**C64 Keyboard Controlled Kernal Switch Rev. 1**

**Module Description**

Table of Content

[Introduction 1](#_Toc29834839)

[Usage of the Kernal Switch 2](#_Toc29834840)

[The Unstoppable (EXROM) Reset 3](#_Toc29834841)

[Assembly 3](#_Toc29834842)

[Troubleshooting 4](#_Toc29834843)

[Keyboard Monitoring 5](#_Toc29834844)

[Programming Model 6](#_Toc29834845)

[Connectors 6](#_Toc29834846)

[J1 - Keyboard connector to mainboard 6](#_Toc29834847)

[J2 – In System Programming 6](#_Toc29834848)

[J3 – Serial/Debug 7](#_Toc29834849)

[J4 – IO-Connector 7](#_Toc29834850)

[J5 - Keyboard connector 7](#_Toc29834851)

[Documents 9](#_Toc29834852)

[Revision History 9](#_Toc29834853)

[Rev. 0 9](#_Toc29834854)

[Rev. 0 → Rev. 1 9](#_Toc29834855)

# Introduction

The C64 Keyboard Controlled Kernal Switch is a module that passively watches the scan activity of the C64 keyboard and is capable of setting the higher address bits for Kernal adaptors, similar to project number 123 C64 Kernal-Adaptor/Switch for *Short Board*s that can address up to 7 Kernals, project number 124 C64 Kernal-Adaptor/Switch for *Long Boards* that can address up to 8 Kernals and/or the Character Set Adaptor project number 126 C64 CHARSET-Adaptor/Switch that can address 16 character sets.

A signal is provided to reset the C64 while/after selecting the Kernal via the three most significant EPROM address bits. Two other GPIO-pins (and RESIO) of the processor Atmel ATmega328p are routed to the pin header J4. Firmware v0.2 uses RESIO for driving a signaling “Power” LED.

The ATmega328p can be configured with an Arduino Uno. A pin header provides access to the serial interface for uploading the compiled Arduino sketches. Initial programming is performed via the ICSP connector J2.

The board fits between the keyboard connector of the C64 mainboard and the keyboard itself. All keyboard signals are interconnected and some keyboard signals are monitored by the micro controller.

Watch a brief overview video: <http://bit.ly/C64KbKernalSwitch-Intro>

# Usage of the Kernal Switch

To make use of the Kernal Switch, more than one Kernal should be programmed to the EPROM on the EPROM adaptor. The Kernal can be selected by pressing RESTORE and a number key simultaneously. The Kernal, determined by the number will be selected and the C64 will be reset. This selection is non-volatile and will be effective even after switching off the C64.

Holding down the RESTORE key for about 2-3 seconds and then releasing it (without pressing a number), a reset of the C64 is performed. In case the Power LED is connected to the Kernal Switch as well, this LED will switch off for a short time, after the time for a normal RESET is elapsed (software v0.2 or later). Holding down the RESTORE key for longer (about 4 seconds), the “unstoppable” EXROM reset will be performed (software v0.1 or later).

In case an empty Kernal memory slot is selected, the C64 will crash, since there is no Kernal found. In this case, the C64 does not perform a keyboard scan and since the Kernal Switch is only listening to the keyboard scanning, but does not perform any keyboard scan, the Kernal number could not be set manually anymore. After several seconds without keyboard scan, the Kernal switch will automatically recover to the first Kernal and reset the C64. This will recover the system.

Some cartridges will not perform a keyboard scan as well (such as the Dead Test cartridge). The recovery mechanism will conflict with this kind of cartridge and reset the C64 after several seconds. To circumvent this problem, it is possible to deactivate the Kernal switch by pressing RESTORE 0 (zero). This will switch to the first Kernal and set a non-volatile flag. After switching off the C64 and inserting the cartridge, no keyboard activity timeout will be performed, the cartridge will work normally.

