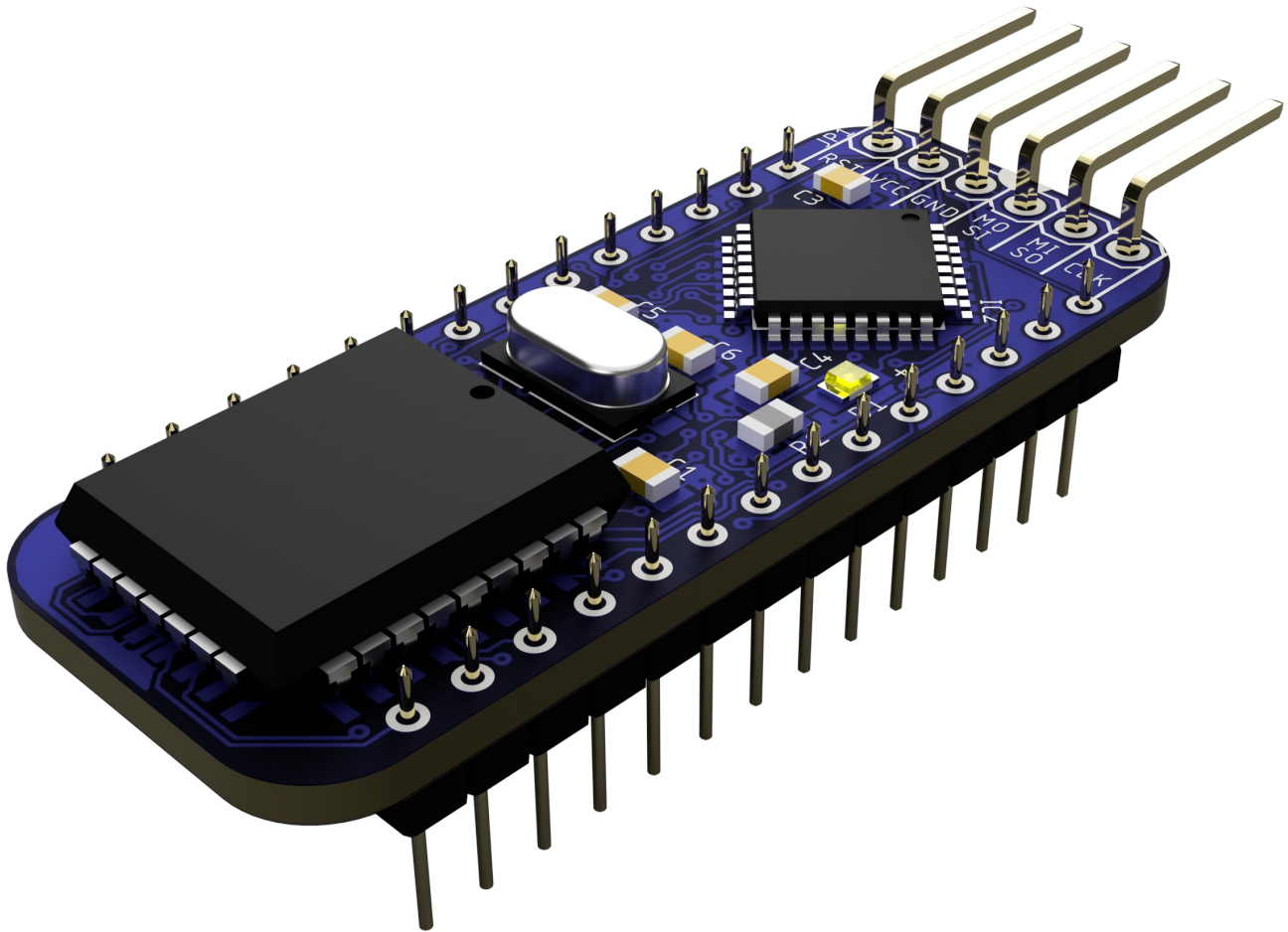


Retroninja Switchless Multi-ROM for 27128 & 27256



Technical Guide

Table of Contents

Background.....	4
Theory of operation.....	4
Usage examples.....	4
Commodore 1541-II/1571/1581 DOS ROM Switcher.....	4
Commodore 64 shortboard/C128/TED series menu-driven Kernal switch.....	6
Assembling the module.....	9
Parts.....	10
Programming the microcontroller.....	11
Programming the flash chip (ROM).....	11
Document revision history.....	11

Background

I first created this ROM switcher as a DOS ROM switcher for the Commodore 1541-II disk drive which uses a 27128 compatible ROM. The space in the 1541-II is very limited. That's why the module is so small and has to use SMD components.

