when writing. The sound signals encode binary data in a special kind of frequency modulation known as FSK (Frequency-Shift Keying) following the Kansas City standard. This norm establishes that a “0” bit is encoded by one wave cycle at a frequency of 1200 Hz. While a “1” bit is encoded by two cycles of a 2400 Hz wave. The sound wave reproduces a sequence of beeps alternating between 1200 and 2400 Hz, which in fact represents zeroes and ones that are transmitted from or to the computer.

In the MSX, all this encoding is generated by software. The bit 5 of the PPI port C is used to generate a squared waveform when writing a file to the data recorder. This is made typically by the MSX BIOS by writing zeroes and ones in that bit of the PPI port one by one. Fortunately, this FSK encoding system operates at low frequencies, below 4800 Hz in the worst case. The Z80 processor at 3.58Mhz can write these sequences of bits fast enough to ensure the waveform is well formed.

The wave generated through bit 5 of PPI port C is squared because the signals of the PPI ports are digital. Thus, they can only generate +5 and 0 voltage levels, as shown in Figure 8. There is a tape output encoding circuit that transforms this squared wave into a sinusoidal waveform. Which is what the data recorder expects from the computer. The process to read a file is similar. But it involves other peripherals we will cover soon.

The bit 4 of the PPI port C is also involved in the data recording process. It can be used to start and stop the motor of the data recorder automatically when a read or a write operation on a file begins. Some data recorders support this feature.

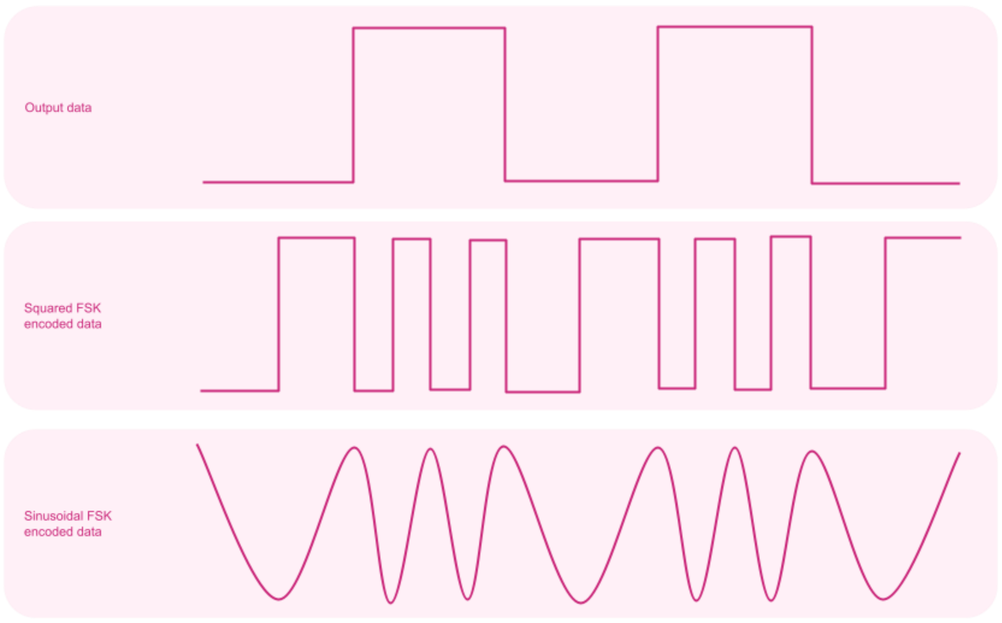


Figure 8: FSK encoding in data recorder

Programmable Sound Generator

A computer without a sound system is like a garden without flowers. And we already know the MSX is one of the best in its category. Of course, it has a programmable sound generator to play sounds and music.

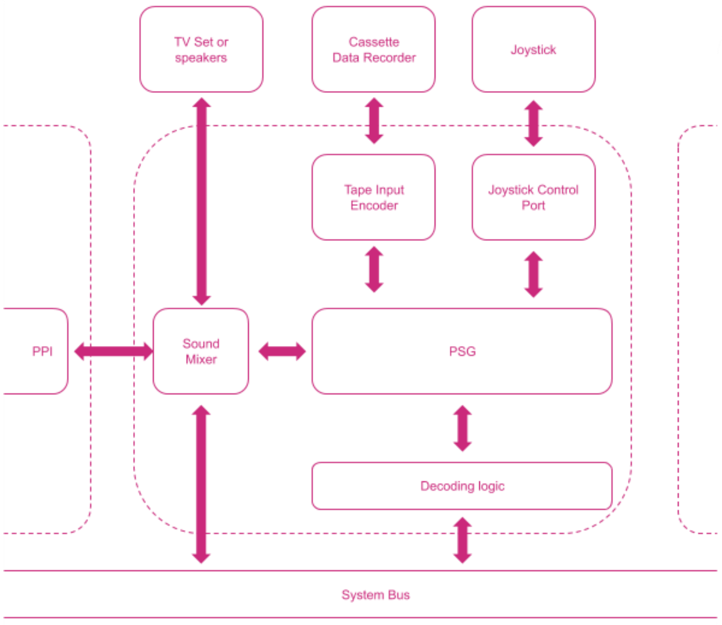


Figure 9: Programmable Sound Generator

This is implemented by the General Instrument AY-3-8910. This is a chip that can generate three independent sound channels programmed by software. It was released in 1978 and used in innumerable arcade machines and microcomputers. Including the ZX Spectrum, the Amstrad CPC and the Apple][.

The 8910 is attached to the system bus of the MSX through a decoding logic to adapt its interface, as shown in Figure 9. Such interface give access to three different I/O ports: the address port, the data write port and the data read port.

The address port is used to indicate the address of the register that will be read or written. The PSG has 16 ports numbered from 0 to 15. Writing the PSG port number in I/O port 0xA0 will prepare the chip to use that port in subsequent read or write operations.

The PSG data write port as available at I/O port 0xA1. This port is used to write bytes to the PSG register that was addressed by previous writes to 0xA0. The PSG data read port at 0xA2 does the same for reading.

The three sound channels produced by the PSG are mixed in a sound mixer circuit. The mixer also receives the inputs from other devices that are also part of the final sound signal sent to the TV set or the speakers, as shown in Figure 9.

If you remember the Figure 6, the bit 7 of PPI port C is used to produce the characteristic click sound when the user types in the keyboard. This sounds comes from a rudimentary squared waveform produced by this bit from the PPI. The system software, typically the BIOS, will write an alternating sequence of zeroes and ones to this bit of the PPI port C that produces the squared wave. The port bit is connected to the sound mixer to include it in the sound signal that goes to the TV set or the speakers.

Finally, there is another input that feeds the sound mixer. It is the sound signal coming from the system bus. As we will see soon, the MSX is designed to be expandable. And to take this idea to the top, the external expansion devices can produce their own sound signal that will be mixed with those produced by the PSG and the keyboard clicks. Thanks to this, the PSG can be complemented with another sound chips connected through the expansion slots.

That is all concerning sound generation. However, the AY-3-8910 was designed to assume other tasks in the system. It also incorporates two I/O ports that can be used to connect parallel peripherals in a similar way as we do with the PPI. These ports are named A and B. They can be programmed as input or output. And in MSX, they are used to connect the general-

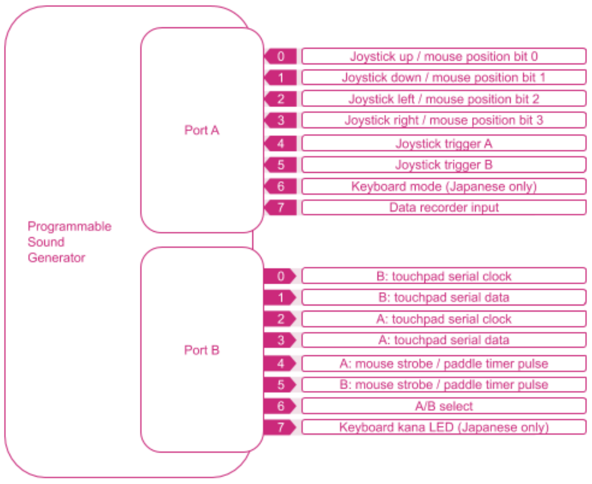

Programmable Sound Generator
Port A
0 Joystick up / mouse position bit 0
1 Joystick down / mouse position bit 1
2 Joystick left / mouse position bit 2
3 Joystick right / mouse position bit 3
4 Joystick trigger A
5 Joystick trigger B
6 Keyboard mode (Japanese only)
7 Data recorder input
Port B
0 B: touchpad serial clock
1 B: touchpad serial data
2 A: touchpad serial clock
3 A: touchpad serial data
4 A: mouse strobe / paddle timer pulse
5 B: mouse strobe / paddle timer pulse
6 A/B select
7 Keyboard kana LED (Japanese only)

Figure 10: PSG I/O ports

purpose ports, the data recorder input and a few other IO/functions as shown in Figure 10.

The general-purpose ports are informally known as the joystick ports. This is because its main usage is to connect joysticks to the MSX. But they can be also used for many other device types such as mouses, trackpads, paddles, etc.

The port A of the PSG is configured as input by the MSX BIOS upon system boot. The four less significant bits are used to read joystick movements or mouse position values. The bits 4 and 5 are used to detect whether the trigger A and B is pressed or not, respectively. The bit 6 is used in Japanese MSX models only. It tells what kind of key distribution the computer has, where 1 is JIS and 0 is ANSI. The bit 7 is used to receive the input signal from the data recorder.

As we have seen before, the data recorder produces a sound signal that is directly connected to the computer. As shown in Figure 9, this analog sound signal is converted into a digital equivalent that can be received in the bit 7 of PSG port A.

The port B of the PSG is configured as output by the MSX BIOS upon system boot. The four less significant bits are used for touchpad devices only to transmit the clock and data signals. For the rest of devices, they will be set to 1. The bit 4 and 5 are the mouse strobe signal or the paddle timer pulses for devices plugged to connector A and B, respectively. The bit 6 is the connector select bit. Did you wonder what device the six less significant bits from port A refer to? Yes, as we can have two joysticks, mouses, etc., it should be possible to read movements from two different devices. The fact is that these six bits from port A are multiplexed. It means the same wires are used to carry the movement information from connector A or B. The bit 6 of port B determines which device will put their information on the wires that connect to PSG port A. Finally, the bit 7 drives the LED of the Kana key in Japanese keyboards.

Expansion slots

We are almost done. There is only one more important part of an MSX system to discuss: the expansion slots.

As mentioned above, the MSX system was designed with the expansibility in mind. The intention of its authors was to allow the users to add new hardware and increase the capacity of their computers as easily as possible. To achieve this, they designed the computer to have expansion slots.

You are probably more than used to use expansion slots, also commonly known as *cartridge slots*. They are 50 pin edge

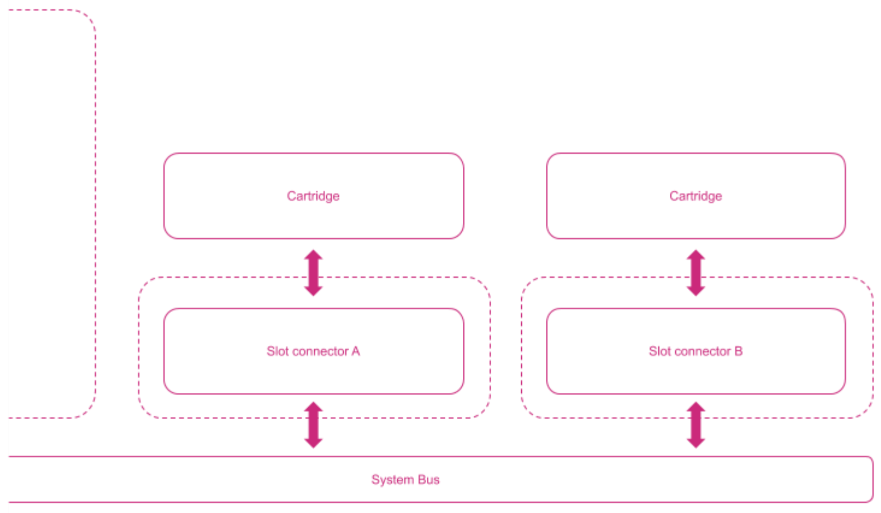


Figure 11: Expansion Slots

connectors in the motherboard of the MSX where the user can insert cartridges to expand the system. These cartridges are RAM expansions, games, storage devices, floppy drives, sound chips, etc. MSX systems have at least one cartridge slot, but it is not unusual to see models with two slots. In some other cases a cartridge slot is complemented with a 50-pin expansion IDC connector that has the same or similar pinout. The popular Spectravideo SVI-728 is an example of this.

The expansibility of the cartridge slot is possible because, in essence, the slot connector is an extension of the system bus, as shown in Figure 11. Most of the lines that comprise the system bus are included in the 50 pins connector. Thus, connecting a cartridge to the slot is equivalent to having such hardware soldered into the motherboard of the computer.

Summary

We have seen an overview of the most important parts that conform an MSX system. There are some others that will not be covered here, such as the serial communications port or the printer port.