After the cartridge was removed, the first Kernal will be executed and (of course) a normal keyboard scan will be performed. Selecting a Kernal with RESTORE and a number will activate the Kernal Switch again and the new selection will be effective.

|  |  |
| --- | --- |
| **Keys** | **Action** |
| RESTORE (hold >1.5sec, <3sec) | Normal Reset |
| RESTORE (hold > 3sec) | EXROM Reset |
| RESTORE <1>…<7> | Short board: Select Kernal number 1…7 |
| RESTORE <1>…<8> | Long board: Select Kernal number 1…8 |
| RESTORE <0> | Select save Kernal (the first), switch off keyboard scan time out (for cartridges) |

<0>: 0 key (zero)

<1>: 1 key

…

<8>: 8 key

# The Unstoppable (EXROM) Reset

This sort of reset will also assert the -Signal (which has to be connected to J4, Pin 5). The background for this is a trick, that some software (e.g. Gyruss, PAC-MAN or Poltergeist) tries to prevent a normal reset of the C64 as kind of a copy/hacking protection. A (fake) cartridge signature is written to the RAM at address $8000 (HEX) and the following bytes. This consists of the cold start vector, the NMI redirection vector and a “CBM80” signature in RAM. On a normal reset, the Kernal finds this signature in RAM and redirect the vectors, mentioned before. This way, after playing these games, the RESET does not work well and other games cannot be loaded. With the EXROM reset, a normal boot can be accomplished. Nevertheless, the signature stays in RAM (even a couple of seconds without power!!!). The next reset will still get redirected by this fake cartridge signature.

The destruction of the signature is required and can be accomplished by the instruction

POKE32772,0

and another (normal or EXROM) reset. This POKE changes the C of CBM80, on boot up, this signature is not found and the cold start and NMI vectors are not modified. The C64 is working again.

It was observed, that the fake signature stays intact even after several seconds (maximum time was >1 minute) of power off. The C64 might boot up to a black screen or even shows parts of the previous game. This has created some spooky experience after playing Poltergeist. An EXROM reset and the procedure described before will fix this problem. Also switching off the C64 for a minute or longer will help.

Some cartridges hold the EXROM signal HIGH. In this case, the EXROM reset might not work. To prevent a short circuit, the 220Ω resistor mentioned before is inserted into the signal on board.

The Final Cartridge III (+) does not work with the EXROM reset. It has an excellent reset switch, anyways.

# Assembly

It is assumed, that the person who assembles the Kernal Switch is experienced in soldering. First the SMD parts should be soldered, the bottom entry receptacle J1 should be soldered last can be aligned to the holes in the board using the pin header J5. Pin2 of J2 has to be clipped (refer to Figure 1) and the connector has to be inserted into the holes from the solder side. Then J1 has to be placed over those pins and the connector has to be aligned in a way that it is centered on the J1 silk screen drawing and the SMD pins are in the middle of the solder pads (refer to Figure 2). Then one pin has to be soldered. Then the connector might need to be corrected and a second pin can be soldered. If the pin header inserts easily from the bottom, all other pins can be soldered.

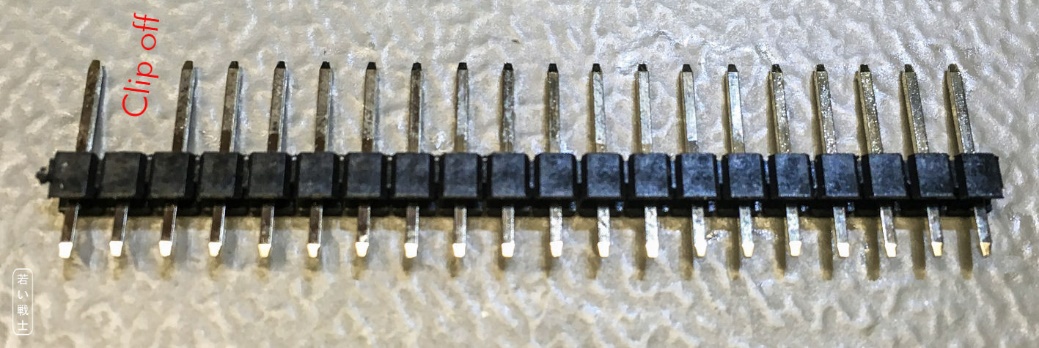


Figure 1: Preparation of J5 (clipping off the 2nd pin)

