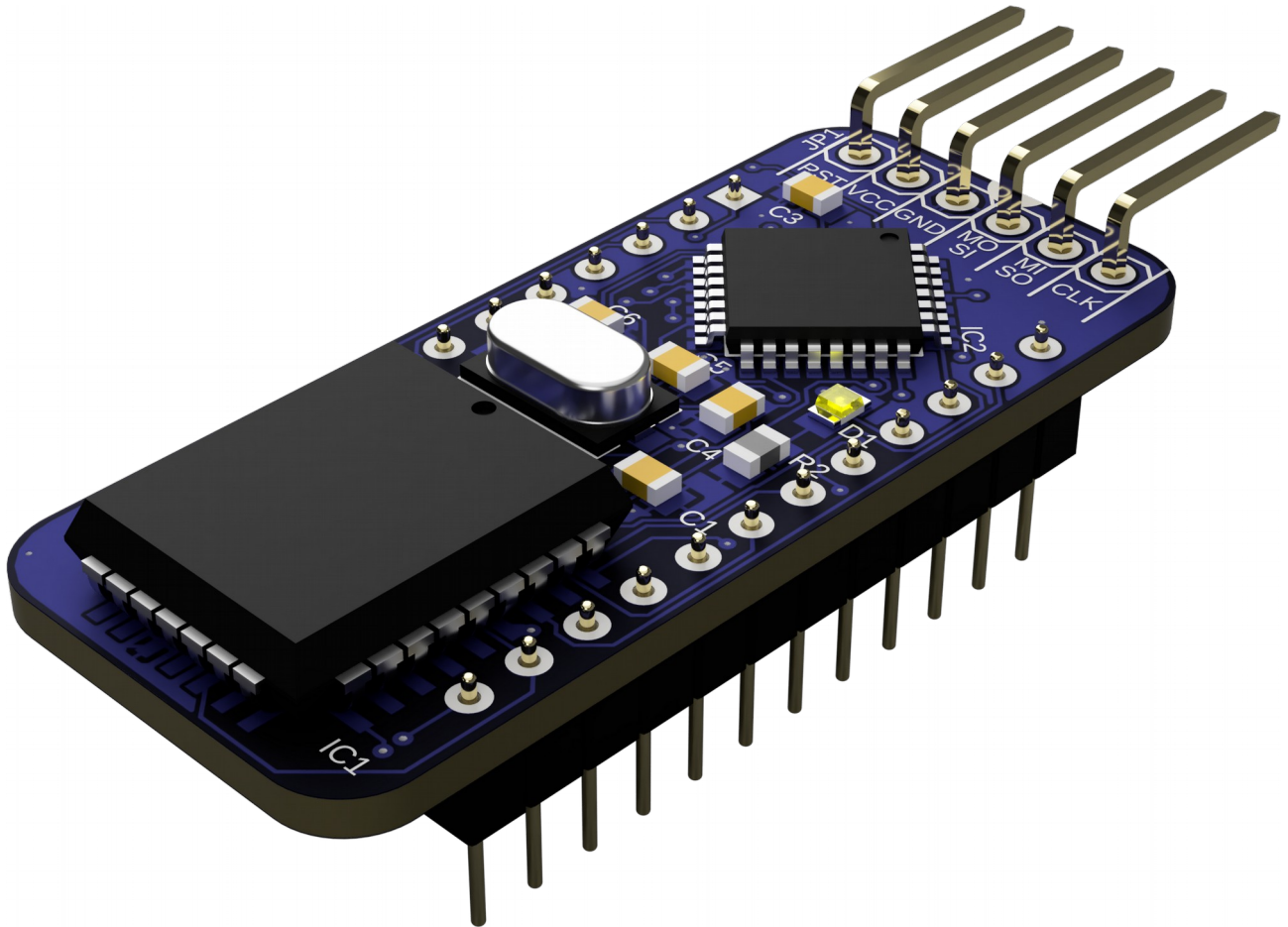


Retroninja Switchless Multi-ROM for 2364



Technical Guide

Table of Contents

Background.....	4
Theory of operation.....	4
Usage examples.....	4
Commodore 1541 DOS ROM Switcher.....	4
Commodore 64 longboard menu-driven Kernal switch.....	5
Commodore VIC-20 menu-driven Kernal switch.....	7
Assembling the module.....	8
Parts.....	9
Programming the microcontroller.....	10
Programming the flash chip (ROM).....	10
Document revision history.....	10

Background

I first created this ROM switcher as a DOS ROM switcher for the Commodore 1541 disk drive which in turn was derived from a Commodore 1541-II drive ROM switch that I had previously designed.