As you can see, and as we revealed at the beginning of this chapter, the architecture of the MSX computer is extremely simple. Compared to other contemporary computers such as the IBM PC, the MSX do not have neither an interrupt nor a DMA controller. It has no mass storage device other than the data recorder and it has no real-time clock. Most of its hardware is based on off-the-shelf components, such as the TMS9918 VDP or the AY-3-8910 PSG. This contrasts with other hardware platforms such as the ZX Spectrum, the Amstrad CPC or even the IBM PC, which mount their own proprietary video display controllers. The MSX was designed to be cost-effective and standard. And this means a simple architecture easy to reproduce.

As we have seen from the beginning of this chapter, the most complex part of the MSX architecture is the memory system. Due to its relevance, we will dedicate a whole section to it.

MSX Memory Slots

Bank switching

The Z80 microprocessor uses 16-bit numbers to represent memory addresses. This means it can reference 65,536 memory locations. Assuming each memory location is 8-bit size, the whole memory space has 64KB.

In the time the Z80 CPU was designed, that memory was enough for almost any purpose. But as the computers acquired more power and the user demands increased, the needs for more and more memory also appeared. In 1984, when MSX architecture was set, most computer engineers were incorporating in their designs methods to break the 64KB barrier present in most CPUs. These methods are based on what is known as **memory bank switching**.

The concept is simple, although the implementation may become cumbersome. Instead of having a single RAM memory region, we will have multiple. Each region will be called *bank.* The system determines what memory bank is used by the CPU in each instant in function of some programmable configuration and the memory address that is being used.

For example, a Z80-based computer can have four banks of 64KB of RAM memory each. If we put them all together, they are 256KB of memory, as shown in Figure 12. That is indeed not bad for an 8-bit computer. The software can choose what bank is active using a 2-bit register that is accessible through I/O requests. In this example the two bits could be part of the port of a PPI or similar, as we have seen at the beginning of this chapter. The system BIOS can preconfigure this bank select register with the value 0b00 so the first 64KBs of memory are visible to the CPU. At some point, the program can write the value 0b01 to the bank select register to switch to the memory bank 1. This means the CPU will see after this the memory from 64KB to 128KB. With bank 0b10 and 0b11, the memory segments from 128KB to 196KB, and from 196KB to 256KB, respectively will be visible to the CPU.

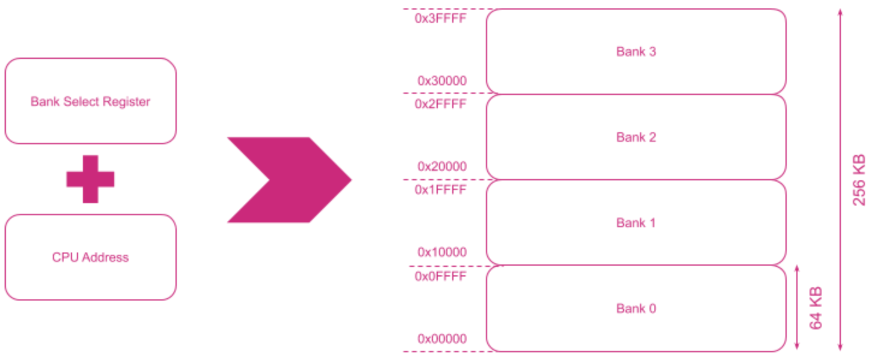


Figure 12: Example memory bank switching

If you think about this in terms of memory addresses, you will note we are extending the memory addresses from 16 bits to 18 bits. To distinguish both, we may call the 16-bit address generated by the CPU as *logical address* and the extended 18-bit address as *physical address*. The physical address results from concatenating the contents of the bank select register to the logical address. Thus, if bank 2 is selected and the CPU requests an access to the memory location at 0x800A, the physical address will be 0x2800A (0x2 concatenated to 0x800A).

Unfortunately, this bank switch solution is too naïve and will not work in a real system. This is because we are considering the whole memory space is switched when a new bank is selected. And this interrupts the execution of the program.

For example, consider a program located at logical address 0x8000 is running with the bank 0 active. This is the physical address 0x08000. At some point, the program decides it needs more memory, so it must switch to the bank 1. After the switch, the memory seen by the CPU at logical address 0x8000 is no longer at physical address 0x08000, but 0x18000. In other words, the program the CPU was executing suddenly disappears. The next instruction the CPU will execute after bank switching is in 0x18000 region of physical memory, which contains uninitialized memory.

As we already mentioned above, this is typically addressed by also considering the segment of the memory that is being addressed as an input to calculate the memory bank. We can extend our example such as instead of having one bank select register, we have four of them. The first register will indicate the bank of memory space located from logical address 0x0000 to 0x3FFF. The second one is the bank for logical addresses between 0x4000 to 0x7FFF. And the third and fourth bank registers will be used for the logical address space from 0x8000 to 0xBFFF and 0xC000 to 0xFFFF, respectively.

In the case of a program that is located at physical address 0x08000, the 3rd bank select register will have a value of 0. The rest of register can have any other values, pointing to the same bank or another. If the program decides to make use of more memory, it can switch the 1st, 2nd and 4th bank registers to point to new banks with available memory. But it must not change the 3rd register or the program will suddenly disappear as we have seen before.

MSX Slots

Although you may find this surprising (or not), what we had described above are the basics of what the MSX system does with its memory.

The MSX divides the 64 KB of the logical address space of the Z80 CPU by four. Each logical address space segment is called a *page*. The pages are shown in Table 3.

Page	Start logical address	End logical address
0	0x0000	0x3FFF
1	0x4000	0x7FFF
2	0x8000	0xBFFF
3	0xC000	0xFFFF

Table 3: MSX memory pages

The physical memory region that is mapped to each page is called a *slot*. There are four primary slots, and each of them can be extended by another four secondary slots to a total maximum of 16 slots. But we will cover this later.

The primary slot configured for each page is determined by the port A of the PPI, as we have seen before. Each pair of bits of that port provide a number from 0 to 3 that tells what slot is used by that page, as we seen in Figure 6. The slots can be reconfigured with I/O write requests to the PPI to switch the bits of the port A.

That is to configure the slots for each memory page. But what is exactly a slot? In the previous section, we have simplified the example by telling each bank is a 64KB region of RAM memory. But the real life is more complicated. In the real world, we have RAM and ROM memories. And in the case of the MSX, we want to have an extensible system. And that includes adding cartridges with RAM expansion or some hardware that has BIOS extensions written in ROM memories to manage the device. That is why in MSX we talk about slots, and not banks.

An MSX slot is anything that can respond to memory read and write operations. Typically, an MSX system has some built-in slots with some RAM or ROM memory. While some other slots are provided by… do you guess? Yes! The cartridge slots!

We had said the cartridge slots are expansions of the system bus. And they are indeed! But each cartridge slot has a designated memory slot. The 50-pin connector incorporates a select signal activated when it is targeted as the active slot when some page is addressed. When this happens, the cartridge connected to the slot must respond to the memory operations when they occur.

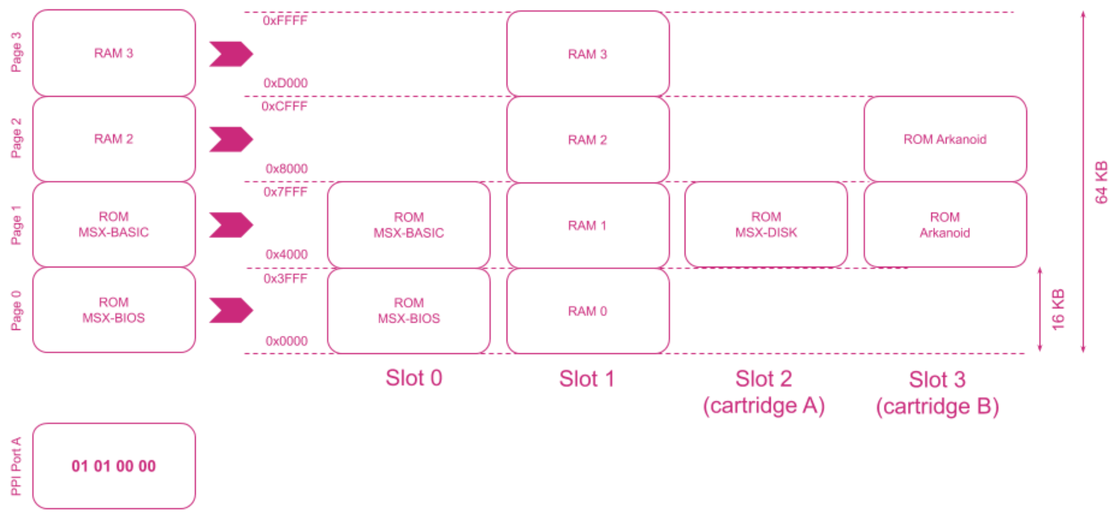


Figure 13: MSX slot configuration

An example of this slot organization is shown in Figure 13. In this case, the system has two built-in primary slots. The slot 0 contains a ROM memory from address 0x0000 to 0x7FFF. This ROM memory contains the MSX-BIOS and the MSX-BASIC in pages 0 and 1, respectively. Pages 2 and 3 are not wired. The slot 1 is a 64KB RAM memory occupying the for pages. The slots 2 and 3 are connected to the cartridge slots of the computer. In this example, we see a floppy drive connected to the cartridge slot A, which is the memory slot 2. This cartridge contains a ROM memory with an MSX-DISK BIOS extension to manage the floppy drive at memory page 1. In the cartridge slot B, which is the memory slot 3, we had plugged the popular game Arkanoid. This cartridge contains a ROM located at pages 1 and 2 with the game program and data.

In this example, the PPI port A is set to 0b01010000. This means the page 0 and 1 are mapped to the slot 0. That is why the MSX-BIOS and the MSX-BASIC are visible to the CPU. The page 2 and 3 are mapped to the slot 1. This is the slot full of RAM memory. Thus, the CPU will see RAM memory in the last two pages.

Now let us say we load a program from the data recorder to the RAM at page 2 and run it. The code of the program can determine it needs more RAM than the 32KB already mapped. In order to use more RAM, it can configure the RAM slot in the page 1 by writing 0b01010100 to the PPI port A. Thus, it gets rid of the MSX-BASIC code to increase the RAM to 48KB. If the program needs even more RAM, it can configure the slot 1 for page 0 as well, such that the whole address space is full of RAM.

Please note the slot reconfiguration is not irreversible. Even in the case the program needs the whole 64KB of RAM memory, it does not mean it cannot use the MSX-BIOS. For example, it might configure page 0 to use the RAM slot only when it must read or write to it. And once finished, restore the slot configuration to have again the MSX-BIOS code in the page 0. The same principle applies to any other memory page. There is only one restriction in this model: you cannot change the slot of the page where the current program is running. Because as we have seen before, this will interrupt the program.

One important note: the slot distribution seen here is just an example. The MSX standard does not indicate any specific memory layout. Some vendors decide to use the slot 1 for RAM. Some others use the slot 3. And they put the cartridges in slots 1 and 2. Some others only have 1 cartridge slot, so one of the memory slots is not used. Each vendor decides their own slot distribution. The software or the users must not make assumptions on where each memory device is located.

Secondary memory slots

Finally, it is time to discuss about the expanded slots. We already mentioned that each slot can be expanded such that it can contain four secondary slots. It means this slot will respond to read and write operations by forwarding the request to another sub-slot. The MSX standard specifies a way to configure the sub-slot that will attend the memory accesses when the primary slot is selected in the PPI. This is with a register located in the address 0xFFFF of the primary slot. This register operates in the same way as the port A of the PPI, with each pair of 2 bits selecting the sub-slot that will be used for each memory page.

Let us see this with an example. Some MSX system may have the slot 1 expanded with two sub-slots. The sub-slot 0 contains 4 pages of RAM. The sub-slot 1 contains one page of ROM with some MSX-BIOS expansion. This primary slot 1 has a hardware register that responds to read and write operations at the address 0xFFFF.

If we want to map the page 2 and 3 to RAM, we must:

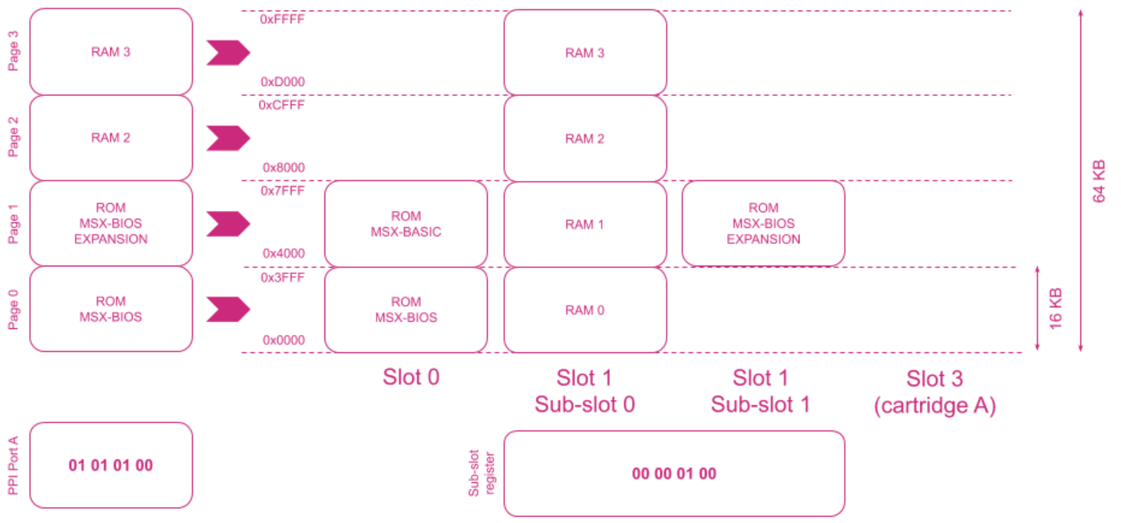


Figure 14: Secondary slot configuration

- Configure the pages 2 and 3 to use the primary slot 1. This means writing the value 0b0101xxxx to the PPI port A.
- Configure the register of primary slot 1 to use sub-slot 0 for pages 2 and 3. This means writing the value 0b0000xxxx to the register of slot 1 at address 0xFFFF.