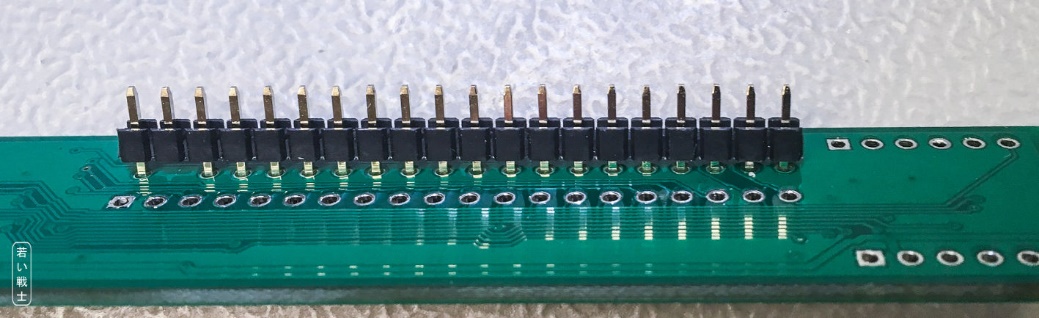


Figure 2: Centering of J1

Finally, the through-hole parts have to be soldered.

Close (or leave open) the solder bridge CP1, depending whether a short board (ASSY 250469, close CP1) or a long board (all other ASSY numbers, leave CP1 open) is present in the C64.

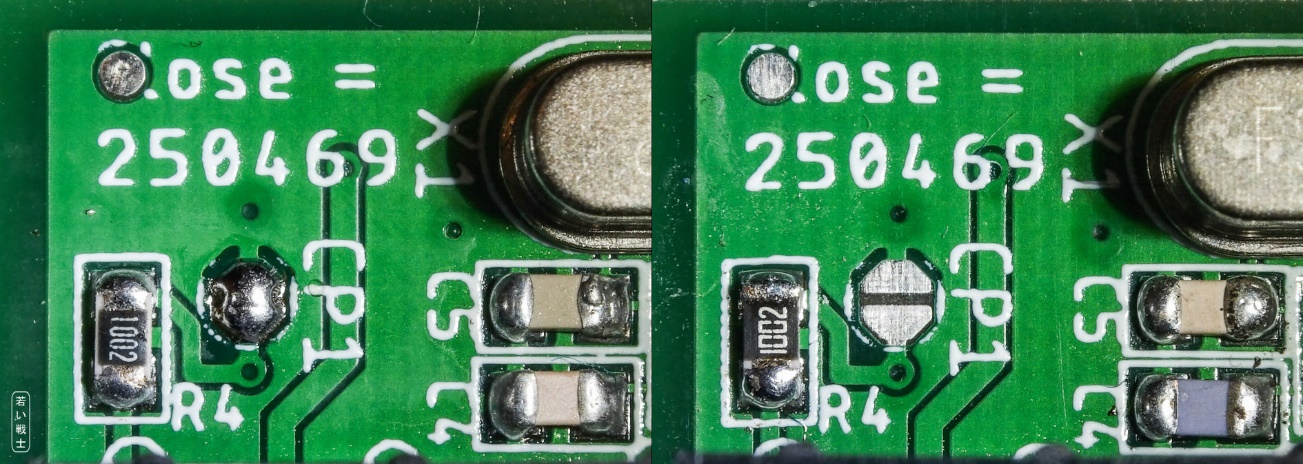


Figure 3: CP1 configured for short board (left) and long board (right)

To prevent the components of the Kernal Switch from making contact with any parts of the C64 mainboards, the area top and bottom of the keyboard connectors can be taped with duct or gaffer tape.

After being completely assembled, the micro controller IC1 (ATmega238P) has to be programmed. Refer to document number 128-6-05-00 (Programming Guide).

For wiring the Kernal Adaptor, the Kernal Switch and the C64, refer to document number 128-6-04-01 (Wiring).

The Assembly of the Kernal Adaptor might require de-soldering the Kernal ROM U4 and installing a socket.

# Troubleshooting

In case the C64 does not start properly, make sure that the Kernal Adaptor is inserted properly into the socket and the EPROM is properly programmed. At least the first Kernal has to be programmed. The Kernal Switch should sit firmly on the keyboard connector and does not make electrical contact with any component leads of the C64 mainboard.

The Atmega328P has to be programmed with either the production file or the \*.hex and the fuses have to be set properly (refer to document 128-6-05-01: Programming Guide).

Also, it is important to open or close the configuration solder bridge CP1 on the module to determine between a short and a long board. For C64 short boards, the solder bridge has to be closed, the BASIC has to be in the first 8k of the EPROM and a Kernal (original recommended) has to be in the 2nd 8k of the EPROM.