I realized that it can be used for a lot of other equipment that is using similar ROMs so it was re-branded and modified making the clock inverter optional.

Theory of operation

The circuit is meant to sit in a ROM socket instead of a 24-pin 2364 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 address pins above A12 (A13-A18) are controlled by the MCU which gives a maximum of 64*8KByte switchable 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 DOS ROM Switcher

The MCU source code for the 1541 ROM switch can be found under samples in the GitHub repository at: <https://github.com/RetroNynjah/Switchless-Multi-ROM-for-2364>

At power on, the MCU in the ROM switcher reads address 0 from its internal EEPROM to find out which ROM it should select, sets the A13-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 16 MHz, reading and analyzing data on a 1MHz data bus can be a challenge so the data on the 1541 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 1541. 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 A13-A18 on the flash chip according to image selection and it stores the ROM number at address 0 of its internal EEPROM for use at next power-on. It then resets the drive by toggling the IEC 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
```

The LOAD command in the example will return a file not found error but you should see the drive switch ROM and perform a drive reset. The ROM switch LED will blink 2 times to indicate that image 2 was selected.

When building a 1541 ROM switch you should bridge the solder jumper like this:

SJ1	
Inverted Clock	Non-inverted Clock
	

Note: To invert clock we must also install IC5

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 and clones that has 24-pin ROM sockets.

1541C/1541-II drives and some clones have 28-pin 27128 type ROMs. I have another Multi-ROM version that is compatible with those in the following repository:

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

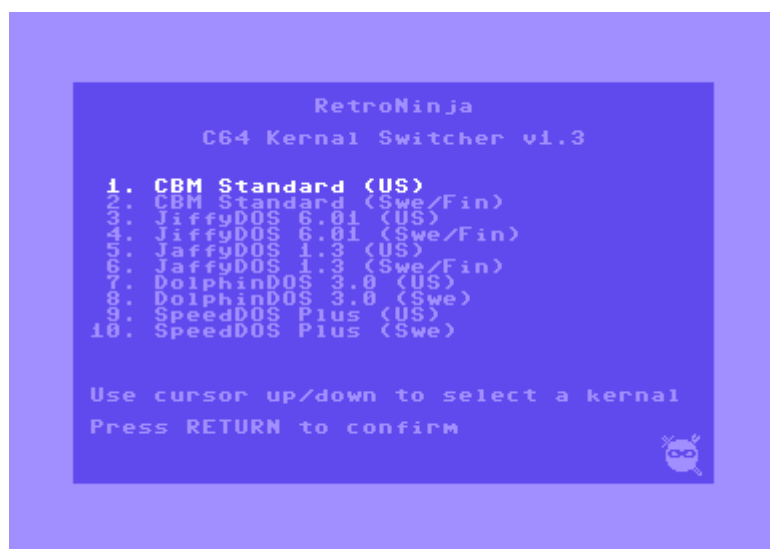
Commodore 64 longboard menu-driven Kernal switch

The MCU source code and Mini-Kernal source code for the C64 longboard kernal switch can be found under applications in the GitHub repository at:

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

At power on, the MCU in the Kernal switch reads address 0 from its internal EEPROM to find out which Kernal it should select, sets the A13-A18 address pins on the flash chip accordingly and resets the computer 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. For this application I created a custom Mini-Kernal for the C64 that displays a list of Kernals to choose from.