Analogously, to have the MSX-BIOS Expansion accessible at page 1, we must:

- Configure the page 1 to use the primary slot 1. This means writing the value 0bxxxx01xx to the PPI port A.
- Configure the register of primary slot 1 to use sub-slot 1 for page 1. This means writing the value 0bxxxx01xx to the register of slot 1 at address 0xFFFF.

As you might figure out, accessing the address 0xFFFF in certain primary slots is tricky. A primary slot will respond to that address only when page 3 is configured to use that slot because address 0xFFFF belongs to page 3. Thus, in certain configurations it is necessary to configure the page 3 to use that slot first, and then write to the address 0xFFFF to set the sub-slot configuration. After this, we must restore the previous configuration of the primary slots to leave the things in a consistent state.

But the problems do not end there. As we mentioned above, the MSX standard does not impose any specific slot layout. The users and their software must not make any assumption about where RAM or ROM is located. This implies the software, including the MSX BIOS, must be able to discover where the memory is. This can be done with a subprogram that scans the slots looking for RAM memory. But how can it determine if a primary slot is expanded or not? Consider an expanded slot has a register at address 0xFFFF. But a primary slot may have a RAM memory in this very address. In both cases, this address can be written and read. How could the software say if this is a cell in a RAM memory chip or it is a hardware register?

The MSX standard has an answer to this. It states that the hardware register of an expanded slot must respond by inverting the contents when the register is read by the CPU. Thus, if the software writes 0b01010101 to the address 0xFFFF of a primary slot and immediately after that it reads the contents, there are three possible results:

- 0b01010101. The same value that was written. It means this slot contains RAM and it is not expanded.
- 0b10101010. The value that was written with its bits inverted. It means this slot is expanded.

- Any other value. The slot does not have anything responding to write operations at 0xFFFF. It means indeed the slot is not expanded. It might be a primary slot with ROM at page 3, the slot is not connected, or it is a cartridge slot that has nothing plugged.

As you can see, the memory slots system is rather complex but powerful. Each memory page can be mapped to a maximum of 16 different slots. This is a total address space of 1 megabyte of memory. Not bad considering that *640 KB ought to be enough for everybody*.

Architecture of Artemisa MSX

The Artemisa Homebrew DIY model 101 implements most of the parts of the MSX standard have we seen above.

In a few words, Artemisa is an MSX computer with 64KB of RAM memory and two cartridge slots. It has two general purpose (joystick) ports, a data recorder connector, an external keyboard and an interchangeable graphics card. All this encapsulated in a transparent acrylic case of 300x150x35 millimeters.

There are just a few things you typically find in an MSX computer that are not present in Artemisa:

- Serial communications port. This can be used to connect modems or other communication devices.
- Printer port. Used for printing.
- +12- and -12-volts power lines. The built-in hardware of Artemisa use 5 volts power only. However, the +12- and -12-volt lines are included in the pinout of the cartridge connectors. Artemisa wires these pins to ground instead.

Keyboard

Most of the MSX computers, as many other microcomputers built during the 80s, had an integrated keyboard. These computers look like a big keyboard that contains all the elements inside. In contrast, a few MSX vendors followed the same approach as IBM did with their PC: separate the keyboard from the computer connected through a keyboard port. This makes the system less integrated. But also brings some benefits. First, the keyboard can be easily replaced in case of failure. Second, it is easier to manufact computers at scale with different keyboard distributions and layouts, as they only differ in the keyboard.

Artemisa also uses an external keyboard to get advantage from these benefits: reduce the design cost and let the user to choose their keyboard option. It has a custom 25-pin connector that can be used to connect an external keyboard. This can be a custom keyboard built for the Artemisa, or an adapter that allows to connect a PS2 or a USB keyboard.

The Artemisa Homebrew DIY model 101 is shipped by default with a PS2 adapter. You can use it to connect a regular, unexpensive PC keyboard. A microcontroller will map the scancodes from the PS2 protocol to simulate a keyboard matrix the MSX will see connected to the port.

Graphics card

The same reasons to have a separated keyboard also applies to have a separated graphics device. Each region has its own TV encoding standards. Some countries use NTSC. Some others use PAL. Some users prefer an unexpensive device that uses composite video output. Some others prefer a RGB output via an SCART connector.

In order to simplify the design and allow the users to customize the video output according to their needs, Artemisa Homebrew DIY separates the graphics card from the rest of the computer motherboard. Both are internally connected using a 50-pin ribbon cable.

CMOS Technology

We will not go too deep in this subject, but it might be worth to discuss a little bit about the semiconductor technologies behind Artemisa.

As you probably know, there was a key invention that boosted the development of computer systems: the transistor. This semiconductor device is the most basic building block used to make basic digital circuits such as logic gates. There are multiple technologies used to implement transistors depending on the semiconductor materials used in the process. Each technology has different properties, constraints and tradeoffs.

During the 70s and 80s, the most used transistor type in consumer electronics was the bipolar junction transistor (BJT). The BJT transistors were the foundations of the Transistor-Transistor Logic technology, or TTL. One of its subfamilies, the TTL-LS, was the de-facto standard used in early microcomputers. The TTL-LS devices switch relatively quick from one logic state to another. This means they have low propagation delays (a voltage change in the input of a gate is quickly reflected in the output voltage), and they can operate at high frequencies (many voltage changes per second can occur). However, this comes in exchange of a high power-consumption. Also, the current they can drive is quite limited. This affects the number of input devices than can be connected to the same output of a TTL device, what is known as the fan-out problem.

In the middle of the 80s, another technology displaced TTL-LS in the consumer electronics market. It was the Complementary Metal-Oxide-Semiconductor (CMOS) technology. It was made up by Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFET). CMOS had two interesting characteristics compared to TTL devices. First, they had a low static power consumption. This means CMOS devices consume much less power than TTL devices when the voltage of their inputs is not changing. Fortunately, for nearly all logic gates in a computer, this is most of the time. This had an interesting side effect: the integration scale is significantly increased. Less power consumption means more transistor density in a chip core (without making the chip burn), what leads to miniaturization. In fact, modern inventions such as the microprocessor were possible thanks to the MOSFET transistors and the derived logic families like NMOS and CMOS. Secondly, CMOS devices can drive more load than TTL equivalents. This practically eliminates the fan-out problem from the equation. These devices also have a better electrical noise reduction. However, the toll paid by CMOS devices is their speed. They respond relatively slow to the changes in their logic state. What cause relatively large propagation delays compared to TTL devices and force them operate at low frequencies. Fortunately, a subfamily known as High-speed CMOS (HCMOS) reduced the propagation delays to acceptable levels.

Roughly speaking, you can assume a TTL logic gate consumes 1000 times the power of its CMOS equivalent. In the other hand, a CMOS device has propagation delays 2x to 5x bigger than their TTL counterparts. The good news is that an MSX computer is a relatively slow machine. The CPU operates at 3.58Mhz and each interaction with the system bus typically involves several clock cycles. The response times are in the range of hundreds of nanoseconds. That is by far in the range of the capabilities of the HCMOS devices.

Taking these considerations, Artemisa uses CMOS devices where possible instead of the TTL equivalents used in the MSX computers designed during the 80s. The glue logic is implemented by integrated circuits of the 74HC family. And the system is designed to operate with Z84C00 and 82C55 chips, the CMOS versions of Z80 microprocessor and the PPI, respectively. There are just a few TTL chips in the design that do not have CMOS equivalents, such as the 74LS07.

Integration level.

As discussed in the beginning of this book, Artemisa Homebrew DIY wants to make you learn how an MSX works while you assemble and solder its parts. The easier the assembling process the better. That is why this version of Artemisa sacrifices the miniaturization in favor or an easier soldering experience.

As consequence, all the chips used in the design have Dual Inline Package (DIP) encapsulation. These are the classic chips used in early microcomputers, formed by two lines of broad pins with a separation of 2.55mm. The same principle is applied to the rest of components. Resistors and capacitors have Through Hole (TH) encapsulation to facilitate the manipulation.

Memory technology

The RAM memory was one of the main limitations of early microcomputers. And not for performance reasons, but costs. Memories were really expensive these days. And engineers had to find ways to make them as affordable as possible.

All RAM memories are made by several x-y matrices of bit cells. For read and write operations, the chip expects to receive the row and column addresses in order to activate the cell where that bit is located and perform the operation. Typically, the bits of the physical memory address are split in two halves such that *n* less significant bits are the row address and *m* most significant bits are the column address in the memory matrices. By having eight matrices, one per each bit in a byte, we can implement the system memory of an 8-bit computer.

There are two principal technologies to build RAM devices: DRAM and SRAM. DRAM technology is based on a combination of one capacitor and one transistor to implement each bit cell. The capacitor is used to determine whether the cell contains a logic 1 (charged) or a logic 0 (discharged). The transistor is used as a switch to control the access to the capacitor. Each bit cell is put in a x-y matrix that is addressed by row and column. As it uses only two elements that can be built into a microchip, DRAM is unexpensive compared to other memory technologies.

Unfortunately, DRAM have a big drawback: capacitors tend to discharge by their own. With the time the charge leaks away, so a charged capacitor representing a logic 1 passes to represent a logic 0. Considering the small size of these capacitors, this discharge time is in the order of milliseconds. The RAM chips must *refresh* the contents of its cells periodically to prevent information lost. In most of the cases, they cannot do that by themselves. And require external circuits to perform the refresh actions periodically. In addition, DRAM chips typically multiplex the pins used for row and column addresses. They depend on a multiplexor circuit to decode the bits from the address bus that belong to row and column. Both the refresh and the row/column multiplexor circuits are not trivial, and they complicate the design of the system.

The alternative to DRAM is the Static RAM (SRAM). This kind of memory uses pure semiconductor devices to implement bistables that can keep 1 bit of information. Typically, six MOSFET transistors are needed per bit cell. What makes them much more expensive than the DRAM equivalents. However, as they do not use capacitors there is no need for refresh circuits.

In the present, the DRAM is still used in modern computers with gigabytes of memory due to its density. SDRAM and DDR technologies are DRAM. However, vendors do not produce DRAM chips for small memory amounts any longer. The SRAM technology has evolved enough to become an unexpensive solution in the range of kilobytes and megabytes. All parts traditionally used in 80s computers are obsolete. And replaces are only available in the secondhand market.

Because of this, and the fact that it simplifies the design of the system, Artemisa uses SRAM memory instead of DRAM. Thanks to this, it does not have to implement a memory refresh circuit or a row/column address multiplexing.

Memory layout

As discussed above, the MSX standard does not define how much memory an MSX computer must have or how it is distributed along the different memory slots.

Artemisa have 64KB of RAM memory located in memory slot 1. These are 4 pages of RAM that occupy the whole slot.

Concerning ROM memory, Artemisa has 32KB of ROM memory where MSX System (MSX BIOS plus MSX Basic) can be found as almost any other MSX system. However, the motherboard provides up to 512KB of total ROM memory. As only 32KB of it are mapped to the MSX in the pages 0 and 1 of memory slot 0, this means the ROM memory has up to 16 different regions of 32KB. Each region contains a different image of the MSX System. A set of 4 jumpers in the motherboard can be used to select the image that will be mapped in the MSX memory space.

The reason to have multiple interchangeable MSX System images is to choose different versions of it, each one supporting different regional configuration. The MSX System image for USA is not the same as the MSX System image for Spain. They expect different keyboard distributions, use a different character set and, in some cases, they even have a different bootscreen message.

Jumper config	ROM contents
0b0000	MSX System v1.0 International
0b0001	MSX System v1.0 USA
0b0010	MSX System v1.0 Japan
0b0011	MSX System v1.0 UK
0b0100	MSX System v1.0 France
0b0101	MSX System v1.0 Germany
0b0110	MSX System v1.0 Italy

0b0111	MSX System v1.0 Spain
0b1000	MSX System v1.0 Arabic
0b1001	MSX System v1.0 Korea
0b1010	MSX System v1.0 Russia
0b1011	Artemisa Diagnosis Tool v0.1

Table 4: MSX System images in ROM memory

The Table 4 summarizes the contents of the Artemisa ROM distribution and the jumper configuration to activate each. As you can see, most of them are MSX System versions. The last one, activated with the jumper configuration 0b1011, is the Artemisa Diagnosis Tool. This is a utility that replaces MSX Basic in order to perform some basic system checks that could be of help diagnosing system problems.

Summary

In this chapter, we have presented the architecture of a MSX computer. We have seen also some design choices made by Artemisa to implement this architecture.

Now it is time to have some action! Prepare your soldering iron because this amazing experience has only just begun.