I realized that it can be used for a lot of other equipment that is using similar ROMs so it was re-branded and upgraded to support switching 27256 ROM images too.

Theory of operation

The circuit is meant to sit in a ROM socket instead of a 28-pin 27128/27256 ROM.

The AVR microcontroller (MCU) is connected to the data pins of the ROM socket through a flip-flop that latches the data to give the MCU more time to read the data. It's also connected to some header pins that can be connected to a clock source, a reset line and optionally other signals. It also has an on-board flash chip that contains multiple ROM images.

With the help of an external clock source, the MCU can capture bytes that are passing on the data bus and react to a predefined string of "magic bytes" by switching the upper address pins on the flash circuit to switch to a different ROM image.

The flip-flop triggers on positive-edge of clock and if data is arriving on the bus at the negative edge of the clock signal then the clock source needs to be inverted for the flip-flop to latch the data correctly. This can be done by installing the inverter gate at IC5 and configuring solder jumper SJ1 for inverted clock.

The need for this differs between implementations depending on how the magic bytes ends up on the bus.

The module has two modes: 27128 and 27256 mode. The mode is selected using a solder jumper. In 27128 mode the A14 address pin on the flash chip is controlled by the MCU giving the MCU five address pins (A14-A18) to use which gives a maximum of 32*16KByte ROMs. In 27256 mode the A14 pin is controlled by the host system leaving four address pins (A15-A18) to the MCU which gives a maximum of 16*32KByte ROMs.

An onboard LED can be used for diagnostic blinking or other feedback.

Usage examples

The two use cases I've used it in is for switching JiffyDOS ROM in Commodore disk drives and Kernal in the Commodore 8-bit computers but it should be usable for other things too.

Commodore 1541-II/1571/1581 DOS ROM Switcher

The MCU source code for the 1541-II/1571/1581 ROM switch can be found under applications in the GitHub repository: <https://github.com/RetroNynjah/Switchless-Multi-ROM-for-27128-27256>

At power on, the MCU in the ROM switcher reads address 0 from its internal EPROM to find out which ROM it should select, sets the A14-A18 (or A15-A18) address pins on the flash chip accordingly and resets the drive to make sure it's booted with the correct image. This can be noticed when the drive is powered on and the drive initializes and stops spinning and then spins up again briefly during the switch reset. It then starts listening for a switch command.

Even at a clock speed of 20 MHz, reading and analyzing data on a 1-2MHz data bus can be quite a challenge so the data on the drive data bus is buffered in the switcher by a 74AHCT273 flip-flop that is clocked by the Write signal on the 6502 MPU in the drive. By clocking the data with the R/W signal we filter out irrelevant bus traffic and the flip-flop holds the relevant data until the next R/W cycle. This gives the MCU a little more time to read the data from the flip-flop.

The MCU listens for a predefined sequence of bytes *in reverse order* (MORNR@) and once the sequence is found, it treats the following numeric character as the target ROM image number. The reverse order of the characters on the bus is why the image number must be specified at the beginning of the command. The MCU sets address pins A14-A18 (or A15-A18) on the flash chip according to image selection and it stores the ROM number at address 0 of its internal EPROM for use at next power-on. It then resets the drive by toggling the reset line.

The command can be sent from basic on a computer such as C64 or C128 by utilizing any disk drive command that contains text strings such as:



```
LOAD "2ERNROM",8
```

If the computer is equipped with JiffyDOS and the correct drive has been selected using CTRL-D or similar, DOS wedge commands can be used for switching ROM:



```
@2ERNROM
```

The commands in the above examples will return a file not found error but you should see the drive switch ROM and perform a drive reset and turn of the error LED. The ROM switch LED will blink 2 times to indicate that image 2 was selected.



When building a 1541-II ROM switch you should bridge the solder jumpers like this:

SJ1	
Inverted Clock	Non-inverted Clock
	



Note: To invert clock we must also install IC5

SJ2	
27256	27128
	

And for a 1571 and 1581 ROM switch the solder jumpers should be configured like this:

SJ1	
Inverted Clock	Non-inverted Clock
	

Note: To invert clock we must also install IC5

SJ2	
27256	27128
	

Some pins on the switch needs to be connected to the right signals in the drive. These signals can be connected to the drive using test clips but to have a secure installation it's better to solder the cables to the bottom of the PCB.

Switch pin	Drive signal
CLK	R/W (6502 pin 34)
MOSI	RESET (Middle pin of IEC DIN connector)

This switch only suits the 1541-II/1571/1581 and clones that has 28-pin ROM sockets.