When the user selects a Kernel 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: RNROM64#1 where 1 is a byte with value 0x01 that tells the switch that we want to switch to kernel image 1.

The Kernel switch now captures the command from the data bus, writes the selected kernel number to the MCU EEPROM for future use, then it switches the address pins A13-A18 accordingly. It then pulls the reset line briefly to make the computer restart using the new Kernel. The computer will continue to start up using this Kernel until a new choice is made.

The flash is populated with the Mini-Kernal followed by up to 10*8 KB Kernels. It would in theory be possible to add up to 63 kernels if both the Mini-Kernal and the MCU firmware was modified in some way to accommodate that many Kernel images.

Here's an example layout that fills out an SST39SF010A:

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

If the same layout would be used with a larger flash chip it should be duplicated to fill out the flash.

When building a C64 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

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
CLK	R/ \overline{W}
MISO	$\overline{\text{RESTORE}}$
MOSI	$\overline{\text{RESET}}$

This C64 Kernal switch only suits the C64 longboards which have a 24pin ROM and has basic in a separate ROM. All C64 motherboards are longboards except for PCB 250469 which can be found in newer C64C's. The C64 shortboard (PCB 250469) have 28-pin 27128 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-27128-27256>

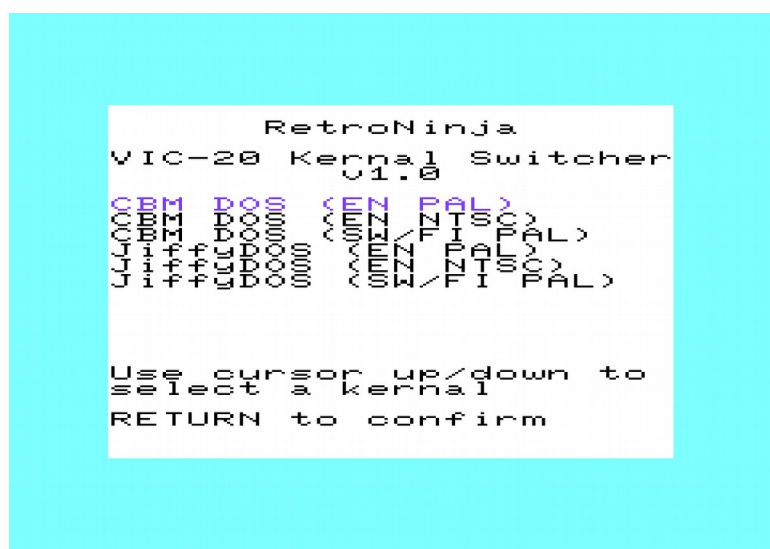
Commodore VIC-20 menu-driven Kernal switch

The MCU source code and Mini-Kernal source code for the VIC-20 kernal switch can be found under applications in the GitHub repository at:

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

At power on, the MCU in the Kernal switch reads address 0 from its internal EEPROM to find out which Kernal it should select, sets the A13-A18 address pins on the flash chip accordingly and resets the computer 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. For this application I created a custom Mini-Kernal for the VIC-20 that displays a list 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: RNR0M20#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 EEPROM for future use, then it switches the address pins A13-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.

