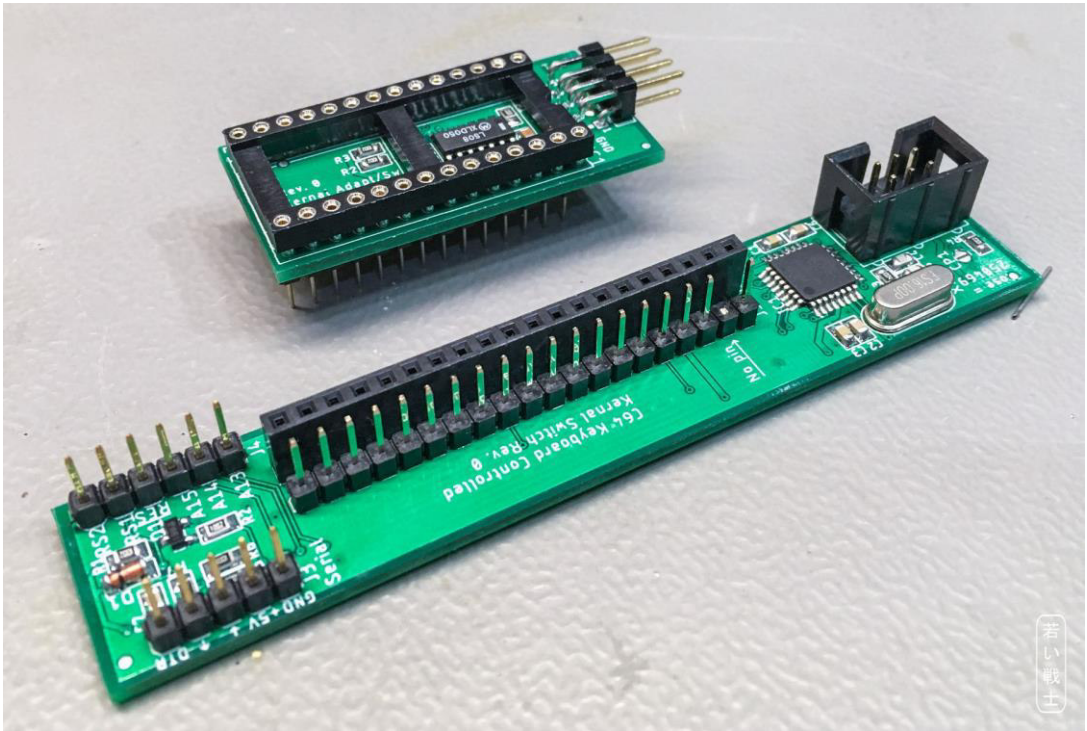
Project Documentation

Commodore C64 – Keyboard Controlled Kernal Switch

Project number: 128

Revision: 1

Date: 25.07.2019



Disclaimer

The usage of the described hardware and the information contained in the documents are at own risk. The author cannot be made liable for the correctness of the development and/or the documentation. It is assumed, that the assembly and installation of the hardware and software is performed by a person of sufficient expertise. The work has to be carried out with care, since it might risk damaging the hardware or the computer it is to be installed in.

C64 Keyboard Controlled Kernal Switch Rev. 1

Module Description

Table of Content

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 Boards* 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 $\overline{\text{RESET}}$ signal is provided to reset the C64 while/after selecting the Kernal via the three most significant EPROM address bits. Two other GPIO-pins ($\overline{\text{EXROM}}$ 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 $\overline{\text{EXROM}}$ -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.

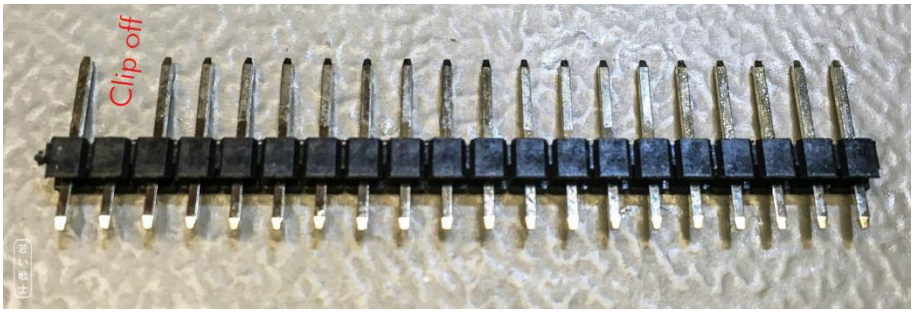


Figure 1: Preparation of J5 (clipping off the 2nd pin)

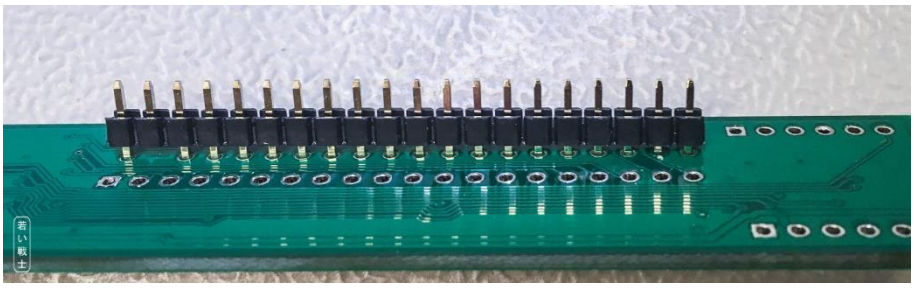


Figure 2: Centering of J1

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.

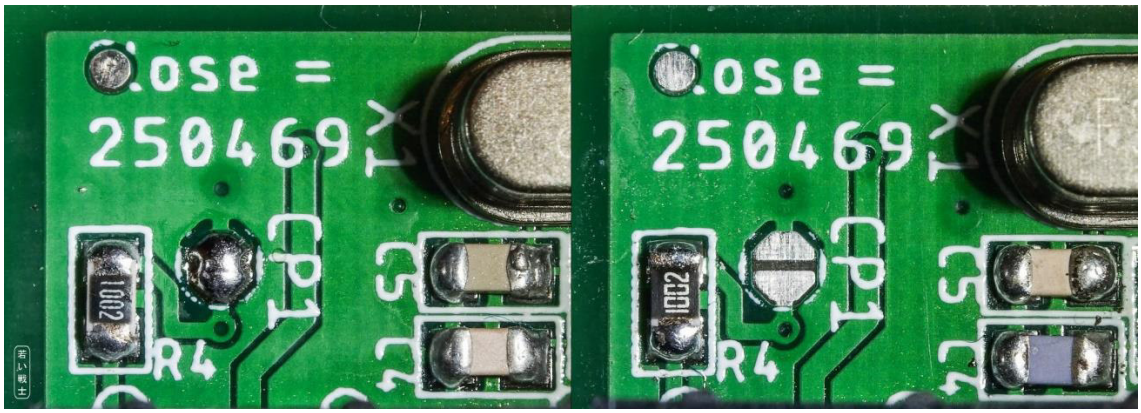


Figure 3: CP1 configured for short board (left) and long board (right)

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 001_{BIN} and for a long board it is 000_{BIN}. 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	$\overline{\text{RESTORE}}$	Input, weak pull-up	RESTORE-Key, active LOW
12	PB0	D8	$\overline{\text{SHORT\_BRD}}$	Input	Short Board, active LOW
13	PB1	D9	$\overline{\text{EXROM}}$	output	C64 $\overline{\text{EXROM}}$ (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	$\overline{\text{RESTORE}}$	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	$\overline{\text{RESET}}$ (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	$\overline{\text{C64\_RESET}}$	RESET signal for the C64 (to be connected to a RESET point on the mainboard)
5	$\overline{\text{EXROM}}$	$\overline{\text{EXROM}}$ 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	$\overline{\text{RESTORE}}$	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

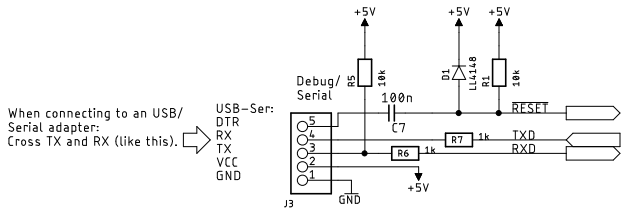
Revision History

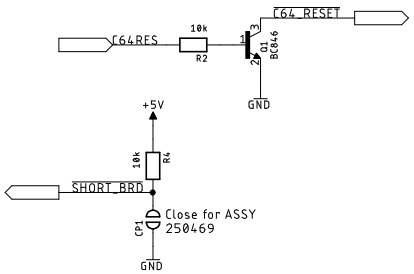
Rev. 0

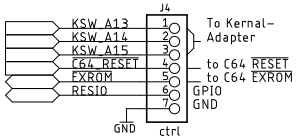
- Prototype (fully functional)

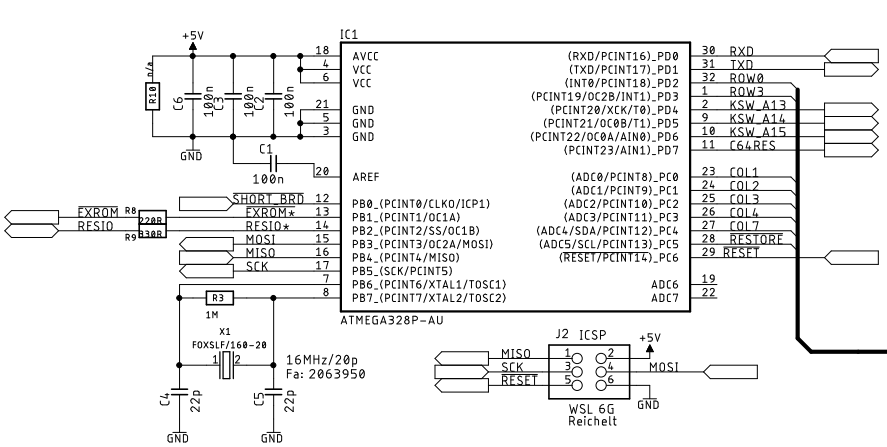
Rev. 0 → Rev. 1

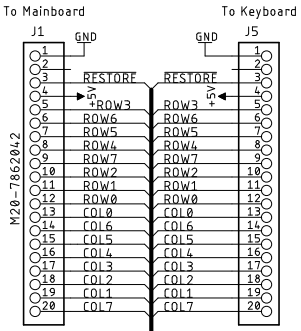
- J4, pin 5 is now for $\overline{\text{EXROM}}$ (software v0.1 or later required)
- J4 has an additional GND Pin (useful in case RESIO is connected to an LED)
- Series resistor in $\overline{\text{EXROM}}$ and RESIO signal (R8, R9: 220Ω or whatever is required).
- Additional 1KΩ (R10) between GND and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











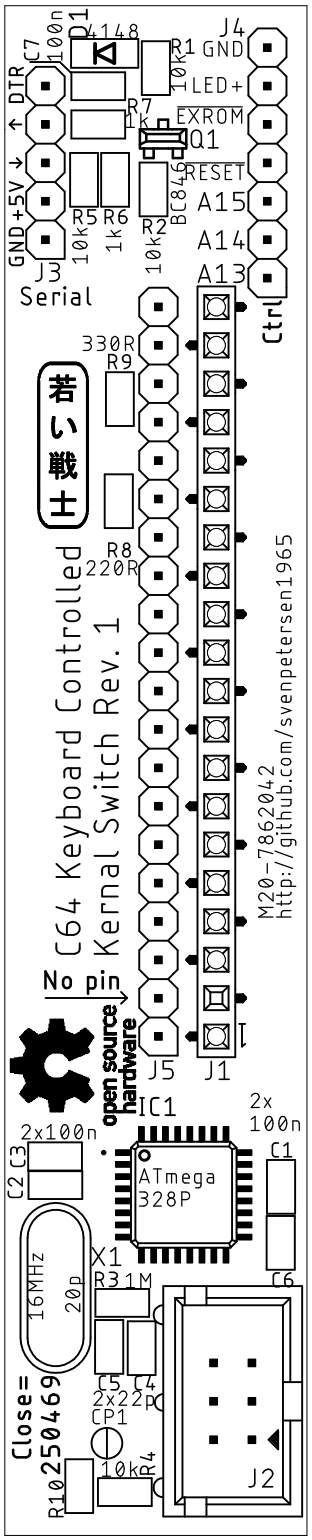
	Column				
	7	4	3	2	1
Row 0	"1"	"9"	"7"	"5"	"3"
Row 3	"2"	"0"	"8"	"6"	"4"

C64 applies LOW pulses to each column and reads back the rows.