1541 drives and some of it's clones have 24-pin 2364 type ROMs and I have another Multi-ROM version that is compatible with those in the following repository:

<https://github.com/RetroNynjah/Switchless-Multi-ROM-for-2364>

Commodore 64 shortboard/C128/TED series menu-driven Kernal switch

The MCU source code and Mini-Kernal source code for the C64 shortboard/C128 kernal switch can be found under samples in the GitHub repository at:

<https://github.com/RetroNynjah/Switchless-Multi-ROM-for-27128-27256>

At power on, the MCU in the Kernal switch reads address 0 from its internal EPROM to find out which Kernal it should select, sets the A14-A18 (or A15-A18) address pins on the flash chip accordingly and resets the drive to make sure it's booted with the correct image.

If at any time, restore is pressed and held for a few seconds the MCU will detect this and switch to Kernal 0 which is a menu Kernal. The C16/116/Plus4 menu is activated by holding RESET for a few seconds and then releasing it.

For this application I created custom Mini-Kernels for the C64, the C128 and the C16/C116/Plus4 that display lists of Kernals to choose from.



When the user selects a Kernal image from the menu, the Mini-Kernal writes a predefined switching command to RAM along with a byte that indicates the selected image number. It does this over and over. Example of the command: RNR0M64#1, RNR0M128#1 or RNR0M+4#1 where 1 is a byte with value 0x01 that tells the switch that we want to switch to kernal image 1.

The Kernal switch now captures the command from the data bus, writes the selected kernal number to the MCU EPROM for future use, then it switches the address pins A14/A15-A18 accordingly. It then pulls the reset line briefly to make the computer restart using the new Kernal. The computer will continue to start up using this Kernal until a new choice is made.

Having different commands for the platforms makes it possible to equip both the ROM sockets in a Commodore 128 (not CR/DCR) with Kernal switches and keep them separated.

In this case both switches will trigger their menus when RESTORE is held as the MCUs don't know if they are being addressed or not. The MCUs will switch back to the previous image if they detect that a reset is being performed by something other than themselves.

To activate the C64 switch menu in a C128 you hold the CBM key together with RESTORE. When selecting an image from the C64 kernal menu you hold CBM while you press RETURN to make it boot into C64 mode with the new Kernal.

The C128CR/DCR doesn't have a separate ROM for the C64 part and doesn't need an separate switch for C64 mode but it needs to have the C64 Basic and Kernal images bundled in the same flash ROM as the C128 Kernals. This switch will switches Kernals for both C64 mode and C128 mode at the same time. This can be both good or bad depending on how you combine your Kernals.

It also needs to have the Mini-Kernals modified so that both Mini-Kernals uses the same switching command such as RNRMDCR#1 and have an MCU firmware compatible with 27256 switching.

For the C128, the flash is populated with a 16 KB C128 Mini-Kernal followed by up to 31*16 KB Kernals (good luck finding more than a couple).

For the C64, the flash is populated with a 16 KB combination of 8 KB basic ROM and 8 KB Mini-Kernal followed by up to 31*16 KB Basic+Kernal combinations.

Here's an example ROM layout for a C64 shortboard or the C64 Kernal in the C128 that fits an SST39SF010A:

Basic 8kB	Mini-Kernal 8kB	Basic 8kB	CBM Kernal 8kB	Basic 8kB	JiffyDOS US 8kB	Basic 8kB	JiffyDOS SE 8kB	Basic 8kB	JiffyDOS DK 8kB	Basic 8kB	JaffyDOS 8kB	Basic 8kB	DolphinDOS 8kB	Basic 8kB	SpeedDOS 8kB
16kB		16kB		16kB		16kB		16kB		16kB		16kB		16kB	

Example for a C128 with an SST39SF010A:

Mini-Kernal 16kB	CBM Kernal 16kB	JiffyDOS US 16kB	JiffyDOS SE 16kB	JiffyDOS DK 16kB	Mini-Kernal 16kB	Mini-Kernal 16kB	Mini-Kernal 16kB
---------------------	--------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------

Example for a C128CR/DCR with an SST39SF010A:

C64 Basic 8kB	64 Mini 8kB	128 Mini 16kB	C64 Basic 8kB	64 CBM kernal 8kB	128 CBM kernal 16kB	C64 Basic 8kB	64 Jiffy US 8kB	128 Jiffy US 16kB	C64 Basic 8kB	64 Jiffy SE 8kB	128 Jiffy SE 16kB
16kB		16kB	16kB		16kB	16kB		16kB	16kB		16kB

As both C64 mode and C128 mode Kernals are in the same ROM a larger flash chip may be needed to accommodate all Kernals. If you have more C64 Kernals than C128 Kernals you still need to fill the vacant C128 slots with working Kernals such as a stock CBM Kernal or you will never be able to start the C128 into C64 mode either.