Preparation

Contents of the package

The package Artemisa Computer Model 101 contains the following elements:

Artemisa Motherboard 101 printed circuit board

Integrated circuits ESD bag, including:

1x Z84C00 CMOS CPU
1x 82C55 Parallel Peripheral Interface
1x AY-3-8910 Programmable Sound Generator
2x HM62256/AS62256 32KB SRAM
1x SST39SF040 512KB flash ROM memory
1x 74HC04 hex inverter
1x 74HCU04 unbuffered hex inverter
1x 74LS07 open-collector hex buffer
1x 74HC08 quad 2-input AND gate
1x 74HCT08 quad 2-input AND gate with TTL-compatible inputs
3x 74HC32 quad 2-input OR gate
2x 74HC74 dual D-type flip flop
2x 74HC157 quad 2-input multiplexer
3x 74HC138 3-to-8 line decoder/demultiplexer
1x 74HC139 dual 2-to-4 line decoder/demultiplexer
1x 74HC153 dual 4-input multiplexer
3x 74HCT244 octal buffer/line driver
1x 74HCT245 octal bus transceiver
1x LM311 differential voltage comparator

Sockets and connectors bag, including:

2x IC Socket 28p .600 mil
1x IC Socket 32p .600 mil.
3x IC Socket 40p .600 mil.
1x IDC 50p. board connector
1x DB25 male board connector
1x DIN45326 8p. board connector
2x 50p. edge card connector
1x 55x21mm barrel DC connector
2x DE9 make connector

Resistor bag, including:

1x 100 Ω resistor
1x 220 Ω resistor
1x 470 Ω resistor
3x 1K Ω resistor
1x 2K2 Ω resistor
3x 2K7 Ω resistor
6x 4K7 Ω resistor
4x 10K Ω resistor
1x 20K Ω resistor

1x 100K Ω resistor
1x 220K Ω resistor
1x 1M Ω resistor
2x 10K Ω 5p. resistor network
1x 10K Ω 7p. resistor network
2x 10K Ω 9p. resistor network

Capacitor bag, including:

2x 33pF ceramic capacitor
28x 100nF ceramic capacitor
2x 22nF ceramic capacitor
2x 1uF electrolytic capacitor
4x 10uF electrolytic capacitor
2x 100uF electrolytic capacitor

Semiconductor bag, including:

2x 1N4448 small signal fast switching diode
1x 2N2222 NPN bipolar transistor
1x 2N2907 PNP bipolar transistor
1x 3mm LED diode

Miscellanea components bag, including:

1x relay 5VDC
4x pin header 3p. 100mil
4x pin header jumper
1x 6x6mm angled push button
1x 3.58 MHZ crystal

If some part is missing or damaged, please contact support@artemisamsx.com.

Required tools

You will need the following tools to assembly your GFX9918 unit:

Electric soldering iron

One suitable for soldering electronic components.

Spool of solder cable

If you have a limited experience on soldering, lead-based alloy is recommended. It melts at lower temperature, facilitating the job.

Soldering flux

A chemical cleaning agent used in the soldering process. Improves the thermal conductivity of the metals, ensuring lasting joints.

Wire cutters

You will need to remove the remains of legs for resistors and capacitors.

The following tools may be useful during the assembly process but are not required. If you have access to them, it would be a good idea to prepare them. If not, do not worry. You can assemble your unit without them.

Multimeter

It might be useful to check circuit voltage and continuity.

Oscilloscope

Useful if you want to debug some analog signals.

Logic analyzer

Useful if you want to debug some digital signals.

Motherboard Assembly

Anatomy of Artemisa 101 Motherboard

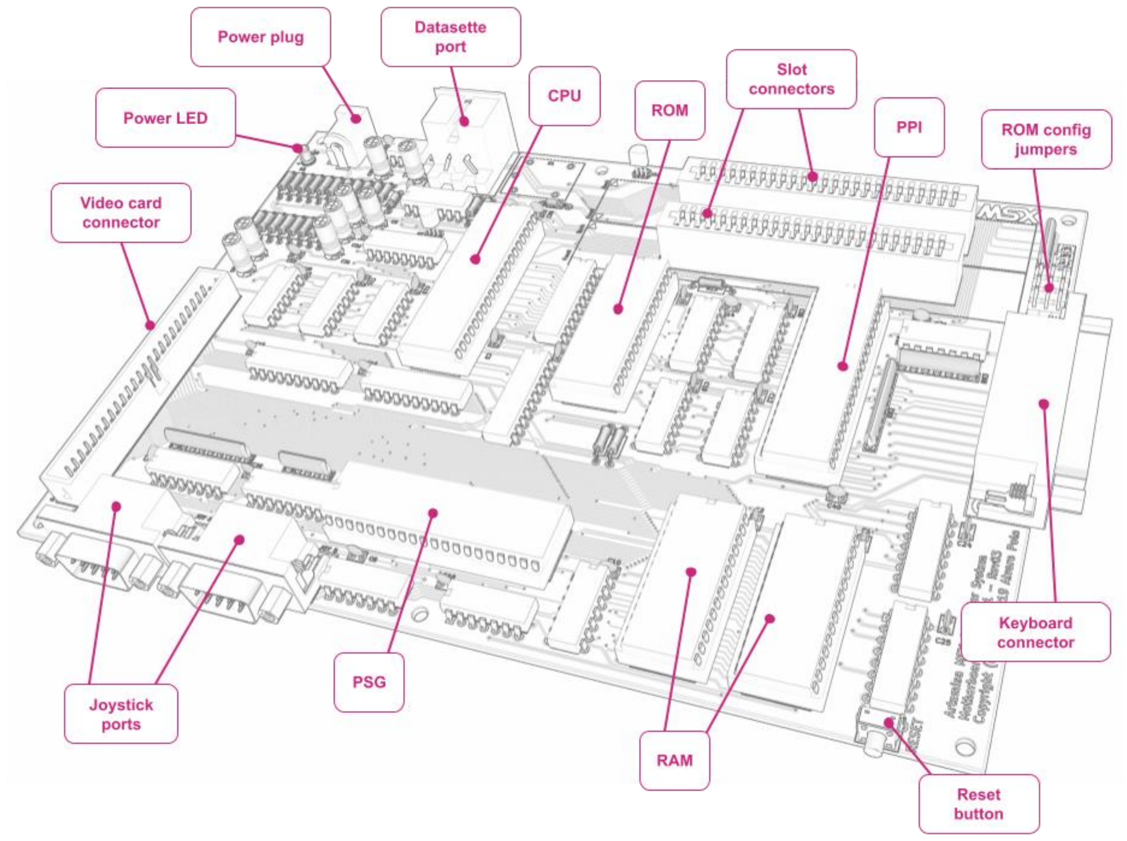
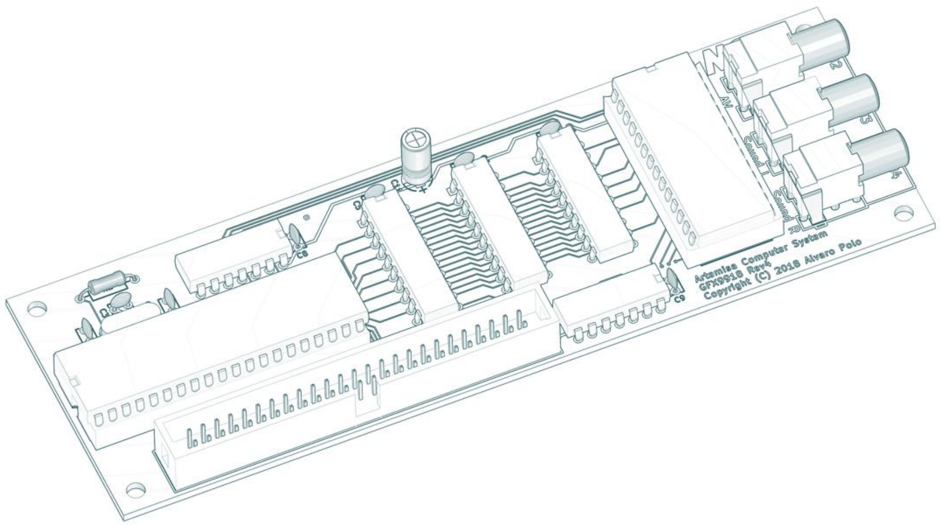


Figure 15: general anatomy of Artemisa 101 motherboard

Appendix A: GFX9918



Architecture of GFX9918

In the case of GFX9918 video card, these basic elements, shown in Figure 16, are:

- A TMS9918A VDP chip.
- 16KB of static memory used as VRAM.
- A custom input decoding logic. A few logic gates to interpret the system bus signals according to the interface of the TMS9918.
- A custom memory decoding logic that will demultiplex the signals coming from the VDP and will adapt to the timings of the VDP.
- No specific video output processing. The TMS9918 generates a composite video signal suitable for any TV set. Both this video output, and the sound signal coming from the motherboard, are available through a triple RCA connector.

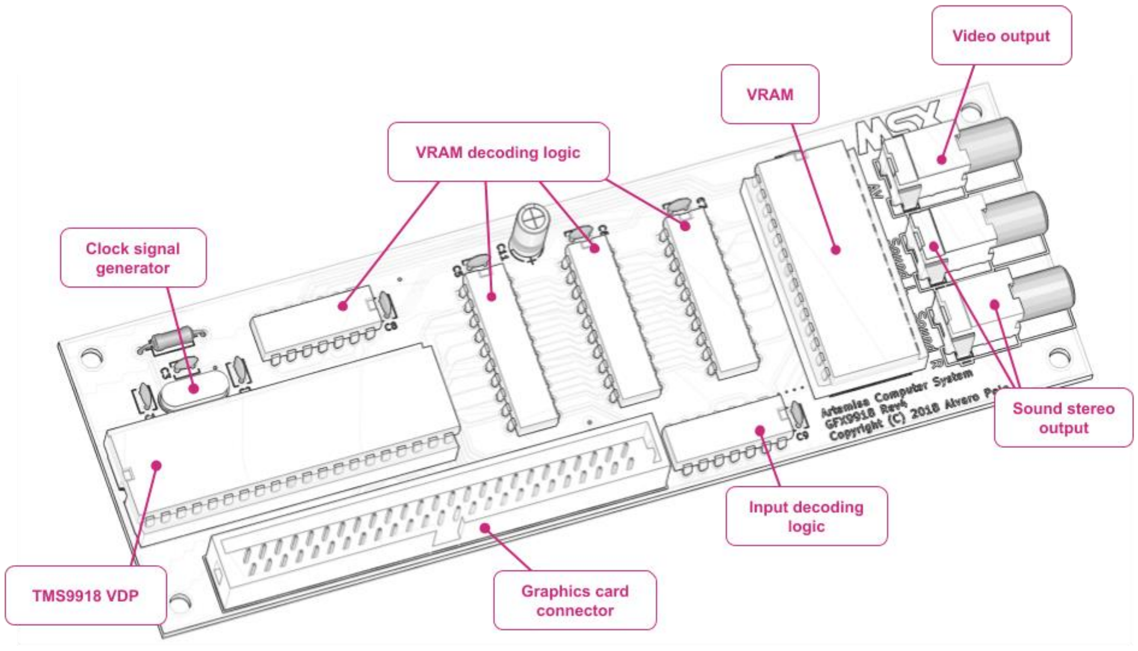


Figure 16: GFX9918 Basic Elements

Operation of GFX9918

The TMS9918 video display processor contains so internal register that can be programmed by the user:

- There are eight internal mode registers that can be used to configure things such as the screen mode, the VRAM base addresses for the different memory areas, the color palette, etc.
- There is one internal address register that will determine the VRAM address where read and write operations will be located.

As mentioned above, the TMS9918 chip requires a dedicated video memory separated from the main memory of the computer. This means the CPU will communicate to the VDP using exclusively the IO read and write operations.

The MSX1 standard establishes that the VDP is attached to the IO ports from 0x98 and 0x99, as shown in Table 5:

Port	Description
0x98	The data port. This is where the CPU will read or write when it wants to transfer data from or to the VDP.
0x99	The command port. On read, this will return the contents of the status register. On write, this will be interpreted as a command sent by the CPU to the VDP.

Table 5: VDP IO Ports

This is a summary of the typical operations the CPU requests to the VDP:

- Obtain the status of the VDP. This is done by reading the command port at IO address 0x99. This is useful to detect sprite collisions and frame rendering completed.
- Write into a mode register. This is done by writing two subsequent bytes to the command port at IO address 0x99. The first byte is the content to be written to the mode register. The second byte must match the binary pattern 0b10000xxx, where xxx is the 3-bit address of one of the eight available mode registers. The VDP knows this is a write request to a mode register because of the 0b10000 prefix.
- Write to VRAM address. This is done by first writing two subsequent bytes to the command port at IO address 0x99. These two bytes has the purpose of setting the contents of the internal address register mentioned above. The first byte contains the less significant bits of the address. The second byte must match the pattern 0b01xxxxxx, where xxxxxx are the six more significant bits of the address. The VDP knows this is a write request to VRAM because of the 0b01 prefix. After that, it expects the data to be written into the data port at IO address 0x98. The internal address register is autoincremented. After each write to the data port, it will increment its value pointing to the next byte in VRAM. This can be used by the CPU in order to transfer a large block of bytes by just setting the VRAM address once, and then writing the bytes in a row.
- Read to VRAM address. It works exactly as the write to VRAM but using the prefix 0b00 instead of 0b01. And reading from the data port instead of writing.