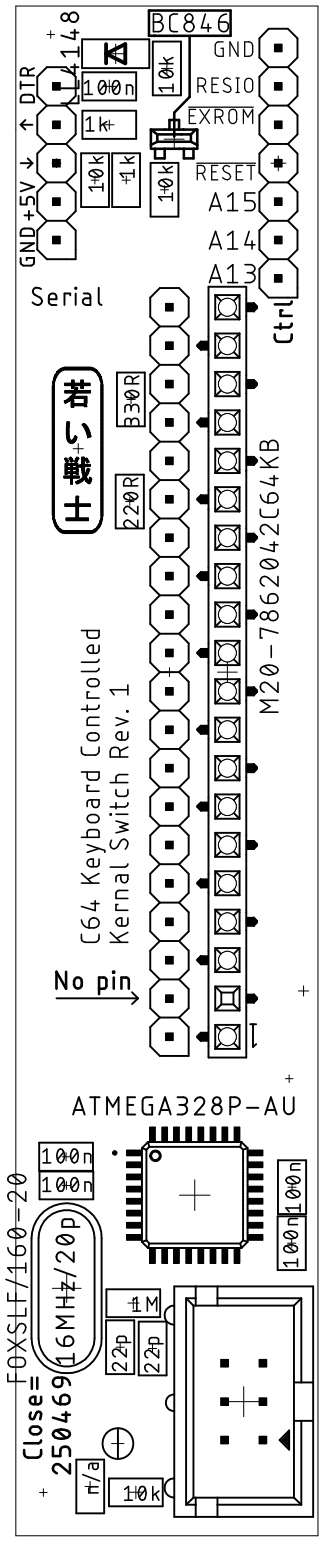
open source hardware

Title:	C64 Keyboard Controlled Kernal Switch	Doc.-No.:	128-1-01-01
		Draft.:	Sven Petersen
Date:	08.08.2020 19:06	Rev.: 1	Page 1/1
File:	C64_Kernal_Switch		
	https://github.com/svenpetersen1965		A3

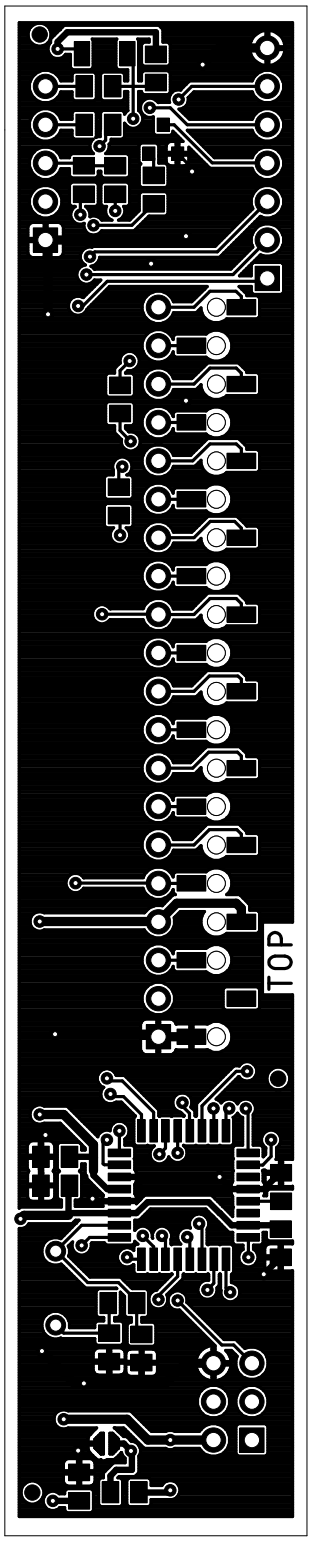
Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
08.08.2020 19:06		Rev.: 1
placement component side		



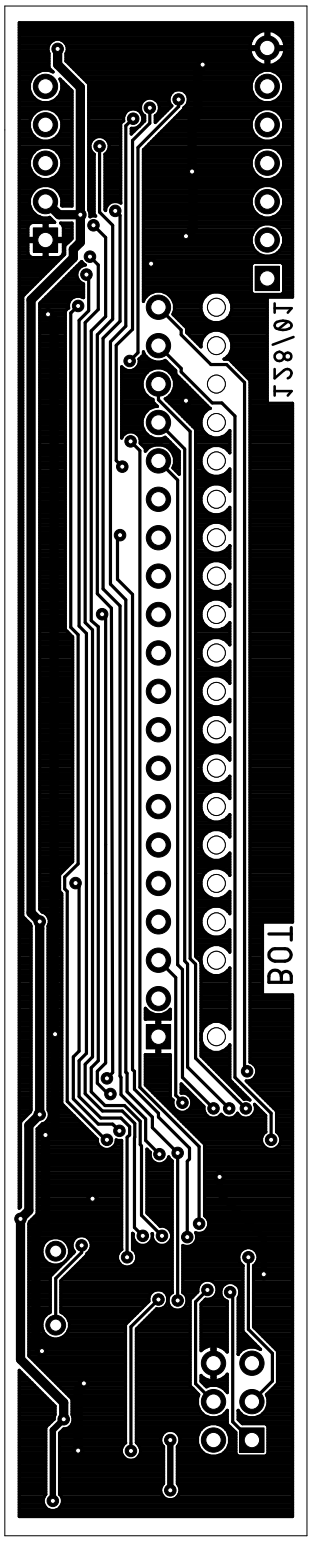
Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
07.01.2020 12:17		Rev.: 1
placement component side		



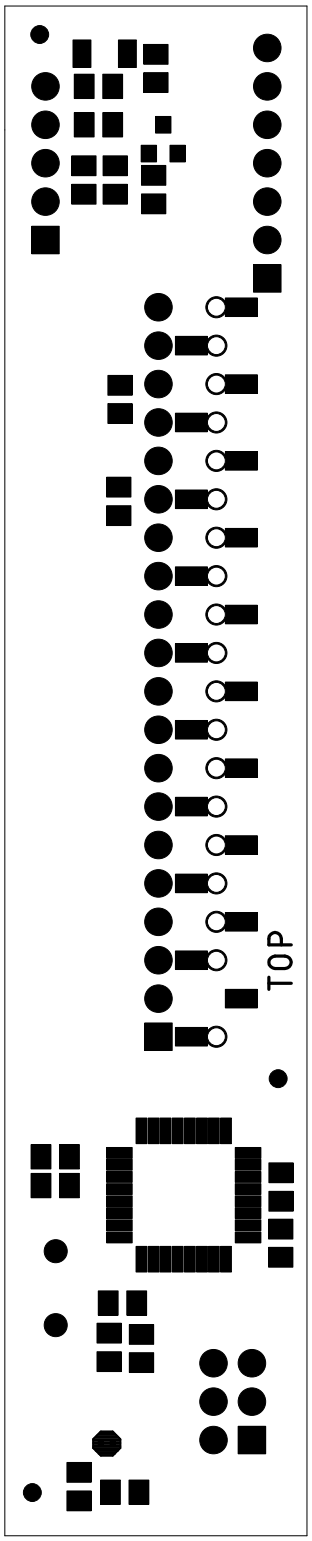
Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
07.01.2020 12:17		Rev.: 1
top		



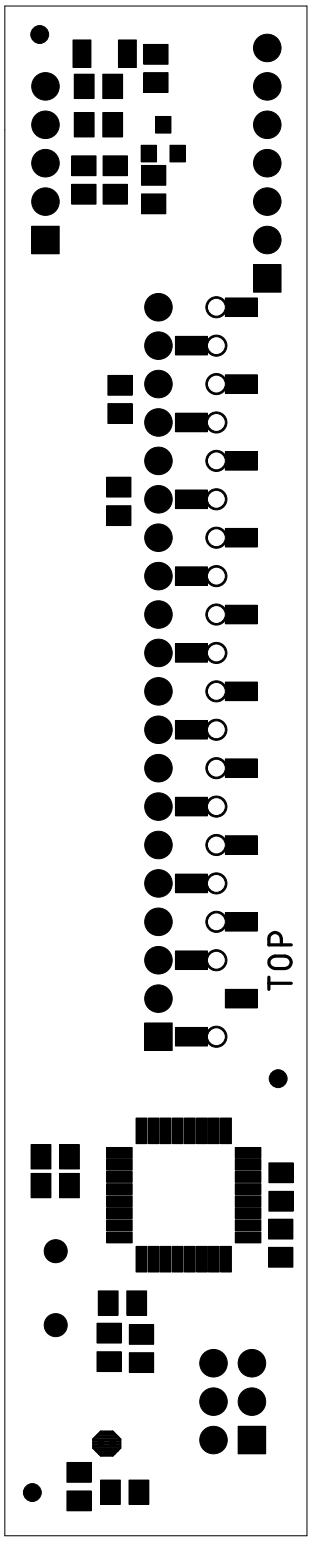
Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
07.01.2020 12:17		Rev.: 1
bottom		