When building a C64/C128/TED-series Kernal switch you should bridge the solder jumpers like this:

SJ1	
Inverted Clock	Non-inverted Clock

Note: As we don't invert the clock we don't need to install IC5 either

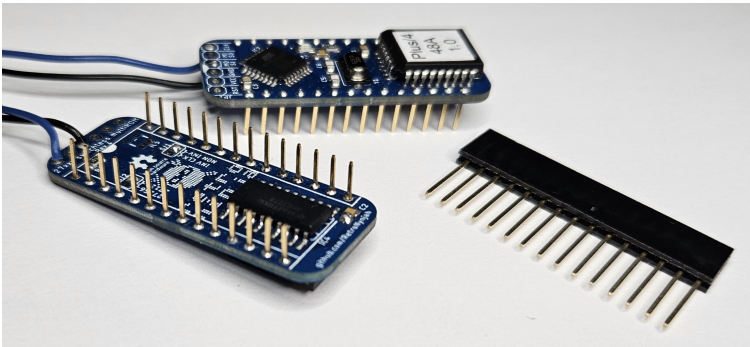
SJ2 (all except CR/DCR)	
27256	27128

SJ2 (C128CR/DCR)	
27256	27128

Some pins on the switch needs to be connected to the right signals in the computer. These signals can be connected to various ICs in the computer using test clips but to have a secure installation it's better to solder the cables to vias on the PCB. See pictures in the samples directory of the GitHub repository.

Switch pin	Computer signal	Comments
CLK	$\overline{R/W}$	
MISO	$\overline{RESTORE}$	Only on C64/C128
MOSI	\overline{RESET}	

When building a Kernal switch for the Plus/4 (or C116) you will need to keep the profile low by using the flat kind of headers that are usually sold as “Stackable Arduino headers”.



Optionally you can remove the Kernal socket and solder the switch directly to the motherboard using any kind of header.

This switch when configured as a C64 Kernal switch fits the C64 shortboard (PCB 250469) and the C64 part of the C128. C64 longboards have 24-pin 2364 type Kernal ROMs and I have another Multi-ROM version that is compatible with those in the following repository:

<https://github.com/RetroNynjah/Switchless-Multi-ROM-for-2364>

Assembling the module

The PCB is small and a bit difficult to solder but it can be hand soldered. The reason for the small size is space constraints in the Commodore 1541-II disk drives that this PCB was initially designed for. Other drives such as 1571 and 1581 have the same constraints.

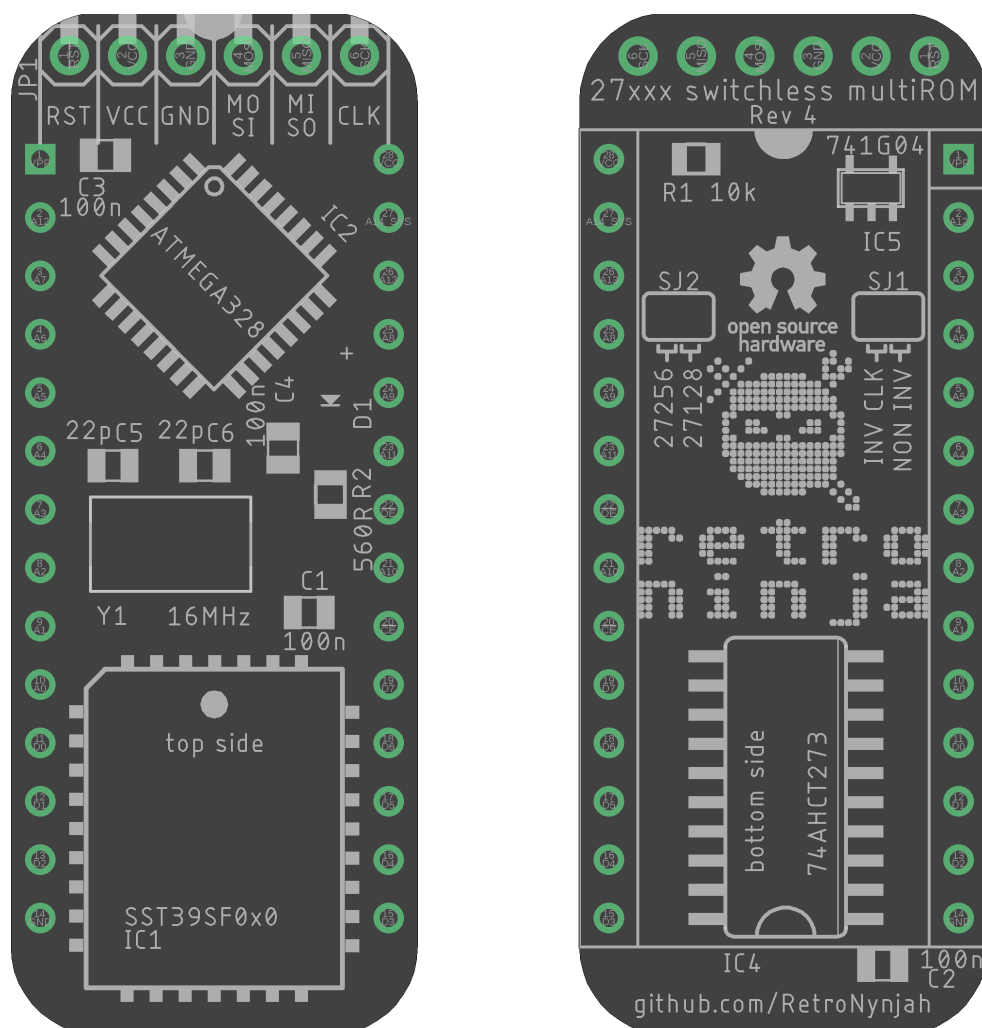
The flash chip (PLCC) is the trickiest part to solder and I recommended reflow soldering the top side using stencil and solder paste. Then solder the bottom side by hand and finally solder the pin headers.