Preparation

Contents of the Package

The Artemisa GFX9918 package should contain the following elements:

Artemisa GFX9918 printed circuit board

Integrated circuits ESD bag, including:

1x TMS9918: Video Display Processor (VDP
1x 62256: 32,768-word × 8-bit High Speed CMOS Static RAM
1x 74HCT04: hex inverter
3x 74HCT574: 8-bit D-type flip-flop with 3-state outputs
1x 74HC32: quad 2-input OR gate

Sockets and connectors bag, including:

1x IC Socket 28p .600 mil
1x IC Socket 40p .600 mil.
3x RCA connectors (yellow, white and red)
1x IDC 50p PCB connector

Passive components bag, including:

2x Ceramic capacitor 56pF
7x Ceramic capacitor 100nF
1x Electrolytic capacitor 10uF
1x Resistor 470 ohms
1x Crystal 10.7386Mhz HC-49S CL=32pF

Triple RCA cable for composite video + stereo sound

If some part is missing or damaged, please contact support@artemisamsx.com.

Required tools

You will need the following tools to assembly your GFX9918 unit:

Electric soldering iron

One suitable for soldering electronic components.

Spool of solder cable

If you have a limited experience on soldering, lead-based alloy is recommended. It melts at lower temperature, facilitating the job.

Wire cutters

You will need to remove the remains of legs for resistors and capacitors.

The following tools may be useful during the assembly process but are not required. If you have access to them, it would be a good idea to prepare them. If not, do not worry. You can assemble your unit without them.

Multimeter

It might be useful to check circuit voltage and continuity.

Assembly

High Level Design

The schematic diagram in Figure 17 shows the main elements of the GFX9918:

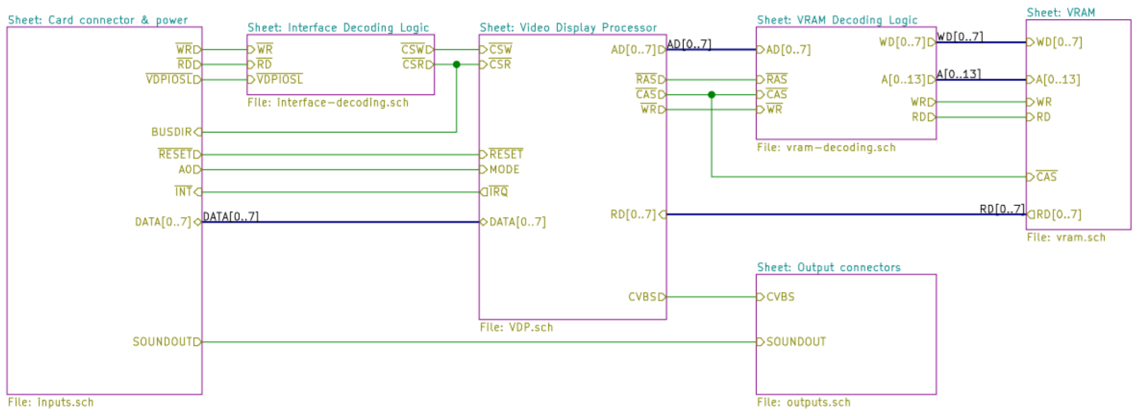


Figure 17: Main Schematic Diagram

- The Card Connector and Power. This is where card connector is placed, providing the control signals and the data bus to make the motherboard interface with the VDP.
- The Interface Decoding Logic. It receives a few control signals coming from the input connector and will generate the control signals adapted to the VDP interface.
- The Video Display Processor. This is the TMS9918 chip and the auxiliary elements that make it work. It receives the inputs from the Interface Decoding Logic and the Input Connector. And generates control signals and buses to interface with the VRAM memory through the VRAM Decoding Logic.
- The VRAM Decoding Logic. As its name suggests, it decodes the buses and control signals from the VDP into something the VRAM memory can understand.
- The Video RAM. The memory chip that will receive the inputs from the VRAM Decoding Logic and will respond to the corresponding memory accesses coming from the VDP.
- The Output Connectors, where the video output signal generated from the VDP and the audio signal coming from the Input Connector are mixed in a triple RCA connector.

We will discuss each block separately along this section of the manual. This brings an opportunity to understand how each part of the circuit works at the same time we solder the components to the board.

Card connector and power

We will start the assembly with the card connector and the power management elements.

As you can see in the schematic shown in Figure 18, this is quite simple. The 50P IDC connector is identified by J1. As you can see, we will use only the following control signals from the system bus:

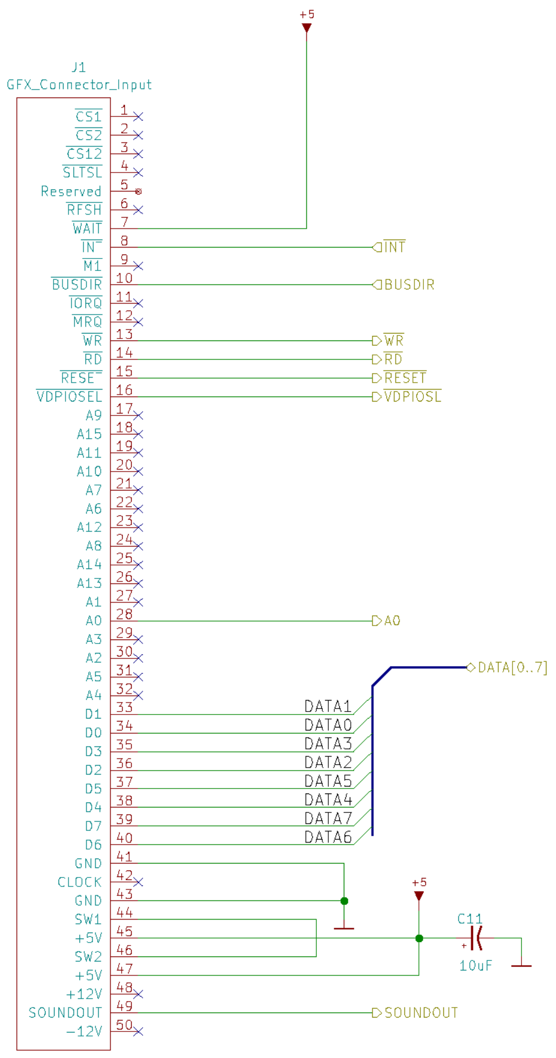


Figure 18: Card connector and power schematic diagram

- The /INT output signal. This will be activated by the VDP whenever it completes a whole screen frame.

CPU interruptions

Interruptions are a common way to communicate events from the peripherals to the CPU in any computer system. In this particular case, the VDP activates the interrupt signal when a whole screen frame is completely rendered. This is 60 times per second in GFX9918 due to the NTSC video encoding standard.

This interruption will cause the CPU to stop executing the current program and invoke an interruption routine, typically provided by the system BIOS. In the MSX system, this interruption is used as a system-wide timer that is triggered at known intervals (every 16.6 milliseconds). The default BIOS routine uses it to perform regular tasks, such as read the keyboard state. The user software can use it as well. For example to move the scene objects in a game.

- The BUSDIR signal. It will be activated by the graphics card whenever a data is sent by the VDP to the CPU.
- The /WR and /RD signals, indicating a write or read operation, respectively.

- The /RESET signal, which is activated when the system is powered on or restarted. This will be used by the VDP to reset to its initial state.
- The custom /VDPIOSL signal, which indicates an IO operation in any of the IO ports designated for the VDP.
- The A0 signal. This is the least significant bit of the address bus. Hence it distinguishes between IO ports 0x98 and 0x99, which are the same binary numbers except their last digit.
- The data bus, which is used to transfer data between the CPU and the VDP.
- The +5V and GND power connections.

As you can see in the schematic, there is also an electrolytic capacitor connected between 5V and GND. This is used to decouple the power circuit from AC fluctuations at low frequencies.

Decoupling capacitors

Electronic components do not consume the same current all the time. They experiment fluctuations. When a current spike occurs, the voltage provided by the power supply also changes. This could cause a voltage drop that can affect not only the component but also some other elements of the same board.

The way to prevent this is to use a decoupling capacitor. Think about it as a local battery connected to some part of the circuit. If the current drawn from the wire changes, the voltage drop can be compensated by the load stored in the capacitor. The volts lost by the power supply are provided by the capacitor. Maintaining the same voltage level until the power supply adapts its level to the new demand, or the extra current demand ends.

Typically, every integrated circuit has its own small decoupling capacitor to protect itself from the voltage fluctuations. There is also a single bulk capacitor aimed to decouple the whole power circuit of the board from the power line coming from the card connector.

Step 1

We will start by soldering the 50P IDC connector J1 in the board, as shown in Figure 19.

Please take care of the orientation of the connector. The notch must look to the bottom edge of the card, where the ribbon cable comes.

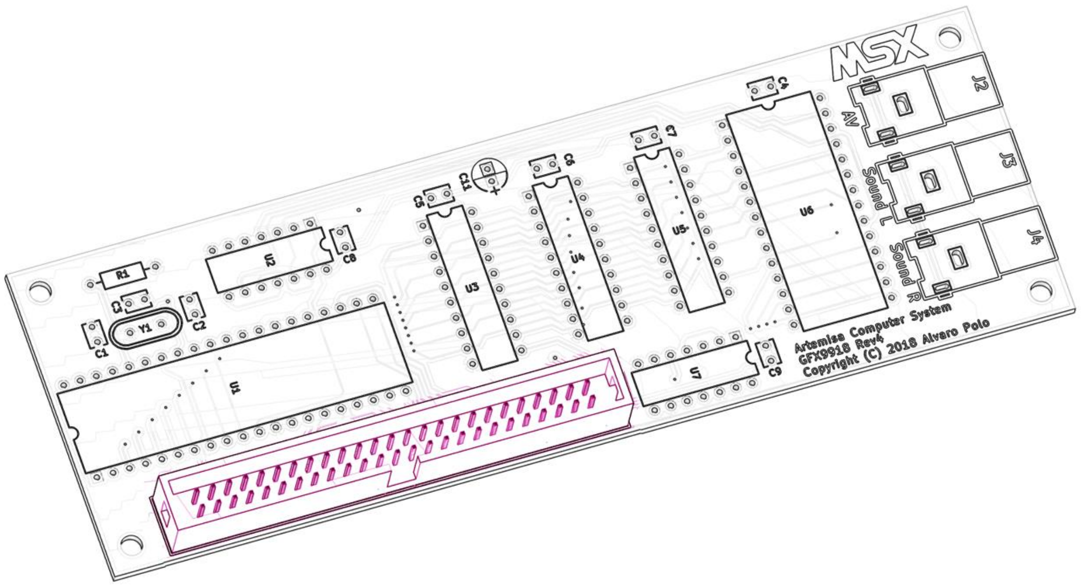


Figure 19: Placement of card connector

Step 2

Now we can solder the electrolytic capacitor C11, as shown in Figure 20.

Take care of the polarity of the electrolytic capacitors. If soldered incorrectly, the part might explode. The longest pad must be soldered in the hole marked with a plus sign.

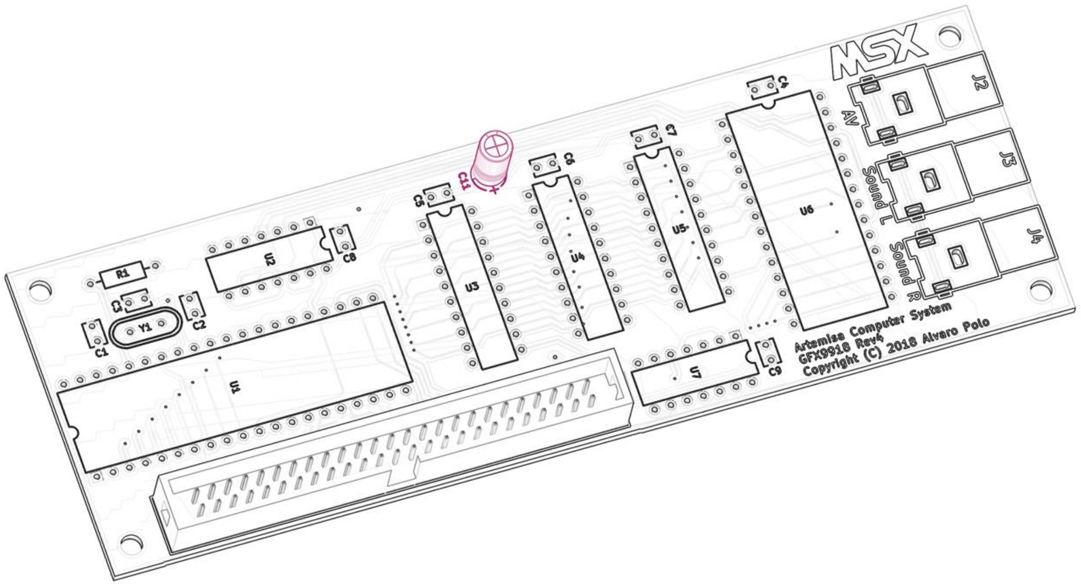


Figure 20: Placement of capacitor C11

Interface decoding logic

The interface decoding logic adapts the signals coming from the input connector into something adapted to the interface of the TMS9918.