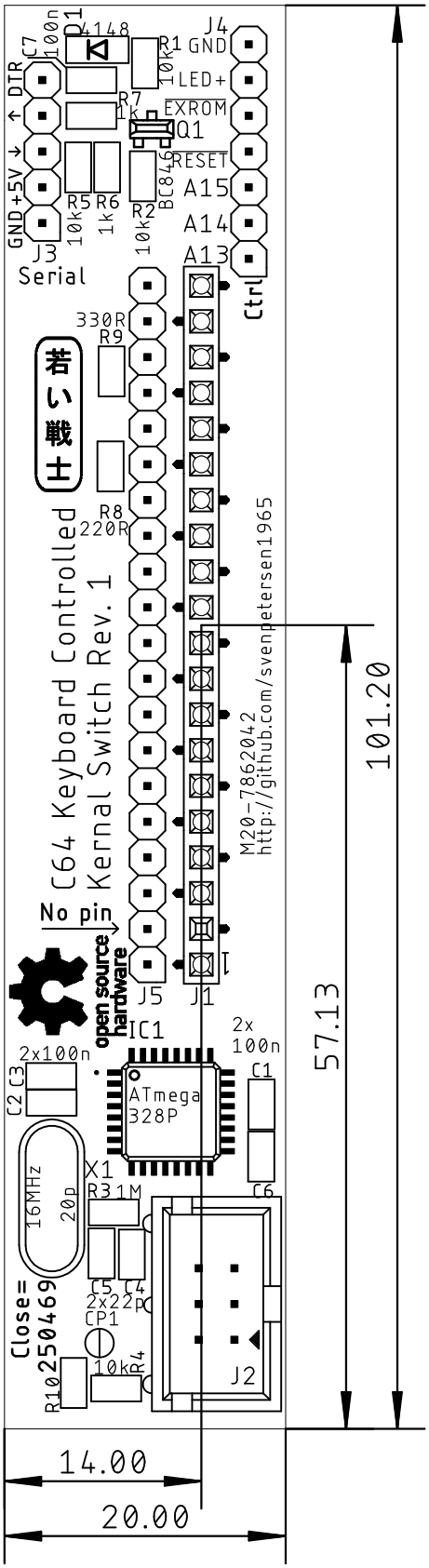
Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
15.01.2020 09:38		Rev.: 1
stopmask component side		



Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
15.01.2020 09:38		Rev.: 1
stopmask component side		



Sven Petersen 2019	Doc.-No.: 128-2-01-01	
	Cu: 35µm	Cu-Layers: 2
C64_Kernal_Switch		
08.08.2020 19:06		Rev.: 1
placement component side measures		



C64 Keyboard Controlled Kernal Switch Rev. 1

Functional Description

This document applies to hardware version v1 with firmware version v0.1 or later.

IC1 is an ATmega328P processor and it provides the functionality of an Arduino Uno. X1 is the 16MHz quartz and forms the quartz oscillator together with C4, C5 and R3. J2 is the connector for in-system programming of IC1. It is required at least once for writing the Arduino bootloader.

A serial interface for debugging is provided on J3. This interface has a TTL-level serial signals. To connect it to a PC, a USB/serial converter board is required. Once the bootloader is programmed to IC1, the serial interface can serve to transfer a compiled Arduino sketch to the Kernal Switch. In case this serial interface is not required and J3, C7, D1and R7 can stay unpopulated.

The Kernal Switch has to distinguish between C64 *long boards* and C64 *short boards*; this because with a *short board* BASIC resides in the lowest 8k of the EPROM.

The address pins A[15..13] (of the EPROM) ranges from 001_{BIN} to111_{BIN}. With a *long board* all 8k memory slots can serve as a Kernal so A[15..13] range from 000_{BIN} to 111_{BIN}.

- CP1 serves for setting the board type: (i) to configure a *long board* leave CP1 open; (ii) to configure a *short board* CP1 must be closed by a simple solder bridge.
- The transistor Q1 serves for resetting the C64 and provides the required open collector output. A HIGH level on C64RES will pull the output $\overline{\text{C64\_RESET}}$ LOW, which then resets the C64.
- J4 is the pin header, which provides the signals for selecting the Kernal (KSW_A13 .. KSW_A15), for resetting the C64 ($\overline{\text{C64\_RESET}}$), the $\overline{\text{EXROM}}$ signal and one GPIO pin (RESIO) which is utilized to drive a power LED (firmware v0.2). The GND pin (in conjunction with the RESIO) might be used for connection an LED.
- R8 will limit the current on $\overline{\text{EXROM}}$, in case a cartridge holds this signals HIGH. It is calculated to provide a stable LOW level voltage together with the 3.9kΩ pull-up resistor (network) on the C64 mainboard. A value of 300Ω might also work, but is not tested.
- R9 can be used as a bias resistor for an LED. 330Ω is a suitable value for a red or green LED.
- J1 is a bottom entry receptacle, the pins of the keyboard connector are inserted through holes in the PCB into this connector from below. All keyboard signals are connected here. J1 provides the supply voltage (+5V) for the circuit on the Kernal switch.
- J5 is the pin header for connecting the keyboard. All (keyboard) signals from J1 are connected to the same pin on J5.
- Only the required scan signals are connected to IC1. That is column 1, 2, 3, 4 and 7, as well as row 0 and 3. The row signals are pulled LOW in case a key in the row is being pressed. These row signals are connected to the two external interrupt pins (INT0 and INT1) of IC1. The key scan signals have to be configured as “input with weak pull-up resistor”, since the TTL logic of the C64 interprets an open input as a HIGH input, while it is an illegal state for the MOS logic of IC1.

C64 Keyboard Controlled Kernal Switch Rev. 1

Software Description

The software is programmed as an Arduino sketch. The purpose for this is to provide the user a way to modify the software for the personal requirements and to upload it via the serial port J3. This requires a cable and USB/serial converter, which will be described in another document.

The name of the Arduino sketch is *c64kbksw.ino*

The setup routine is initializing the pin modes, set some default logic levels, then reads the configuration jumper to determine, if a short board or a long board is detected. It reads the last selected Kernal number from the EEPROM of the microcontroller. If this is in range, the Kernal will be selected with the output bits KSW_A13 … KSW_K15. In case, the value provided by the EEPROM is out of range, the first possible 8k memory slot will be selected as a Kernal (depending on short board or long board).

The serial port is initialized to 9600, 8N1. The software issues some status information over the serial port (software version, short or long board, keyboard scan detection, number of Kernals, active Kernal number). This can be helpful when debugging the system. No data is received on the serial port.

The Reset timing is configured like this:

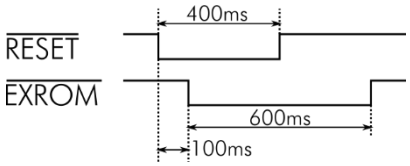


Figure 1: Reset timing with EXROM

The timing was provided by bwack and it works on all C64s with all known Kernals. The EXROM signal is only set low while an EXROM reset. For a normal reset, the EXROM signal stays HIGH. The timing is not very critical, an EXROM signal, that is too long will result in a wrong byte count of the free memory on the boot screen (not 38911 bytes free, but a lower number like 30719 bytes free). It just has to stay low for a little bit longer than RESET.

The C64 scans the keyboard by setting the column bits and reading back the row bits. Each of the 8 columns (the vertical structures of the keyboard matrix) is connected to one side of eight keyboard switches. Eight rows (the horizontal structures of the keyboard matrix). While figuring out, which key is pressed, the C64 sets the columns LOW, one by one. Then it reads back the row (as a byte, one bit per row). A bit value 0 (LOW) indicates the key, which is pressed. In case there is just one key pressed, all other row bits are 1 (HIGH).

There are also some other scan activities, which have to be filtered out. E.g. to determine, if a key is pressed (at all), all columns are set to LOW.

In case there is not proper Kernal in the selected memory slot, the C64 crashes (only a software thing, it will not harm the computer). A crashed C64 will not provide any keyboard scanning activities. The software can detect this state and the Kernal Switch will recover to the first Kernal. This is done by reading the column information. In case a value other than "HIGH" will occur in any of the bits before a timeout period elapses, the Kernal is considered to be working.

The two relevant row signals (ROW0 and ROW3) are connected to the two hardware interrupts. Pressing one of the number keys will generate an interrupt. So, there are two interrupt routines. Those will read the column data (the complete PORT C of the processor) and mask the relevant bits (0..4) if those are neither 11111_{BIN} nor 00000_{BIN}, a number key is detected and two flags are set and the column data is preserved, indicating the main loop, that something happened.

The main loop is first trying to find any keyboard activity. Once found, it will not be searched anymore.

Next, the state of the RESTORE key is read. In case the bit is LOW, the restore key is pressed. A count down is started. If the RESTORE key is held for more than 1.5 sec and then released, the C64 will be reset. The time mentioned before will be signalized, in case a power LED is connected to the Kernal Switch. The LED is normally on and will go off for a short time, after the time is reached. After releasing the RESTORE key, the reset signal is issued. The keyboard scan timeout is initialized again.

In case the RESTORE key is held longer than 3 seconds, an EXROM reset will be issued, independent a release of the RESTORE key.

In case the Interrupt Service Routines (ISR) attached to the row bits is informing the main loop, that a number key was pressed, it starts to decode the column information are determines the pressed key. The first key, that is found, will be assumed to be the choice of the user. Multiple keys are not processed.

In case the number is in range, the Kernal number is written to the EEPROM, then the Kernal is selected by setting KSW_A13 … KSW_A15 and resetting the C64. Finally, the main loop waits for 1000µSec. The loop time is not calibrated.

The initial keyboard scan detection might conflict with some cartridges, which do not provide keyboard scanning. One is the dead test cartridge. The Kernal switch can be "deactivated" for such a cartridge, either by pressing RESTORE 0 or by modifying the sketch. (the first the recommended way).

```
#define LED

//#define buzzer

[…]

/* Configuration */

const bool recover_empty = true;      //enable recover to Kernal one, in
case an empty/not working kernal was selected

const int  NumKernal = 8;             //This is the highest valid Kernal
number

[…]

#ifdef buzzer

const bool sig_idle = LOW;    // idle level of J4, pin 6 (should be HIGH
for a power LED)

#else

const bool sig_idle = HIGH;  // idle level of J4, pin 6 (should be HIGH
for a power LED)

#endif
```

For this purpose, the constant `recover_empty` has to be set to false and the constant `NumKernal` has to be set to the number of Kernals programmed.

For signaling, either an LED or a buzzer with a **built-in tone generator** can be used. In case the LED is used, it is assumed, that it replaces the power LED. So, it will be on after power up and switched off for signaling. In case a buzzer will be used, it is assumed, that this should be normally off and switched on for signaling. This is configuring the idle level of the signal (sig_idle).

C64 Keyboard Controlled Kernal Switch Rev. 1

Wiring v1.1

Connections

There is one required connection, the Control Signals Cable. The Serial Cable to the USB/Serial-converter is optional and only required for uploading an Arduino sketch to the Kernal Switch.

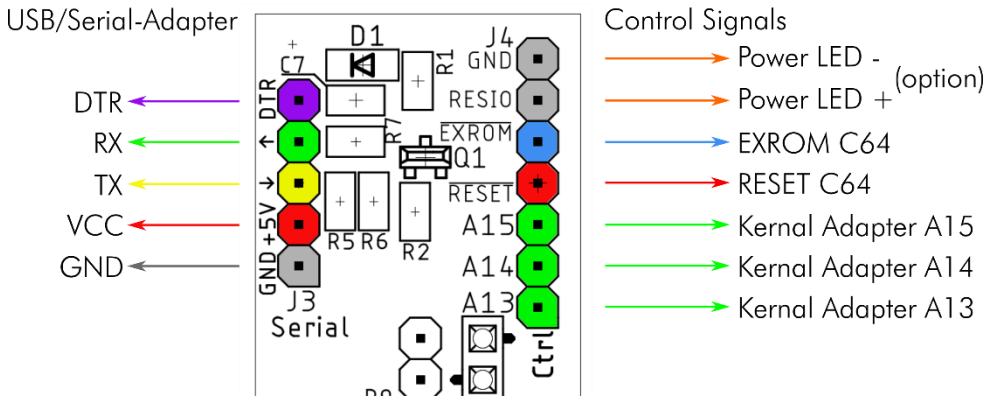


Figure 1: Connectors J3 and J4 of the Keyboard Controlled Kernal Switch

Control Signal Cable

A wire connection between the keyboard controlled Kernal switch and the Kernal adapter is required to select the desired Kernal. Further on, the module has to reset the C64 after changing to that Kernal.