There are two solder jumpers on the bottom that must be configured.

If you configure SJ1 for inverted clock then you must also install IC5. Otherwise you can leave it out. It will only be used when SJ1 is configured for inverted clock.

Note: Don't forget to program the flash before soldering!

Parts



Top side is to the left, bottom side is to the right.

Component	Part	Description	Comment
JP1	Header	6-pin right-angle	
D1	LED	0805 LED	
C1, C2, C3,C4	100nF capacitor	0805 ceramic	
C5, C6	30pF capacitor	0805 ceramic	22pF works fine too
Y1	ECS-200-20-3X-TR	20MHz Crystal	16MHz is ok for 1MHz systems
R1	10kΩ resistor	0805 Resistor	
R2	560Ω resistor	0805 Resistor	
IC1	SST39SF0x0	32-PLCC Flash 4.5-5.5V	SST39SF010/20/40
IC2	ATmega MCU	32-TQFP MCU	Atmega48/88/168/328
IC3	Headers	2 header rows, 14 pin	Use machined or flat pins - not square
IC4	74AHCT273	20-SOIC 5.3mm Flip-Flop	
IC5	741G04	SOT23-5 1 x Inverter	Only needed if clock needs inverting

The ATmega model is not critical. The firmware requires less than 2KB of MCU flash. I have tested Atmega48, 48A, 48PA, 48PB, 88A, 328P and they have worked for me. Any other pin-compatible ATmega could work too depending on your application but it must be able to run at 20MHz if you are going to use it in a 2MHz system such as 1571/1581. For 1MHz systems it is enough with 16MHz.

Programming the microcontroller

The firmware can be flashed using the 6-pin ISP header. The source code is written for Arduino and programming the firmware can be done directly from the Arduino IDE using an ISP programmer.

If you are using an Atmega328 it can be programmed as an Arduino UNO. The other variants require custom board definitions. I recommend the DIY AVR cores that can be installed using the Arduino Boards Manager (<https://github.com/sleemanj/optiboot>).

If you can't or don't want to use Arduino I have some pre-compiled hex files for the Kernal switch and drive ROM switch along with fuse configurations and syntax examples for how to program them using avrdude. These can be found under the Applications folder in my GitHub repository.

Programming the flash chip (ROM)

The flash chip needs to be programmed before soldering it. If you need to reprogram the flash later it must first be de-soldered and then re-soldered again after reprogramming.

You need to concatenate all your images into one file and fill up any empty space in flash with valid extra images to avoid bricking your host device if an incorrect image number is selected. If that would happen the EPROM can be reset by re-flashing the firmware.

In 27128 mode all images should be 16KB and in 27256 mode all the images need to be 32KB.

Document revision history

1.0 Initial version

1.1 BOM Corrected

1.2 Updated BOM with 20MHz crystal, changed AVR core and added information about 1571 & 1581

1.3 Updated kernal switch section with C16, C116 and Plus/4