This sounds complex, but it is as simple as a pair of AND logic gates as shown in Figure 7.

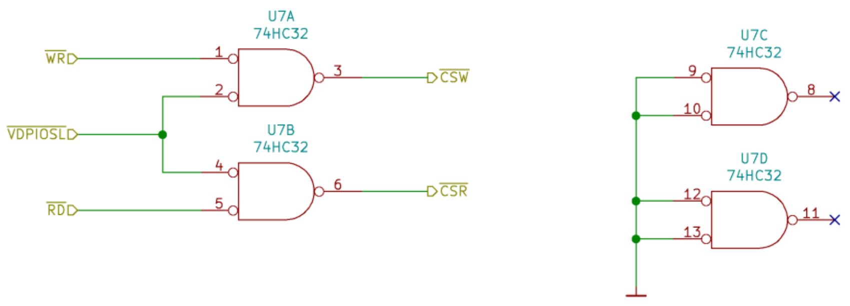


Figure 21: Interface decoding schematic diagram

Unused gate inputs

As you can see in Figure 21, there are two AND gates on the right side that are unused. However, their inputs are connected to GND instead of left disconnected. This is to prevent floating values that could damage the circuit due to electrostatic currents. Also, a floating value might cause fluctuations in the logic inputs, that in turn increases the power consumption in case of CMOS components. Any unused logic gate should be connected to either GND of VCC for a fixed input value.

The following inputs and outputs are handled in this decoding phase:

- The /WR input signal is activated when a write operation is requested by the CPU.
- The /RD input signal is activated when a read operation is requested by the CPU.
- The /VDPIOSL input signal is activated when an IO request to the VDP ports is requested by the CPU.
- The /CSW output signal indicates the VDP is selected for a write operation. This is the logic expression: /CSW = /WR *and* /VDPIOSL.
- The /CSR output signal indicates the VDP is selected for a read operation. This is the logic expression: /CSR = /RD *and* /VDPIOSL.

Please note all these control signals are active-low.

Active low signals

There are two different conventions used to consider a digital signal is active or inactive. One is to assume a high level means active and low level means inactive. This is called active-high. It is the most natural convention due to the assumption that low (0) means not while high (1) means yes. However, the reverse logic, called active-low, also exists. It considers a low level means active and a high level means inactive. All signals whose name is prepended with a slash (e.g., /CSR) are active-low.

In despite of active-high to be more natural, active-low is widely used in digital electronics. There are two main reasons for that. One of them is that old TTL family gates consume significantly more power when they receive a low level input. Considering the signals are most of the time inactive, active-low convention saves a lot of power.

The other reason is that combining active-low logic with open-collector outputs and pull-up resistor gives an OR function without using any logic gate. This combination of elements is not present in this circuit.

Step 3

Solder the integrated circuit U7 (74HC32) and its decoupling capacitor C9 (100nF) as indicated in Figure 22.

Take care with the orientation of the integrated circuit. The IC has a a notch at the top that must match the drawings in the board.

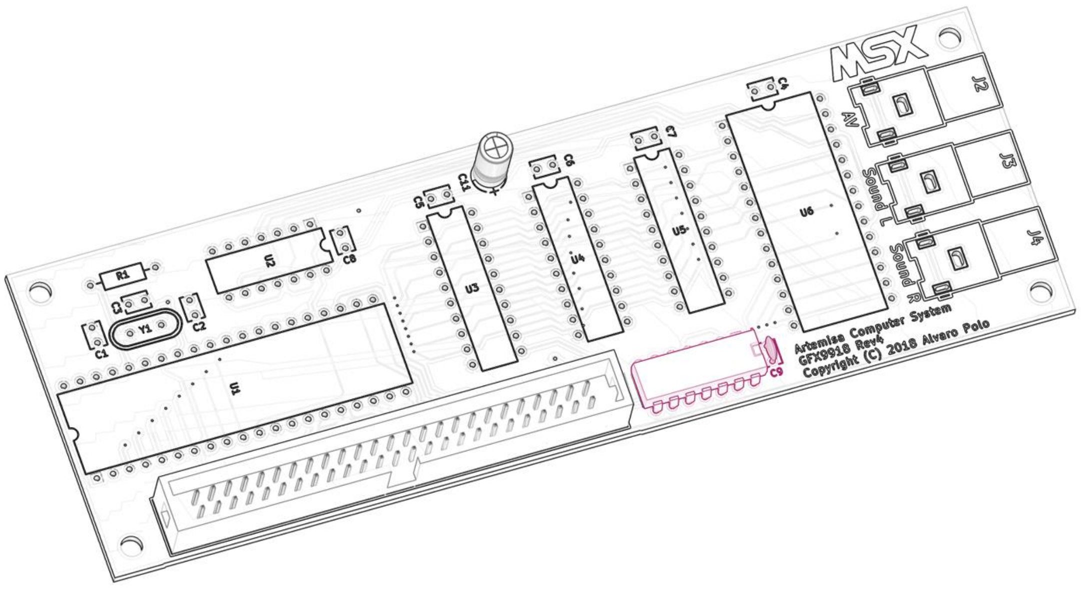


Figure 22: Placement of integrated circuit U7

Video Display Processor

The Video Display Processor is mainly the TMS9918 chip and the associated clock generator circuit, as shown in Figure 23.

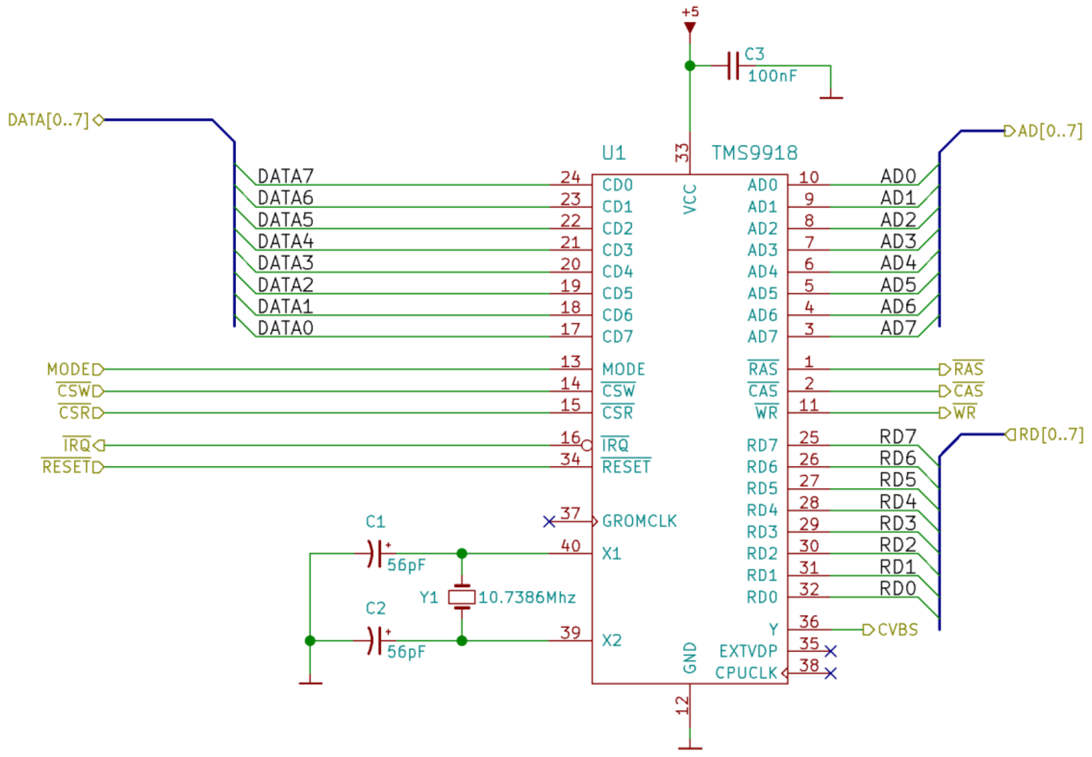


Figure 23: Video Display Processor schematic diagram

The VDP receives the following upstream signals:

- The data bus coming from the CPU. It comes from the graphics card connector.
- The /CSW and /CSR signals, which are generated by the Interface Decoding Logic. Active when write and read operations, respectively, are requested by the CPU.
- The mode signal, which determines whether an IO request for the chip refers to the data register (low) or the command port (high). This is connected to the lowest bit of the address bus, or A0. Thanks to this, the data register will be selected on IO port 0x98 (0b10011000), and command register on port 0x99 (0b10011001).
- The /IRQ signal generated by the VDP. This will be active when the VDP wants to interrupt the CPU. This happens every time a new screen frame is drawn. This signal is directly connected to the graphics card connector.
- The /RESET signal. This comes from the graphics card connector and will be activated when the system boots up or is reset. This is used by the VDP to reset into an initial state.

The VDP also generates the following downstream signals:

- The AD bus, which multiplexes the VRAM higher 7 bits of the address (the Column Address), its lower 7 bits (the Row Address) and the data output (in case of writes) when the VDP accesses its memory.
- The RD bus, which is the input data bus coming from the VRAM. It is only used for reading.
- The /RAS (Row Address Strobe) and /CAS (Column Address Strobe) signals. They indicate a row and a column memory access, respectively. They are part of the memory access sequence performed by the VDP.
- The /WR signal. This is active when the current memory access is for writing, and inactive if it is for reading.
- The CVBS signal. This is the composite video output signal generated by the VDP. This is connected to the Output Connectors.

Memory buses multiplexing

In many computer systems, it is very common to split memory address by row and column. Therefore, you might see the /RAS and /CAS lines present in many other designs.

There are two reasons for this. First, because of the internal construction of the memory chips. They are disposed as a matrix of bit cells. Hence, it is straightforward to assume each bit is accessed by its column and row addresses.

In the other hard, having row and column address is good to reduce the size of IC encapsulation and hence save space in the electronic boards. This is possible because both row and column addresses can be multiplexed in a single bus. Multiplexing means that different information is transmitted on the same channel at different instants. Thanks to this, we can save half of the width of address buses. For example, eight pins are enough to address 64K elements by multiplexing 8-bit row and 8-bit column addresses.

In the case of TMS9918, the 8-bit bus AD multiplexes the row and column addresses, along the output data for write operations.

Along the upstream and downstream signals, there is a clock generation circuit that can be seen in the schematic diagram. This is comprised by a 10.73 Mhz crystal and a pair of ceramic capacitors. They conform a resonator circuit used internally by the VDP to generate the internal clock signal used for its operation.

Please note the lines from the data bus are inversely connected to the VDP. This is because the Texas Instruments notation assigns the first line number (0) to the most significant bit, and the last line number (7) to the less significant bit. The rest of the world does it the other way around. First number (0) is the less significant bit and last number (7) the most significant bit. Thus, the data bus seems to be reversed but it is not.

Step 5

Solder the 40P DIP socket, as shown in Figure 24.

Just solder the socket, but do not plug the VDP chip yet in it. We will not need it at this point. And having it connected while soldering increases the risk of causing damage to the chips by overheat.

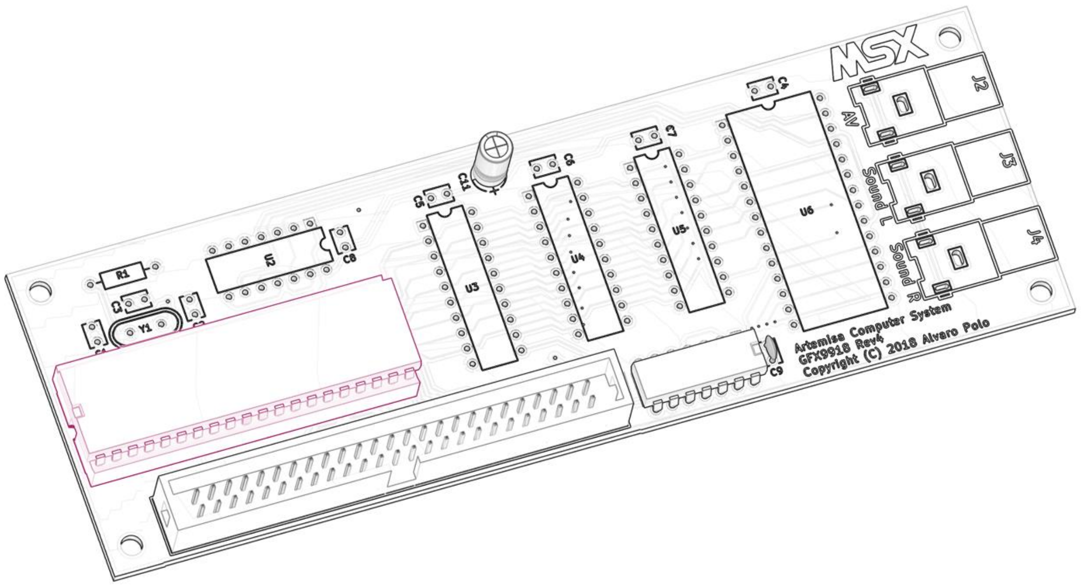


Figure 24: Placement of Video Display Processor

Step 6

Solder the components of the clock signal generation, as shown in Figure 25:

- Crystal oscillator Y1. Value 10.7386 Mhz.
- Capacitors C1 and C2. Value 56 pF.