A recommended connector type is the wide spread Dupont connector. The contacts need to be crimped to the cable—this might be a challenge for some users.

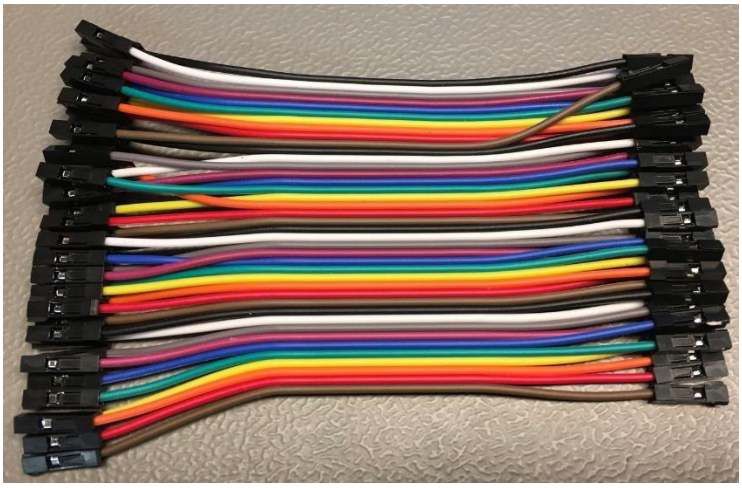


Figure 2: Prefabricated Dupont Cable

Alternatively, prefabricated Dupont cables (female–female) are readily available from eBay and are a suitable prefabricated solution. These prefabricated Dupont cables are usually sold as ribbon cables of up to 40 poles, each wire have an individual connector (Figure 2). The desired amount of wires can simply be peeled off of the ribbon cable. For the C64 *short board*, 30cm cables are required. For the C64 *long board*, 10cm cables are an adequate length. For the long board, the required cable for /EXROM is 30cm and for a short board it is 10cm long.

An alternative may be receptacles which fit on pin headers with the cables soldered to them. The next possible solution might be leaving the pin headers on Kernal switch and Kernal adapter not populated and solder in the cables directly. This is not really recommended.

A pin header (one pin) on the RESET signal of the C64 is required to be soldered. This RESET contact can be found near the user port. Looking from the port side, it is the 3rd contact from the left on the top side. Note, it is not good practice to solder the pin directly at the contact, but rather to a solder pad or via near it.

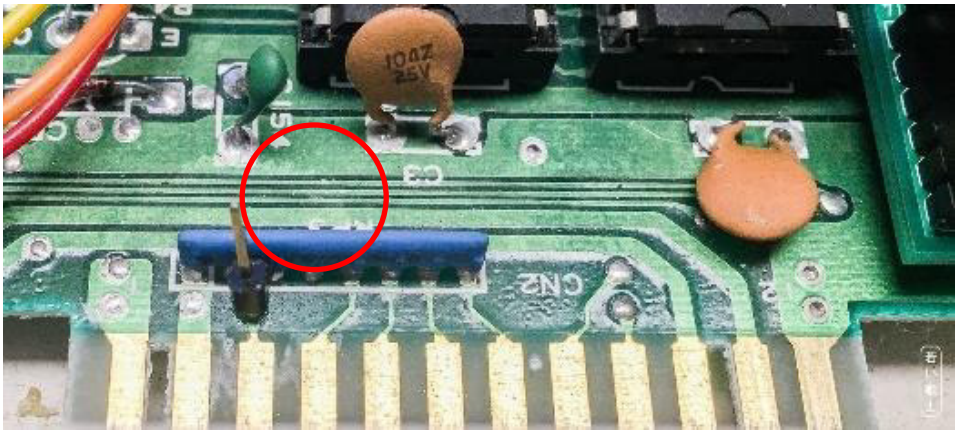


Figure 3: Reset Contact ASSY 250425 (identical to ASSY 250407) – Long Board

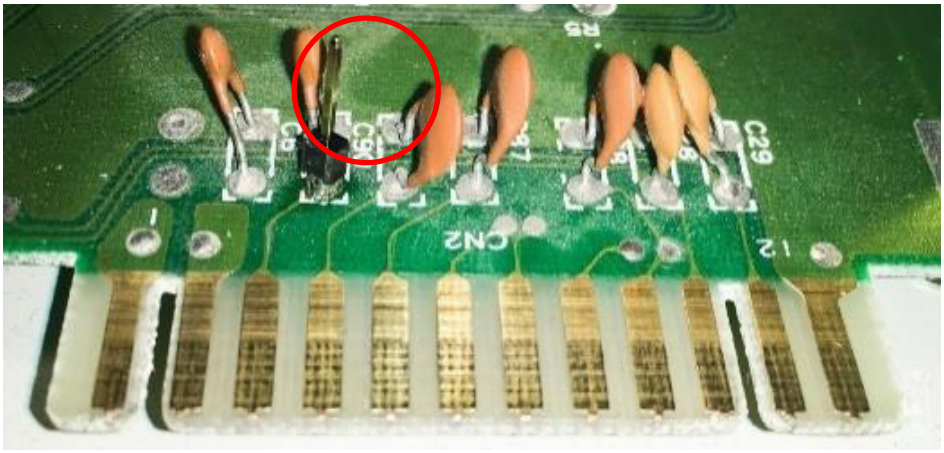


Figure 4: Reset Contact ASSY 250469 (front pin of C90) – Short Board

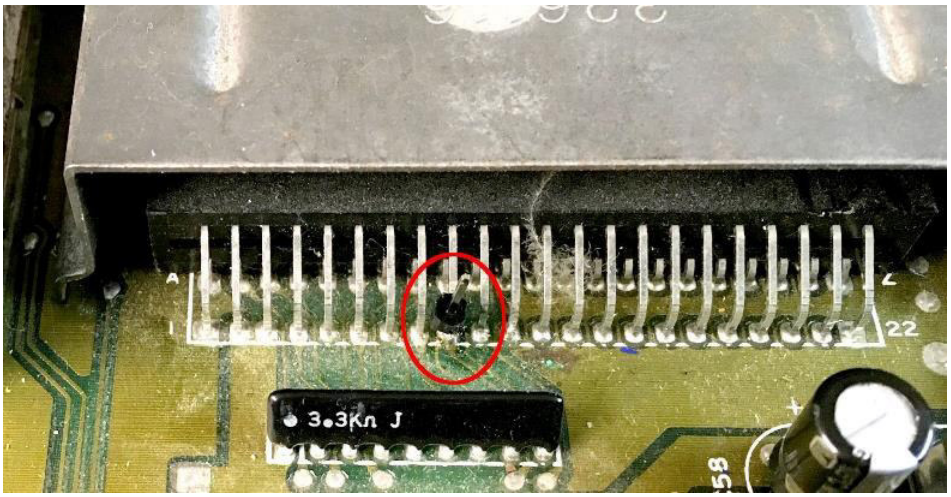


Figure 5: EXROM pin at the Expansion Port (Pin 9)

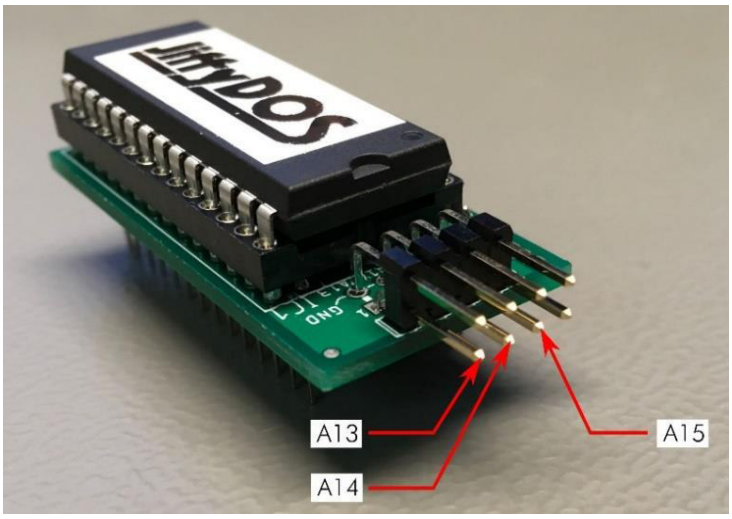


Figure 6: Connector of the Kernal Adapter

The required connections are:

Kernal Switch J4	Kernal Adaptor JP1	C64 Mainboard
A13	A13	-
A14	A14	-
A15	A15	-
$\overline{\text{RES}}$	-	RESET Pin (Figure 3 & Figure 4)
$\overline{\text{EXROM}}$	-	EXROM Pin (Figure 5)

Although the C64 *long boards* and C64 *short boards* require different Kernal adapters, both connectors are the same, as indicated in Figure 5.

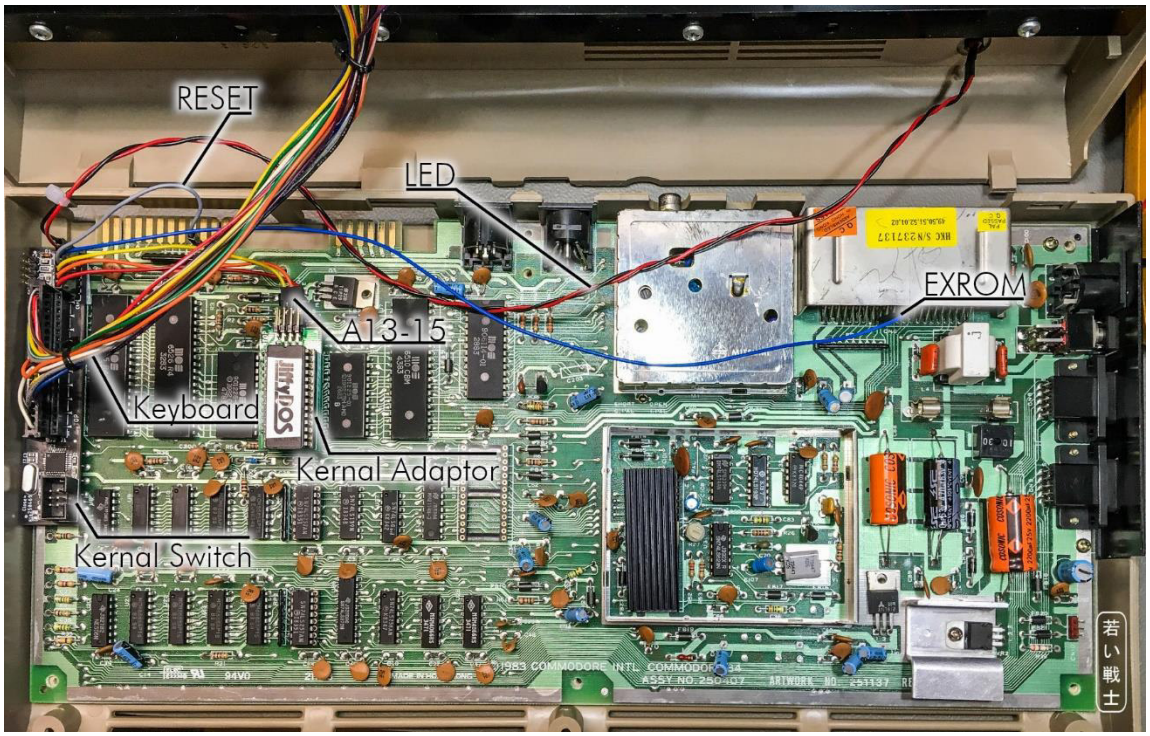


Figure 7: Wiring of ASSY 250407

Figure 7 shows the wiring of ASSY 250407 (ASSY 2502425 is similar). The grey cable is the reset signal, which is attached to the RESET pin header that was installed previously. The blue cable is EXROM.