In case an EPROM smaller than a 27C512 is used, do not connect the unused address bits. On a 27C256, A15 has to be high/open, on a 27C128, A15 and A14 have to be open. A27C64 is not recommended. Refer to the documentation of the used Kernal Adapter.

In case the Kernals do not switch in the sequence as they were programmed into the EPROM, most likely, the wiring between the Kernal Switch and the Kernal Adapter is wrong (refer to document 128-6-04-01: Wiring) or the contacts are not clean or bad in some way. Clean the contacts and reseat the connectors.

Check, if the RESET pin is really connected to RESET of the C64. Again, check the cable to the RESET pin.

In case it is not working at all, remove the cable between the Kernal Switch and the Kernal Adaptor and set jumpers. Set all three jumpers (A13, A14, A15) for a long board and the jumpers A14 and A15 for a short board. In case it is working now, the Kernal Switch is selecting the wrong Kernal (maybe an empty 8k slot). Again, check the cable, measuring the pins A13, A14 and A15 on the J4 with a multimeter. Keep in mind that the first Kernal for a short board is 001BIN and for a long board it is 000BIN. A 0 here is represented by a low voltage (<0.7V) and a 1 is represented by a HIGH voltage (>4V).

If the problem is not solved, checking the solder joints is required. To see, if the software is running on the micro controller, a Serial Cable and a USB/Serial Adaptor plus the Arduino software (Tools → Serial Monitor, set to 9600 baud) might help.

After power up, the software should send messages via the serial cable. The RESTORE key has to be detected and reported (after holding it down) and a number key will be reported too.

Make sure, that the C64 power supply and the PC are connected and that the cables do not form a big loop (ground loop), as this might prevent proper function. The serial connection is only for programming (in some cases) and for debugging, so it is not a problem if a ground loop causes problems. It was experienced during testing, that with a big ground loop (a couple of square meters) the C64 did not always start properly.

# Keyboard Monitoring

The columns 1-4 and 7 as well as the rows 0 and 3 are connected to GPIO-pins of the micro controller. The C64 sends a scan bit to the columns, which are HIGH on default. The scan bit is LOW and it is shifted through the columns, one at a time. The rows are read back by the C64. A LOW signal on one or more row bits indicates a pressed key.

The following keyboard scan information can be obtained by the microcontroller:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Column** | | | | |
| 7 | 4 | 3 | 2 | 1 |
| **Row 0** | “1” | “9” | “7” | “5” | “3” |
| **Row 3** | “2” | “0” | “8” | “6” | “4” |

It is obvious that only the numerals are being monitored. Further the RESTORE key is connected to the microcontroller.

# Programming Model

| **Pin** | **GPIO** | **Arduino** | **Function** | **Configuration** |  |
| --- | --- | --- | --- | --- | --- |
| **30** | PD0 | D0 | RxD | Input | Serial Receive |
| **31** | PD1 | D1 | TxD | Output | Serial Transmit |
| **32** | PD2 | D2 | Row0 | Input, weak pull-up | Active LOW, INT0 |
| **1** | PD3 | D3 | Row3 | Input, weak pull-up | Active LOW, INT1 |
| **2** | PD4 | D4 | KSW\_A13 | Output | Kernal Switch A13 |
| **9** | PD5 | D5 | KSW\_A14 | Output | Kernal Switch A14 |
| **10** | PD6 | D6 | KSW\_A15 | Output | Kernal Switch A15 |
| **11** | PD7 | D7 | C64RES | Output | Active HIGH |
| **23** | PC0 | A0 | COL1 | Input, weak pull-up | Keyboard column 1 |
| **24** | PC1 | A1 | COL2 | Input, weak pull-up | Keyboard column 2 |
| **25** | PC2 | A2 | COL3 | Input, weak pull-up | Keyboard column 3 |
| **26** | PC3 | A3 | COL4 | Input, weak pull-up | Keyboard column 4 |
| **27** | PC4 | A4 | COL7 | Input, weak pull-up | Keyboard column 7 |
| **28** | PC5 | A5 |  | Input, weak pull-up | RESTORE-Key, active LOW |
| **12** | PB0 | D8 |  | Input | Short Board, active LOW |
| **13** | PB1 | D9 |  | output | C64 (a resistor R8 is in series with this signal) |
| **14** | PB2 | D10 | RESIO | TBD | Reserve IO Pin (a resistor R9 is in series with this signal) |