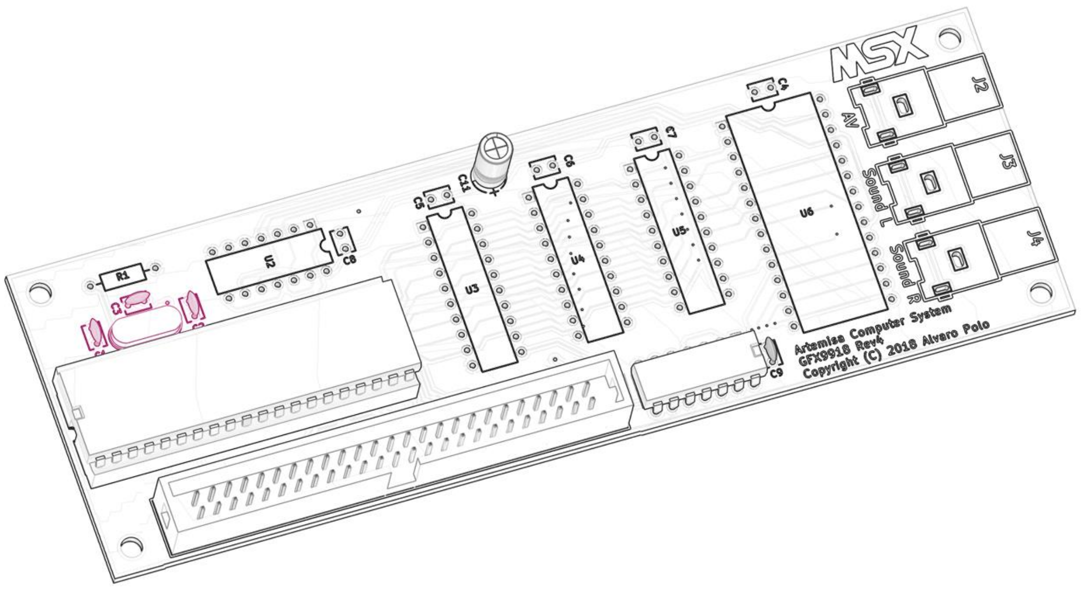


Figure 25: Placement of clock generation circuit

VRAM decoding logic

This is probably the most complex part of the design. This is based upon the work of Tom LeMense[1].

[1] https://cdn.hackaday.io/files/5789247676576/9918-SRAM.pdf

The main goal of this circuit is to strobe the address and data from the AD bus of the VDP at the right instant. As mentioned above, the TMS9918 multiplexes the following information in this buffer while accessing the VRAM:

- The higher 7-bits of the VRAM memory address, which is known as Column Address.
- The lower 7-bits of the VRAM memory address, which is known as Row Address.
- The output data in case the VDP is writing to the VRAM.

The VDP drives the AD bus with each of these items at different instants through a memory access. First, it puts the Row Address, followed by the Column Address, and finally the data to be written in case of a write.

The circuit fetches each item that travels through the AD bus and saves it in three different registers: one for the row address, one for the column address, and one for the data to write. These registers are D-type flip-flops with positive edge-trigger and 3-state output buffers with part name 74HCT574. They are U3, U4 and U5 as shown in the Figure 26.

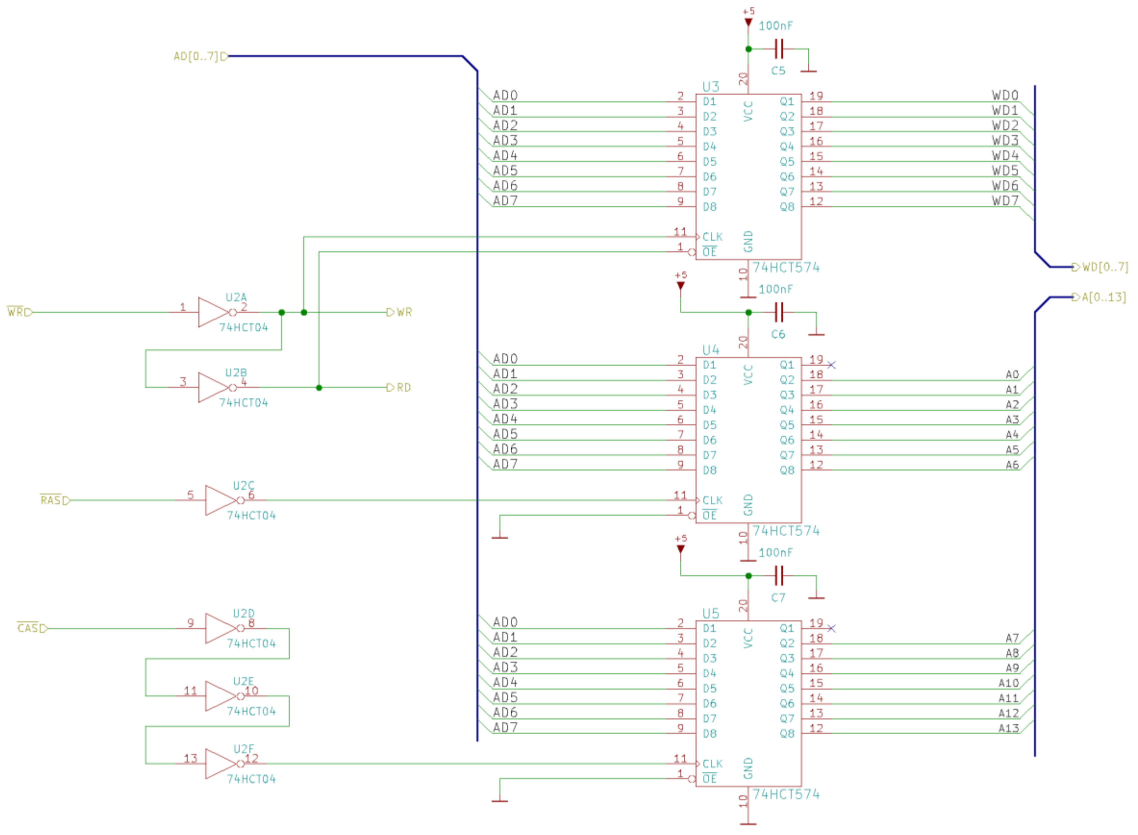


Figure 26: VRAM decoding logic schematic diagram

The flip-flips will demultiplex the signals from AD bus to produce the full 14-bit address in A bus, and the data to be written in the WD bus.

VRAM bus multiplexing

The AD bus is multiplexed in every access of the VDP to the VRAM. For read operations, the /RAS and the /CAS signals determine the contents present in the bus, as shown in Figure 12.

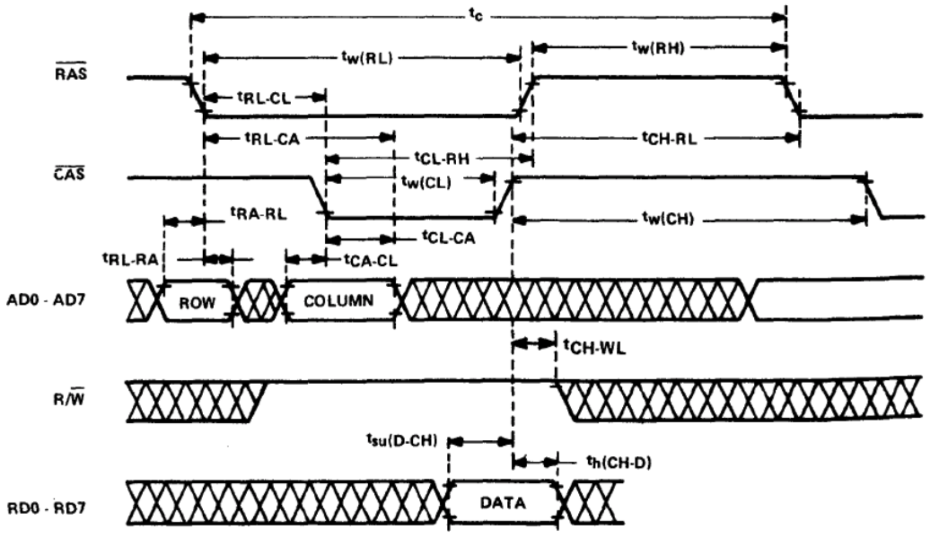


Figure 27: Timing diagram of VRAM read access

- When /RAS activates on its falling edge, the AD bus contains the row address. This signal is inverted and then connected to U4 clock input. This means U4 will register the row address when /RAS is asserted.
- When /CAS activates on its falling edge, the AD bus contains the column address. This signal is inverted and then connected to the U5 clock input. This means U5 will register the column address when /CAS is asserted. The reason to have multiple inverters is to generate large propagation delays from /CAS to the signal input of U5. This is required to satisfy the timing constraints of the VDP.

In case of a write operation, the mechanism is similar. But after /RAS and /CAS signals, the VDP will drive the AD bus with the data to be written, and will assert it with the RW signal, as shown in Figure 28:

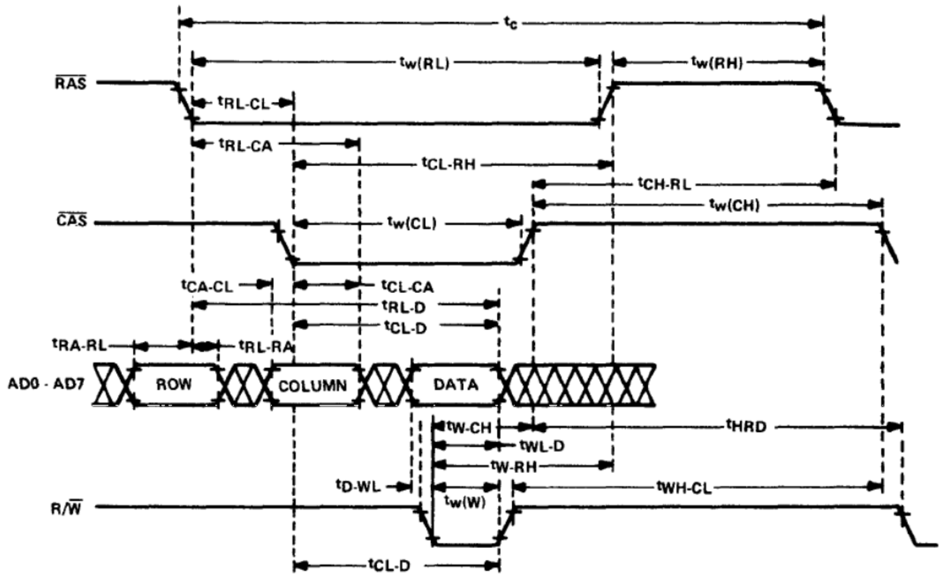


Figure 28: Timing diagram of VRAM write access

The RW signal is inverted and then connected to the clock input of U3. This means U3 will register the data to be written on RW falling edge.

As result, the A bus in the right hand of the schematic diagram contains the full 14-bit address selected by the VDP. The WD bus contains the data to be written. Please note the OE input of U3 will ensure the bus is driven only during the write cycles.

Step 7

Solder the integrated circuits U3, U4 and U5 with their decoupling capacitors C5, C6 and C7 of 100nF, as shown in Figure 29.

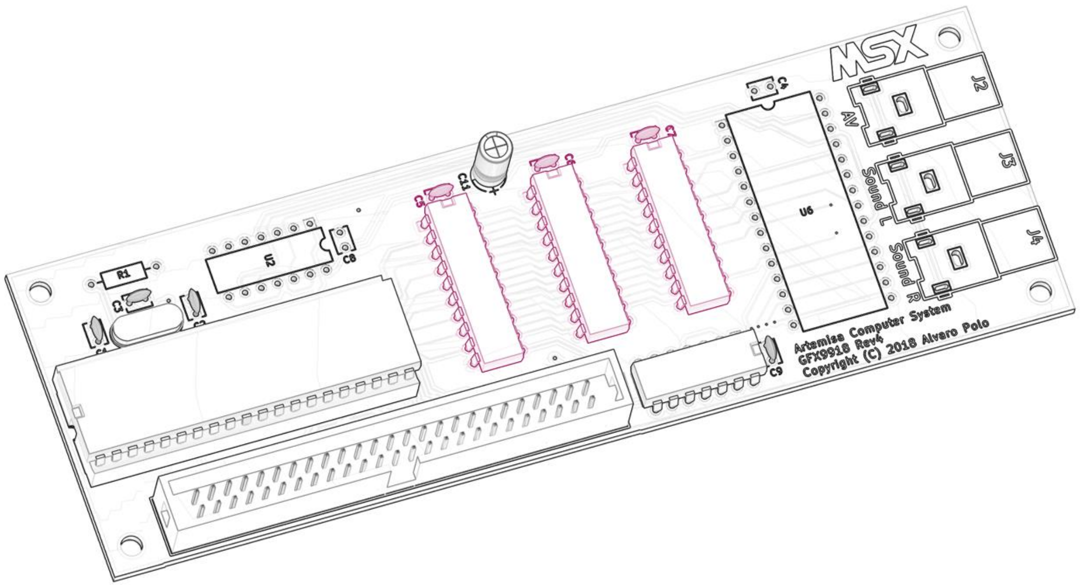


Figure 29: Placement of U3, U4 and U5

Step 8

Solder the integrated circuit U2 and its decoupling capacitor C8 of 100nF, as shown in Figure 30.