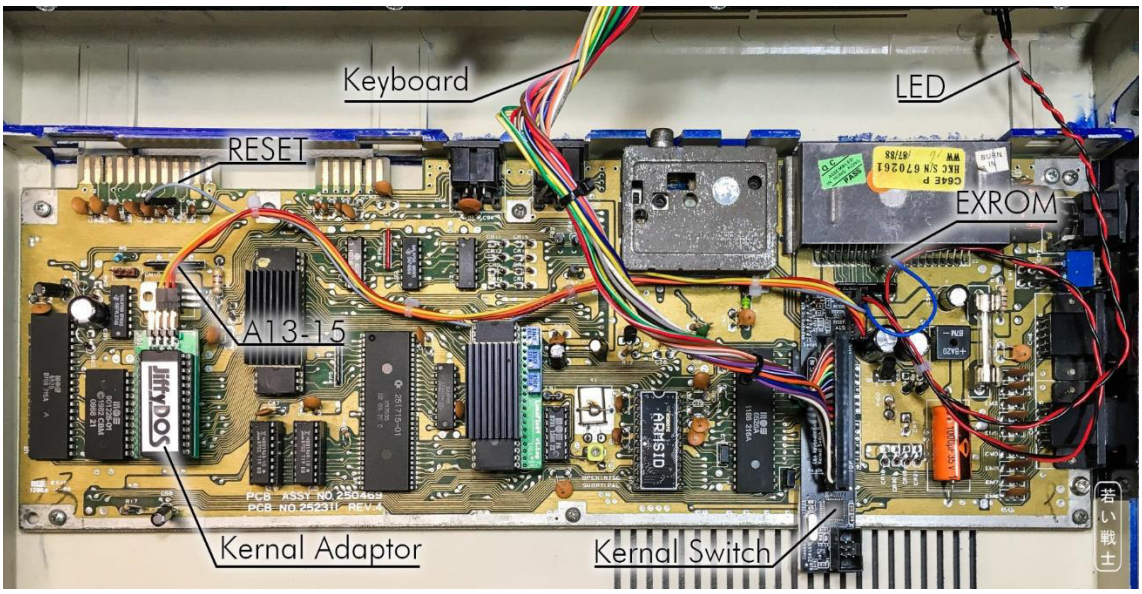


Figure 8: Wiring of ASSY 250469

Figure 8 shows the wiring of ASSY 250469 (*short board*). Again, the grey cable is the RESET signal, which is attached to the previously soldered RESET pin near the user port. The blue cable is EXROM.

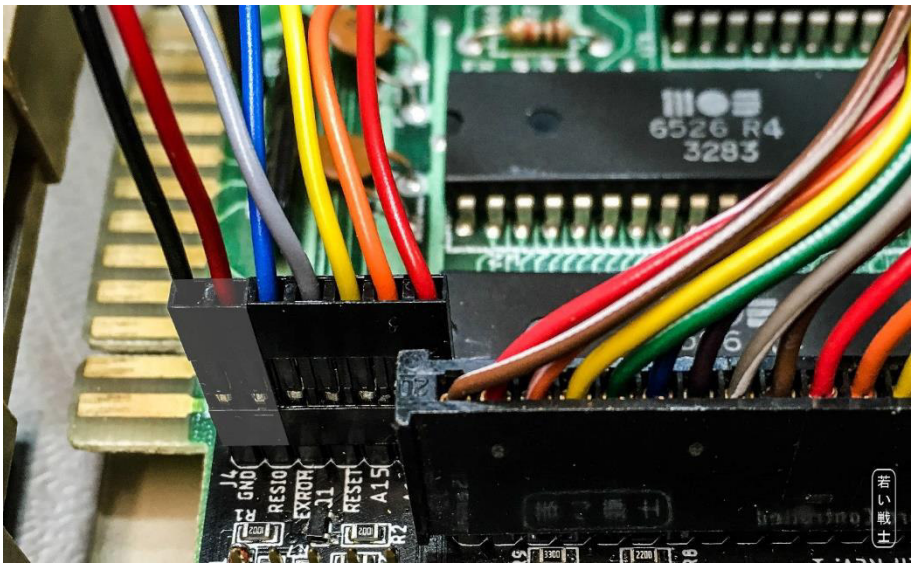


Figure 9: Wiring J4

For (the 7pin) J4, it is recommended to use a 2pin and a 5pin crimp housing/connector, in case the power LED is connected for signaling. This way, the top half of the C64 case can be easily separated from the bottom half with mainboard (disconnect LED and keyboard). Refer to Figure 9.

Serial Cable

The Serial Cable can vary, depending on the used USB/Serial adapter board. An FT232 based adapter is recommended, which is widely used, provides +3.3V and 5V operation and driver software up to Windows 10.

The USB/Serial adaptor provides VCC (+5V) to the module. This is useful while programming the microcontroller but is not recommended if the module is inserted in the C64 and the computer is powered by it. In this case, the C64 would be powered via the USB interface of the computer. This will not damage the C64, however, for proper operation, the VCC should be interrupted or switched off. The serial interface can be helpful while debugging.

Again, the recommended connector type are Dupont connectors with the alternative of using receptacles, which also fit on standard pin headers. Direct soldering is not recommended.

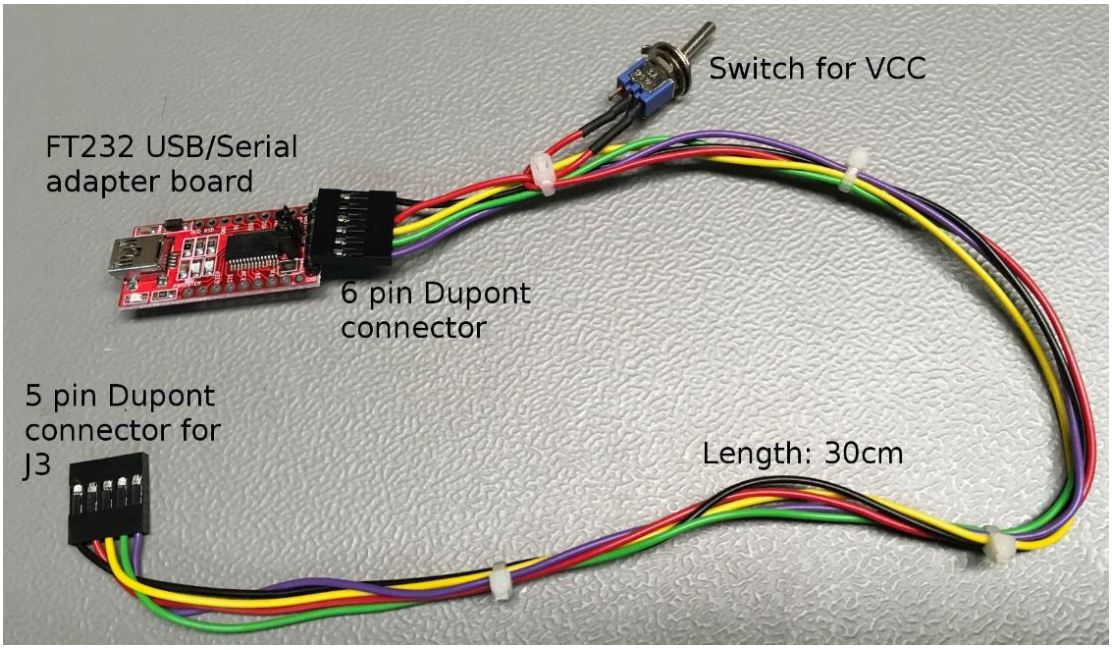


Figure 10: Serial Cable

Notice: During testing of the Kernal Switch, the C64 sometimes would not start. A ground loop was identified as the cause (mains → PC → USB → Adapter → Serial Cable → Kernal Switch → C64 → C64 Power Supply → mains). The power-on failure was resolved after interrupting the ground loop and the issue did not reoccur.

C64 Keyboard Controlled Kernal Switch Rev. 1

Programming Guide

Introduction

The microcontroller on the Kernal switch has to be programmed to function properly. There are several ways of transferring the program into IC1.

Since the program is based on the Arduino IDE, the program can be considered to consist of three parts:

- the Arduino Bootloader (optiboot_atmega328.hex)
- the fuses
- the compiled sketch (c64kbksw.ino)

The bootloader will enable the Arduino IDE to transfer (compiled) sketches via the serial cable into the ATmega328P microcontroller on the Kernal switch.

The fuses are the configuration of the ATmega328P. If those are not set properly, the Kernal switch will not work like desired.

The sketch is the source code of the program. It needs to be compiled and uploaded by the Arduino IDE (=*Integrated Development Environment*).

One Pass Programming

It is possible to transfer the bootloader, the compiled sketch and the fuses in one pass. That requires a suitable **programmer** (like the Atmel ICE, an SDK500 based programmer or another supported programming hardware). Further on, the **Atmel Studio** software is required. It is the IDE for writing software for Atmel controllers and programming the chips.