The flash is populated with the Mini-Kernal followed by up to 10*8 KB Kernals. It would in theory be possible to add up to 63 kernals if both the Mini-Kernal and the MCU firmware was modified in some way to accommodate that many Kernal images.

Here's an example layout that fills out an SST39SF010A:

Mini-Kernal 8kB	CBM EN PAL 8kB	CBM EN NTSC 8kB	CBM SE PAL 8kB	Jiffy EN PAL 8kB	Jiffy EN NTSC 8kB	Jiffy SE PAL 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB	Mini-Kernal 8kB
--------------------	-------------------	--------------------	-------------------	---------------------	----------------------	---------------------	--------------------	--------------------	--------------------	--------------------	--------------------	--------------------	--------------------	--------------------	--------------------

If the same layout would be used with a larger flash chip it should be duplicated to fill out the flash.

When building a VIC-20 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

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
CLK	R/\overline{W}
MISO	$\overline{\text{RESTORE}}$
MOSI	$\overline{\text{RESET}}$

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 some Commodore drives that this module was designed for originally. Other drives any devices may have the same constraints so keeping it small is good for compatibility.

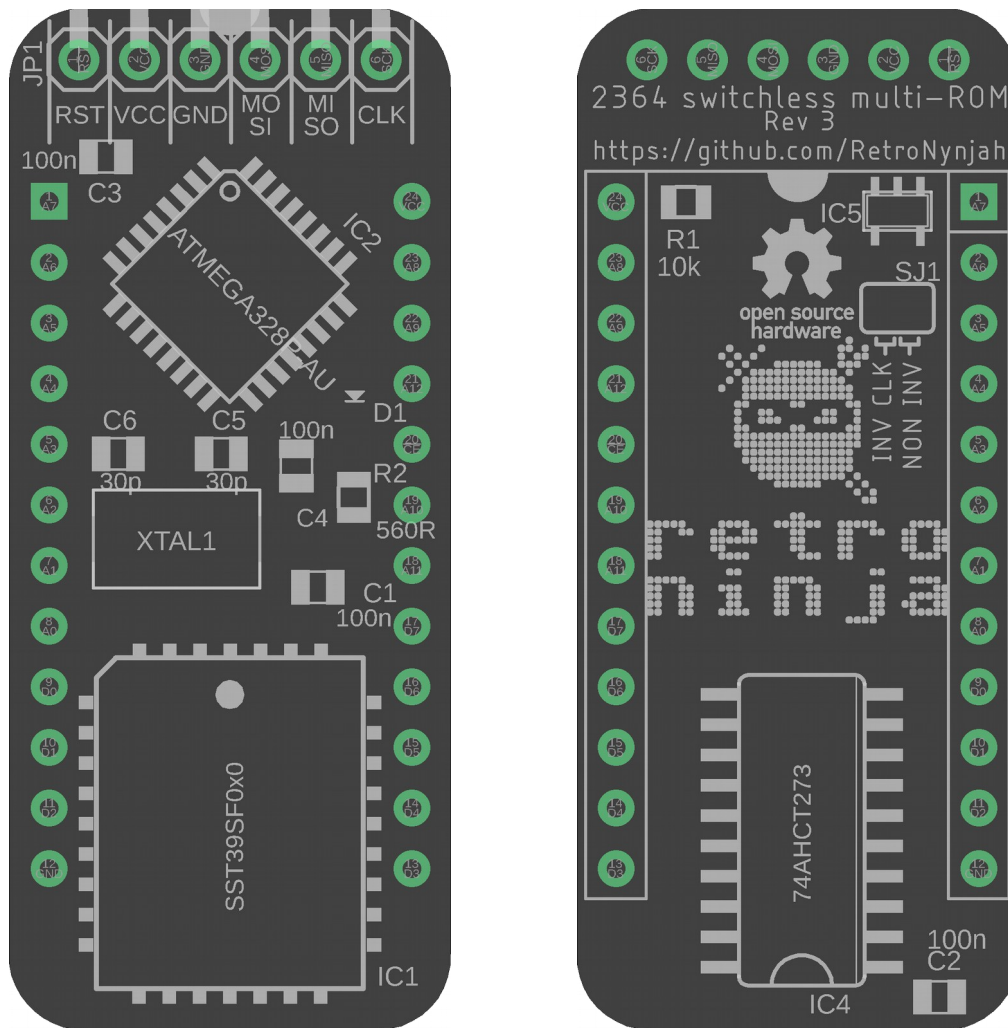
The flash chip (PLCC) is the trickiest part to solder and If you find it hard to hand solder I recommended reflow soldering the top side using stencil and solder paste. Then solder the bottom side by hand and finally solder the pin headers.

There is a solder jumper 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 capacitor	
C5, C6	30pF capacitor	0805 ceramic capacitor	22pF works fine too
XTAL1	ECS-160-20-3X-TR	16MHz Crystal 7.0x4.1 mm	
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, 88A, 328P and they have worked for me. Any other pin-compatible ATmega could work too depending on your application.

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 have used the MiniCore from MCUdude (<https://github.com/MCUdude/MiniCore>) with default board settings.

If you can't or don't want to use Arduino I have some precompiled 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 Examples 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 desoldered and then resoldered 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 EEPROM can be reset by re-flashing the firmware.

All images must be 8kB in size.

Document revision history

- 1.0 Initial version
- 1.1 Cover page image corrected
- 1.2 BOM Corrected
- 1.3 VIC-20 Kernal switch section added