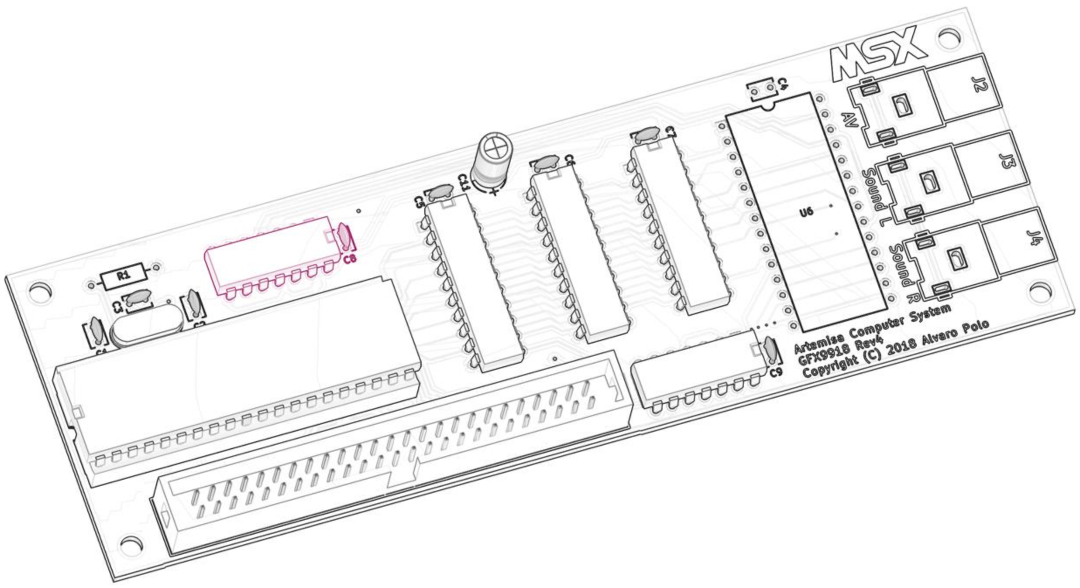


Figure 30: Placement of U2 and C8

Video RAM

The Video RAM circuit is pretty straightforward once the signals generated by the VDP are appropriately decoded. As we have seen, the VRAM decoding logic generates two buses that are directly connected to the VRAM:

- The A bus, which contains a completely demultiplexed 14-bit address.
- The WD bus, which contains the data sent by the VDP for write operations.

As shown in Figure 31, these buses are connected to U6, the static RAM chip with part name 62256.

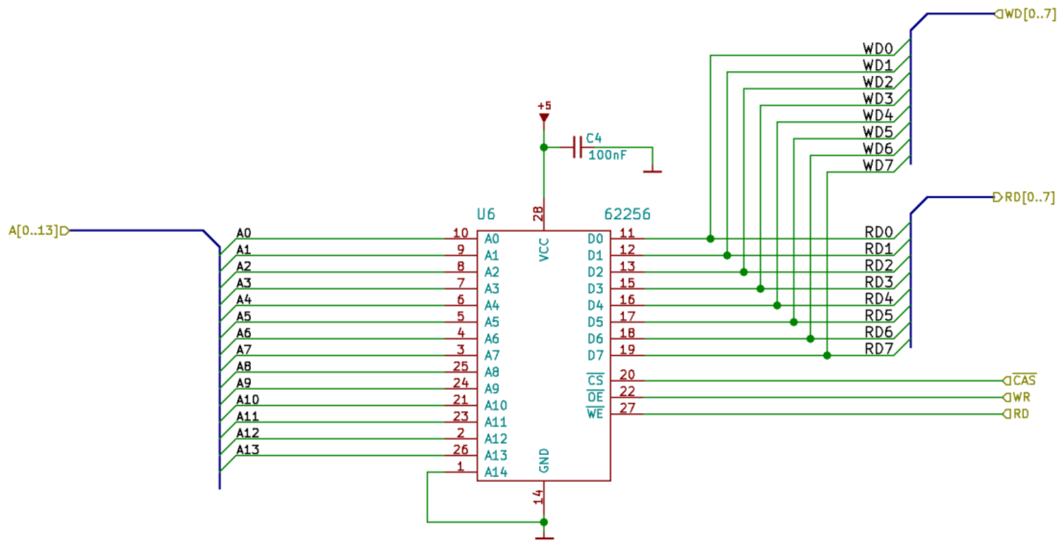


Figure 31: VRAM schematic diagram

In addition, there is another bus RD, which carries the data to the VDP in case of read operations. Both buses, WD and RD are connected to the data pins of the memory chip.

Finally, the memory chip receives the following inputs:

- /CS, the chip select signal. It indicates the memory chip is selected for a read or write operation. This is connected to the /CAS signal produced in the VRAM decoding logic.
- /OE, the output enabled signal. It indicates the memory chip can drive the data bus. This is connected to the WR signal generated by the VRAM decoding logic, which ensures the data bus is driven only during write operations.
- /WE, the write enabled signal. It indicates a write operation when active-low or read operation otherwise.

Step 8

Solder the 28 pin DIP socket in the placement of U6 and the decoupling capacitor C4 of 100nF, as shown in Figure 32:

As we did for the VDP, just solder the socket with the memory chip unplugged in order to prevent any damage caused by overheat.

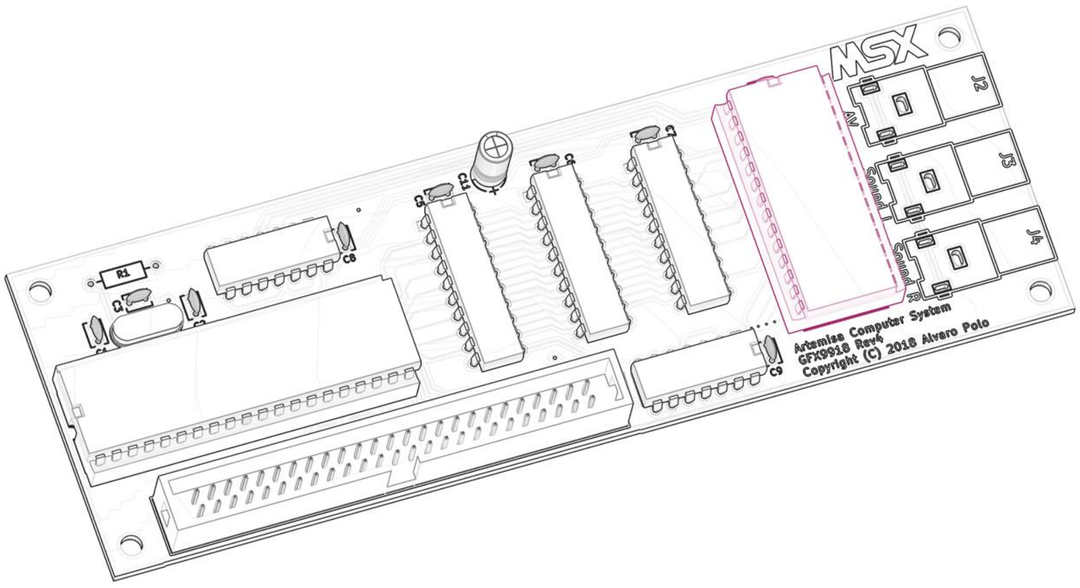


Figure 32: Placement of U6 and C4

Output connectors

Our final stage is to connect the outputs of the circuit to have suitable video and sound through RCA connectors as shown in Figure 33.

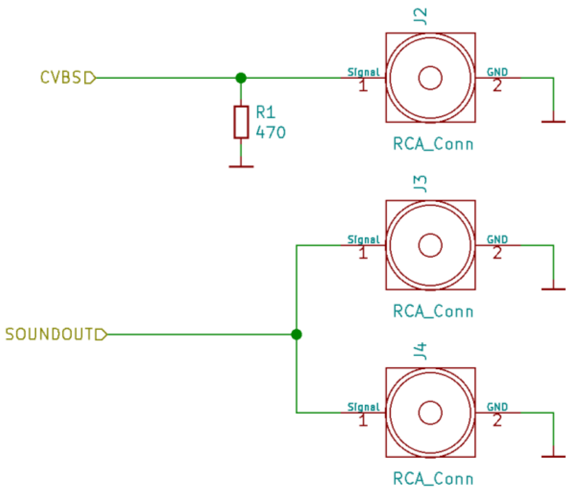


Figure 33: Output connectors schematic diagram

The CVBS signal requires a pull-down resistor of 470Ω to reduce the voltage of the composite video signal. After that, it is directly connected to the yellow RCA terminal.

The sound output from the card connector is directly connected to the white and red RCA terminals for a mono sound setup with two speakers.

These RCA terminals can be connected to a TV set with a NTSC composite video input.

Step 9

Solder the RCA connectors to the board, as shown in Figure 34.

⚠ Mind the colors of the RCA connectors. From the board top: yellow, white and red.

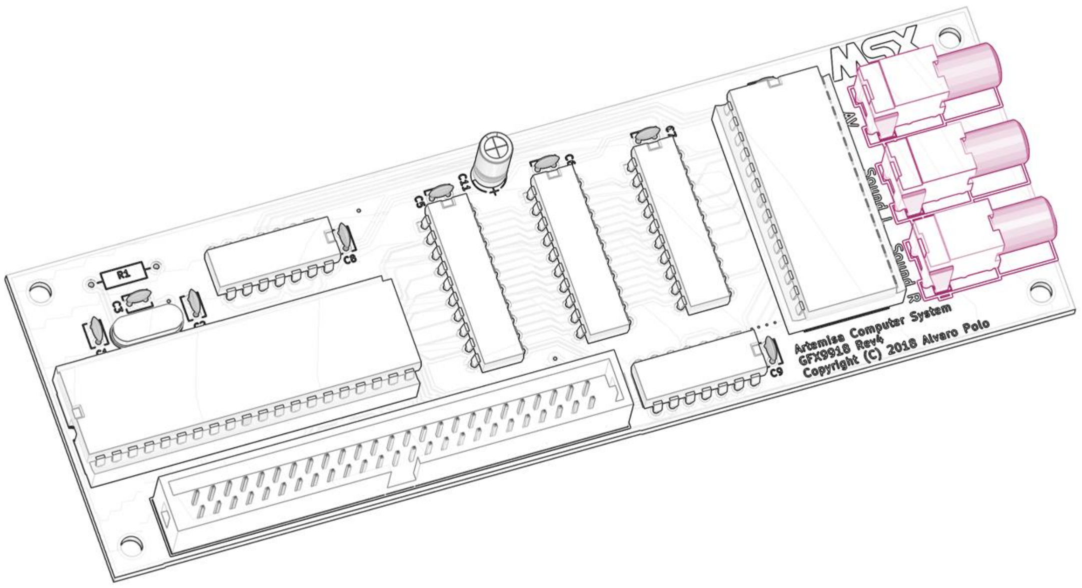


Figure 34: Placement of RCA connectors

Step 10

Solder the resistor R1 of 470Ω as shown in Figure 35.

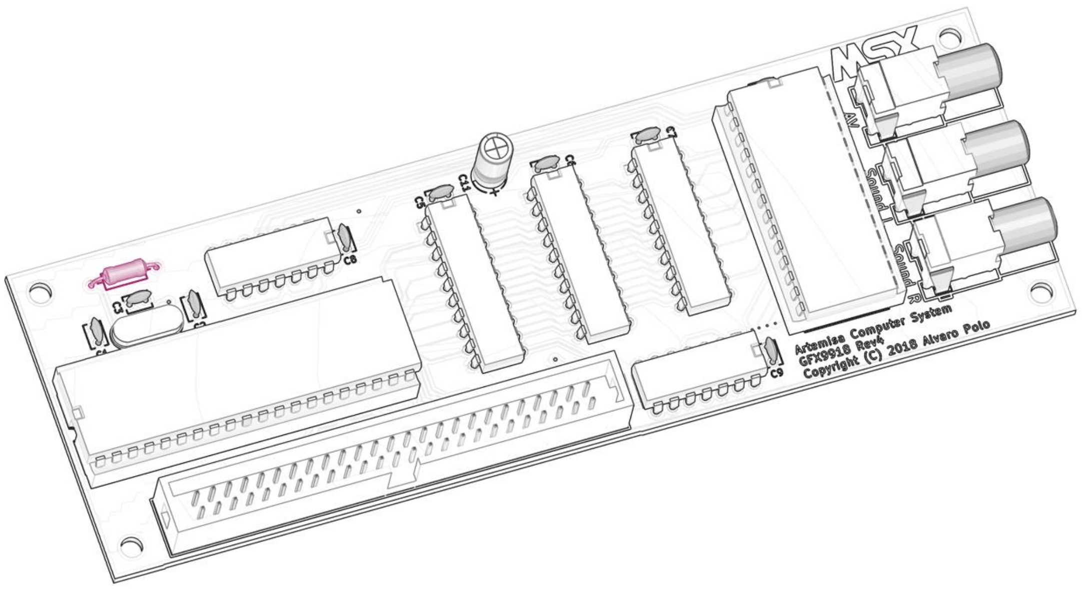


Figure 35: Placement of R1

Step 11

Just one more step and we are done. Now it is time to plug the VDP and the memory chip in their corresponding sockets.

⚠ Make sure the chips are plugged in the right direction. Look for the notch in the IC and the socket and ensure they are in the same side.

Summary

TBD