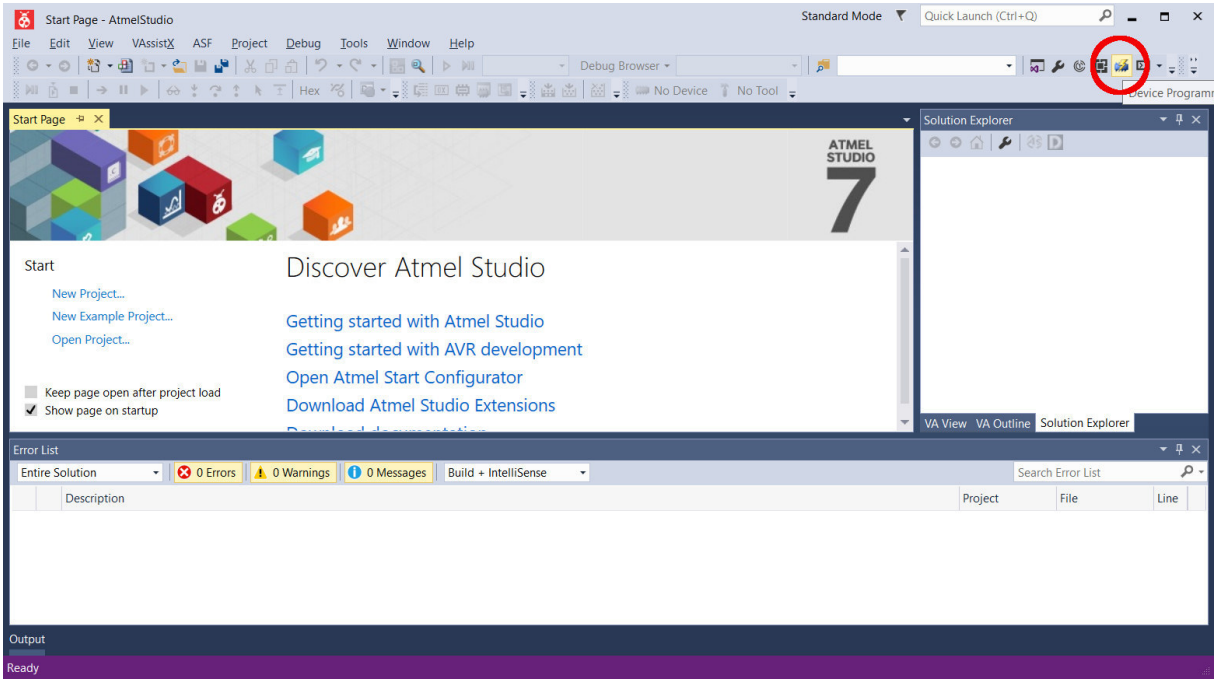


Figure 1: Atmel Studio IDE

The programming data is c64kbksw_v0_0.elf (or a later version). It was read out from an already programmed Kernal switch with a programmer and stored.

Step 1

Start the Atmel Studio IDE and select the Device Programming Tool as shown in Figure 1.

Step 2

Configure the tool (=the programmer), the device (ATmega328P), the interface (ISP = in-system programming) and click "Apply", like in Figure 2.

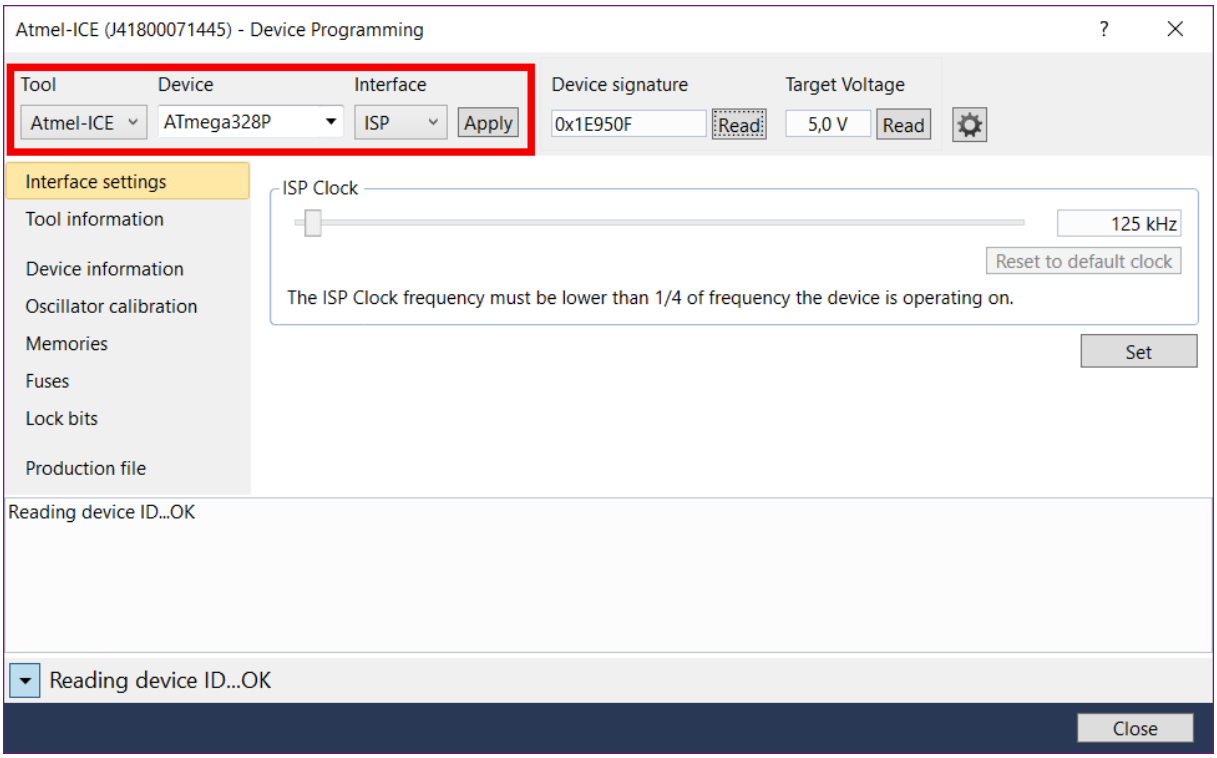


Figure 2: Device Programming

When you click "Read", the Device signature should be displayed and the Target Voltage should be shown (about 5V). In case, it does not, the module needs to be powered externally with 5V (connector J3). It can be done with the USB/Serial adaptor and the serial cable, or just a bench power supply.

Step 3

Click "Production File", then select the ELF production file (*c64kbksw_v0_0.elf* or later version) from the location, where you have stored it. Check "Flash" and "Fuses", also "Erase memory before programming" and "verify programmed content", like shown in Figure 3.

Now, click the "Program" button. The programming will start.

While the programming, the status is reported in the lower part of the window. This is marked yellow in Figure 4. Erasing, programming Flash and Verifying, Programming Fuses and Verifying should all be reported "OK".

Now, the program (bootloader and fuses included) is completely transferred to the ATmega328P. The module should be fully functional now. Since it contains the bootloader, the Kernal switch can be modified with the Arduino IDE and a serial cable.

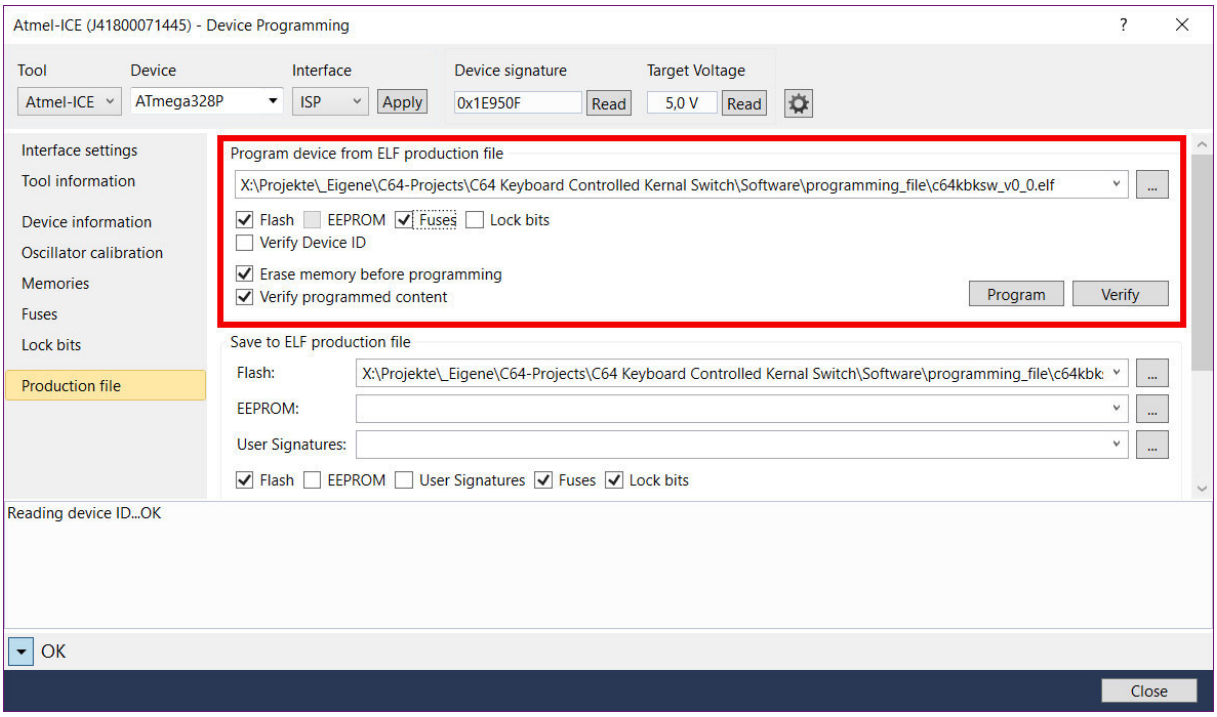


Figure 3: Production File settings

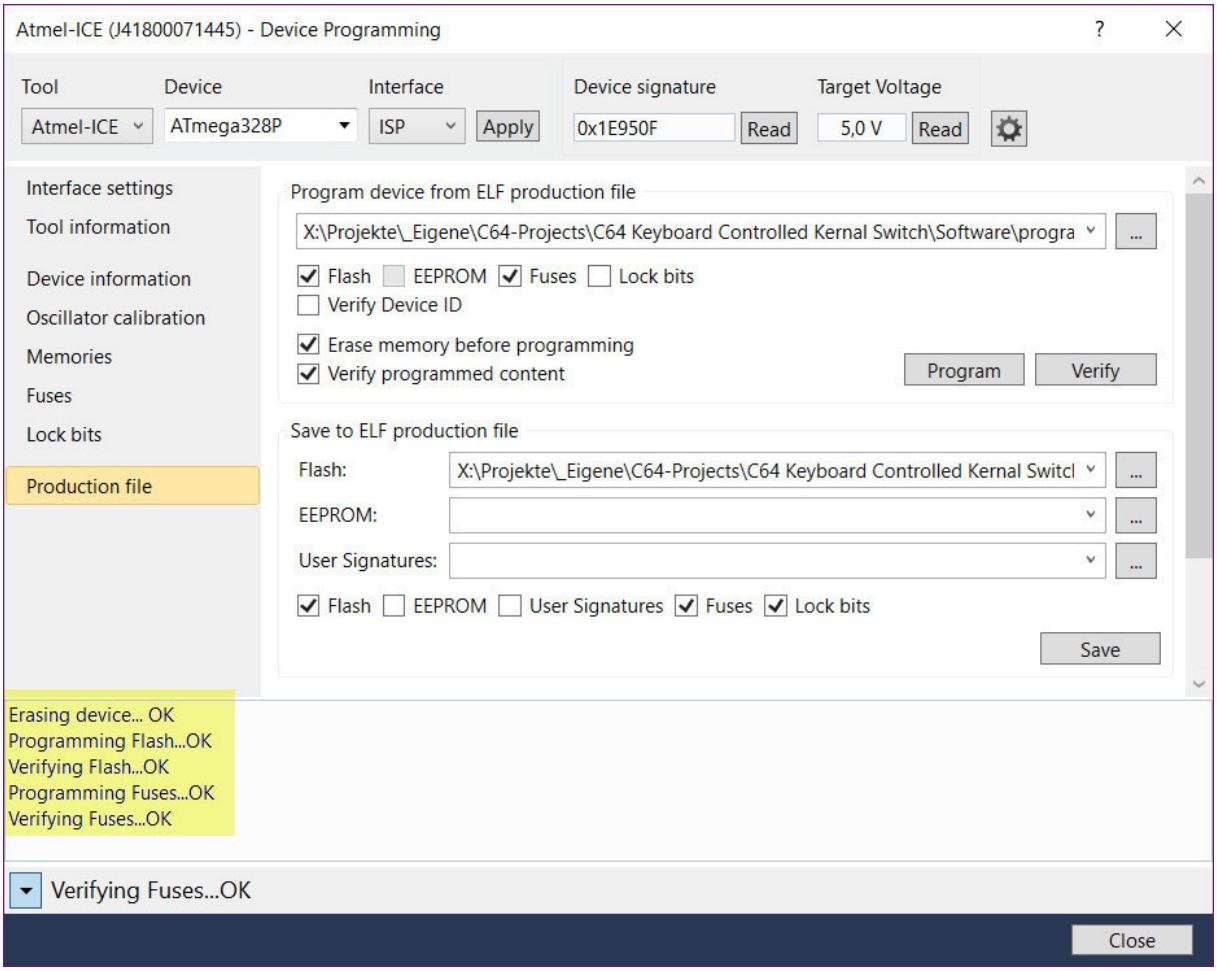


Figure 4: Programming the device

The fuses of the ATmega328P should be set like this:

- EXTENDED: 0xFD
- HIGH: 0xDE
- LOW: 0xFF

This can be checked with the Device Programming tool like in Figure 5.