# Connectors

## J1 - Keyboard connector to mainboard

Bottom Entry receptacle (1x20p, pitch 2.54mm) – Harwin M20-7862042

| Pin | Signal | Pin | Signal |
| --- | --- | --- | --- |
| **1** | GND | **11** | ROW1 (PB1) |
| **2** | No pin | **12** | ROW0 (PB0) |
| **3** |  | **13** | COL0 (PA0) |
| **4** | +5V | **14** | COL6 (PA6) |
| **5** | ROW3 (PB3) | **15** | COL5 (PA5) |
| **6** | ROW6 (PB6) | **16** | COL4 (PA4) |
| **7** | ROW5 (PB5) | **17** | COL3 (PA3) |
| **8** | ROW4 (PB4) | **18** | COL2 (PA2) |
| **9** | ROW7 (PB7) | **19** | COL1 (PA1) |
| **10** | ROW2 (PB2) | **20** | COL7 (PA7) |

## J2 – In System Programming

2x3 box pin header (2.54mm pitch). This connector is used for the initial in-system programming of the ATmega238p (IC1). 89

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Signal | Pin | Signal |
| **1** | MISO | **2** | +5V |
| **3** | SCK | **4** | MOSI |
| **5** | (IC1) | **6** | GND |

## J3 – Serial/Debug

1x5 pin header (2.54mm pitch). In Arduino mode (after programming the Arduino boot loader), a USB/serial converter can be used to transfer the compiled Arduino sketches and for debugging with the serial monitor.

|  |  |  |
| --- | --- | --- |
| Pin | Signal | USB/Serial converter |
| **1** | GND | GND |
| **2** | +5V | +5V |
| **3** | TxD (output) | RX |
| **4** | RxD (input) | TX |
| **5** | DTR (input) | DTR |

## J4 – IO-Connector

1x7 pin header (2.54mm pitch)

This connector outputs the control signals of this board. It is for being connected to the Kernal and/or keyboard adapter. It also provides one reserved IO-Pin for later use. The GND might be useful, in case the RESIO-Pin is used to drive an LED (the power LED)?

| Pin | Signal | Function |
| --- | --- | --- |
| **1** | KSW\_A13 | Kernal Switch/Adapter A13 |
| **2** | KSW\_A14 | Kernal Switch/Adapter A14 |
| **3** | KSW\_A15 | Kernal Switch/Adapter A15 |
| **4** |  | RESET signal for the C64 (to be connected to a RESET point on the mainboard) |
| **5** |  | signal for the C64 (to be connected to an EXROM point on the mainboard) |
| **6** | RESIO | Reserved IO |
| **7** | GND | GND pin for multi-purpose use |

## J5 - Keyboard connector

Pin header (1x20p, pitch 2.54mm) – Pin 2 clipped off (no pin)

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Signal | Pin | Signal |
| **1** | GND | **11** | ROW1 |
| **2** | No pin | **12** | ROW0 |
| **3** |  | **13** | COL0 |
| **4** | +5V | **14** | COL6 |
| **5** | ROW3 | **15** | COL5 |
| **6** | ROW6 | **16** | COL4 |
| **7** | ROW5 | **17** | COL3 |
| **8** | ROW4 | **18** | COL2 |
| **9** | ROW7 | **19** | COL1 |
| **10** | ROW2 | **20** | COL7 |

# Documents

|  |  |
| --- | --- |
| Document | Document Number |
| Disclaimer | - |
| Module Description (this document) | 128-6-01-\*\* |
| Schematic | 128-1-01-\*\* |
| PCB prints | 128-2-01-\*\* |
| Functional Description | 128-6-02-\*\* |
| Software Description | 128-6-03-\*\* |
| Wiring | 128-6-04-\*\* |
| Programming Guide | 128-6-05-\*\* |
| Testing | 128-6-06-\*\* |
| Bill of Material | 128-5-01-\*\*.\* |

\*\* depends on revision

# Revision History

## Rev. 0

* Prototype (fully functional)

## Rev. 0 → Rev. 1

* J4, pin 5 is now for (software v0.1 or later required)
* J4 has an additional GND Pin (useful in case RESIO is connected to an LED)
* Series resistor in and RESIO signal (R8, R9: 220Ω or whatever is required).
* Additional 1KΩ (R10) between GND and +5V for faster discharge