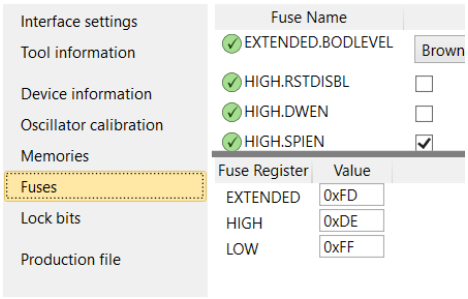


Figure 5: Fuses

One Pass Programming TL866

The ATmega328P can be programmed either with a TQFP32 adaptor or via the ICSP port. In both cases, the program (c64kbksw_v0_0.hex or a later version) has to loaded to the programming buffer (the format is Intel HEX) and the fuses have to be set manually (see Figure 6). The file contains the bootloader and the compiled sketch. No further programming is required.

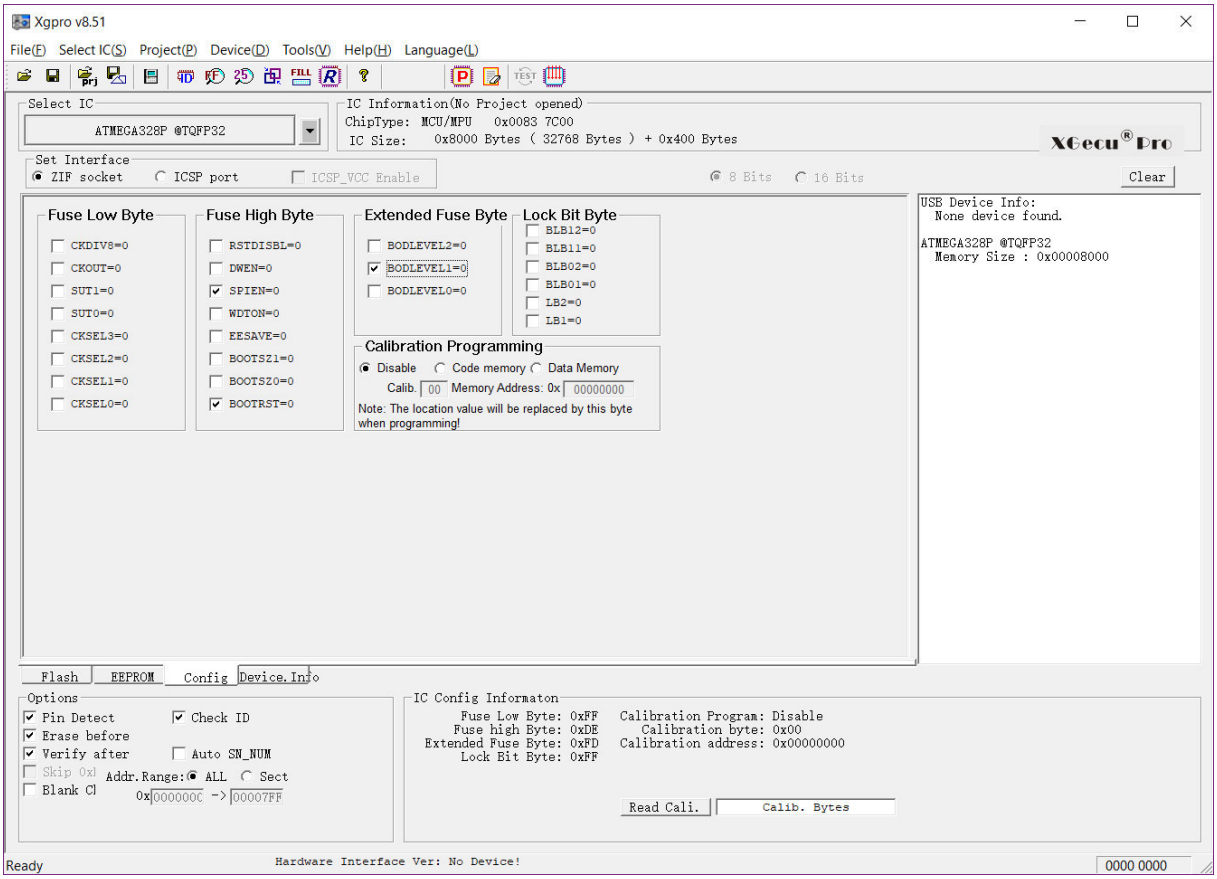


Figure 6: Correct setting of the fuses

This setting can be found in the "Config" tab.

Without a TQFP32 adaptor, the processor can be programmed in-system. The TL866II+ and the TL866A provide such an ICSP port. As a cable, a 6-wire Dupont cable will work (see Figure 9).

The connections should be made according to Figure 8.

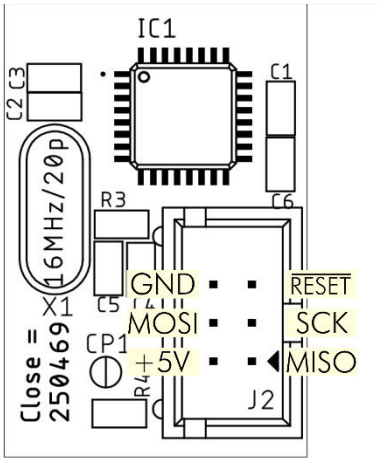


Figure 7: Pinout of the ICSP connector J2

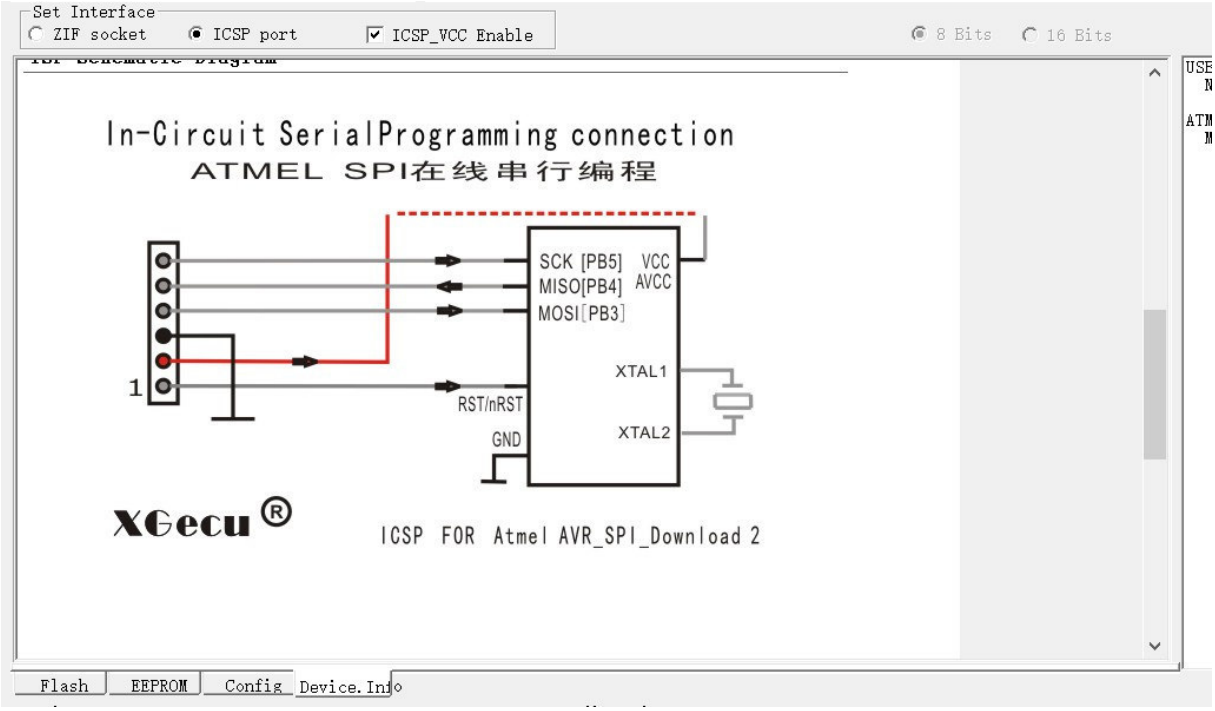


Figure 8: ICSP connection from TL866II+ to ATmega328P

The Dupont cable needs to be connected correctly, the connection has to be double checked to prevent harming the module or the programmer. In case, the repeated programming is intended, it might make sense to build a cable with a real 6x1 and 3x2 pin Dupont crimp housing (the 1pin crimp housings of the cable can be uninstalled).

When programming the controller, make sure, the code memory and the fuses are checked (Figure 10).

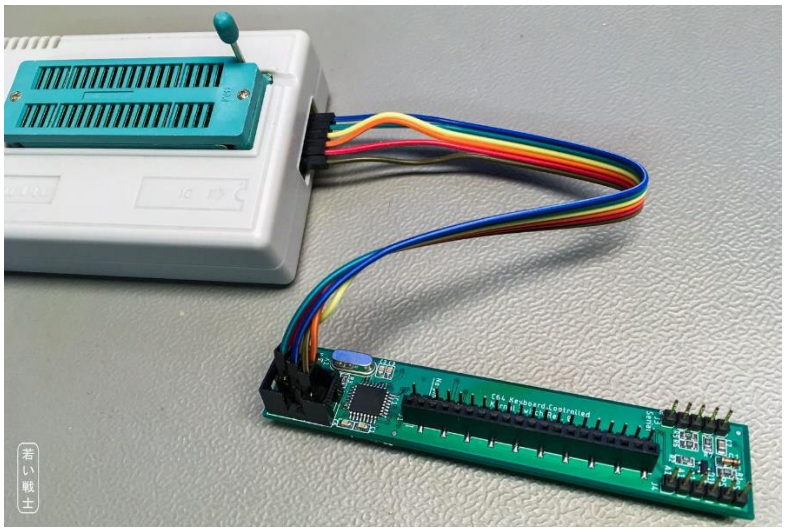


Figure 9: ICSP with a Dupont cable

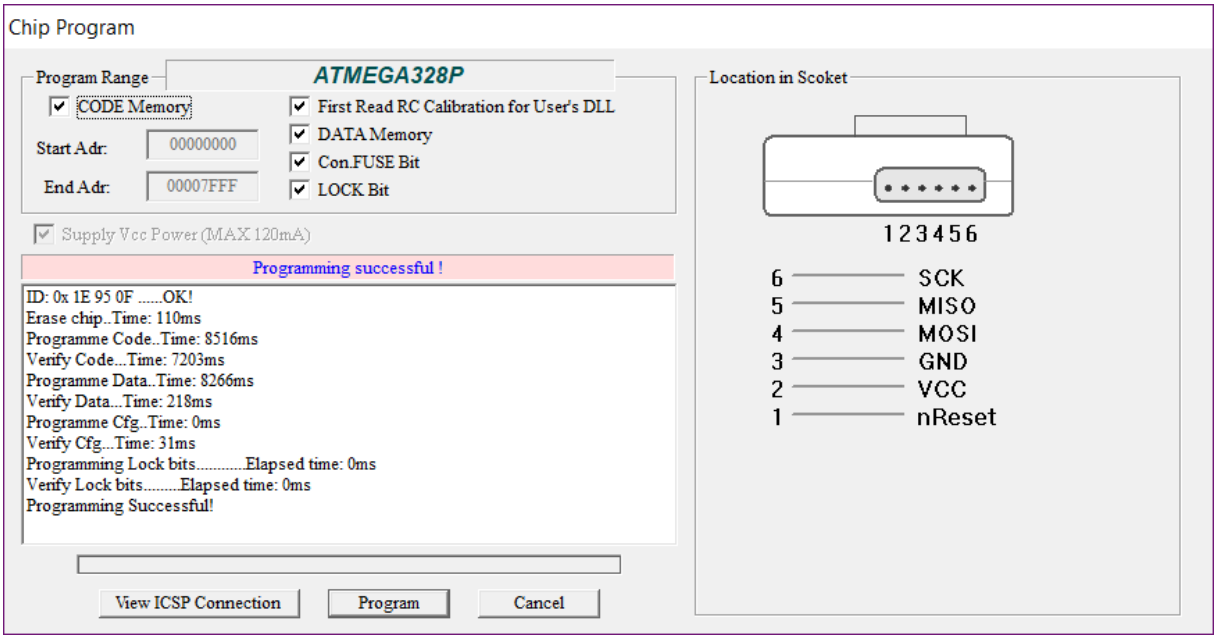


Figure 10: Programming the ATmega328P with the TL866II+

Programming the bootloader from the Arduino IDE

With a supported programmer or a programmer made from an Arduino Uno, the Arduino bootloader can be programmed into IC1. The fuses will be set correctly by the software.

The cheaper In-Circuit programmers for the Atmel processors usually mimic the STK500 development board. The functions described can be found in the Tools menu. Select the right programmer type, for STK500 you also have to set the right port (this is not the COM-Port provided by the USB/Serial adaptor). Once the Programmer and the port are set, connect the module to this programmer and click “Burn Bootloader” in the Tools menu of the Arduino software.

Instructions how to set up an Arduino Uno for programming the bootloader can be found by Google.

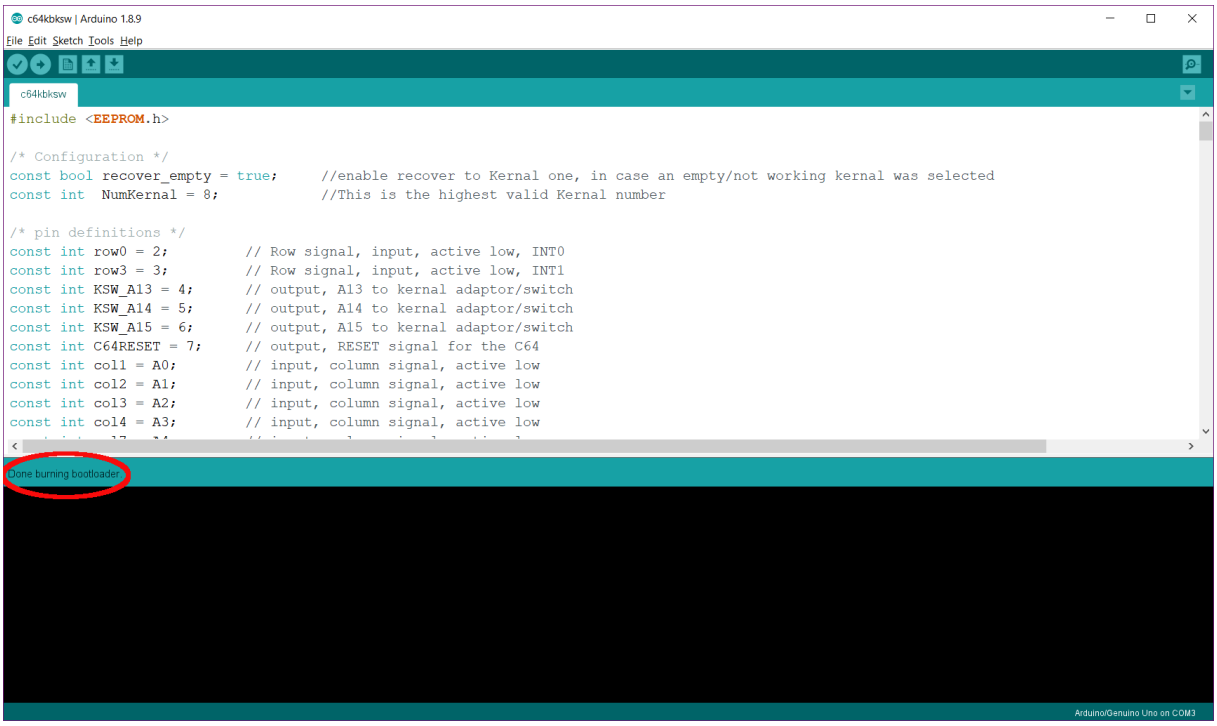


Figure 11: Burning the bootloader with the Arduino software

Note: the Atmel-ICE Programmer did not work properly for burning the bootloader from the Arduino software. If you own this programmer, you can program the bootloader with the Atmel Studio software.

Programming the bootloader from Atmel Studio

The Arduino IDE software contains the programming (*.hex) file for the optiboot bootloader, which is used for Arduino Uno etc.

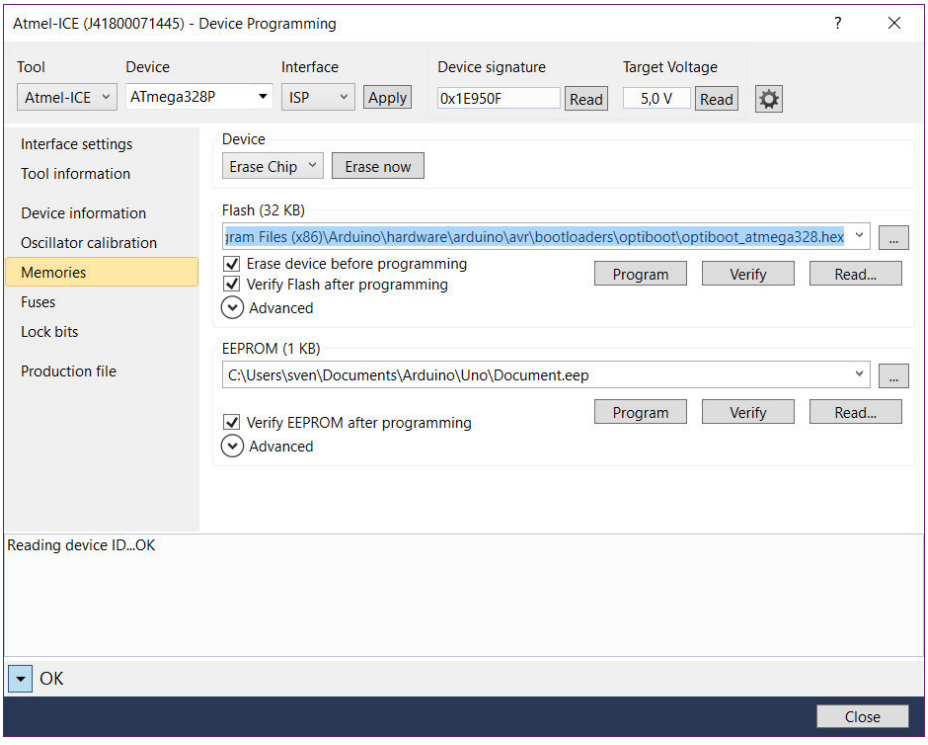


Figure 12: Programming the optiboot bootloader

It can be found here:

```
C:\Program Files (x86)\Arduino\hardware\arduino\avr\bootloaders\optiboot
```

The file name is:

```
optiboot_atmega328.hex
```

The hex-file can be loaded into the program buffer. Follow the steps like shown in Figure 1 and Figure 2, then select memories and for the Flash, select the file optiboot_atmega328.hex from the location described before.

The fuse settings are not contained in the *.hex file, they have to be programmed separately (see Figure 5).

Programming the bootloader with TL866II+

The bootloader that can be found in the Arduino software installation (see previous chapter) can also be programmed with the TL866II+ or TL866A. Please refer to Figure 6 to Figure 10. Remember, it is the file optiboot_atmega328.hex. The fuses are set exactly the same and need to be programmed.

Compiling and Uploading an Arduino sketch

The complete software for the Kernal Switch consists of two parts: the optiboot bootloader and the complied (and actually linked) Arduino sketch (c64kbksw.ino).

Once the bootloader is burnt, the sketch can be transferred via a serial cable as described in document number 128-6-04-00 (Wiring) and a required USB/Serial adaptor.

First the sketch has to be copied to the Arduino sketch folder (usually …\Documents\Arduino\c64kbksw). The sketch has to be opened (File → Open). After the Kernal switch is connected to the USB/serial converter (and this to the PC) via the serial cable and the power is on, the Board (in the Tools menu) has to be set to Arduino/Genuino Uno. Then the Port has to be selected. The USB/Serial adapter will set up as a COM-Port. If there are no other COM-Ports on the PC, it is the only one that will be listed in the (Tools → Port) list. In case you are not sure, press (windows key X) select the Device Manager and look it up in the COM & LPT section (top of the list). It should be something like "USB Serial Port (COMxx)" (xx is the number, you have to remember). In case there is more than one: disconnect the adapter and watch which port disappears and then connect it again and watch what appears.

Once you have found out and set the correct COM Port, the Kernal Switch is connected and powered, you can hit the "right arrow" (= upload icon, 2nd from the left). The sketch will be compiled and the uploaded. Alternatively, select "Upload" from the "Sketch" menu.

If it does not upload, first check the +5V from the USB-Serial adaptor. It can be measured between the +5V and the GND pin of the ICSP connector J2 (see Figure 7). Then, check the cable and finally check the soldering on the PCB (use a magnifying glass or microscope).

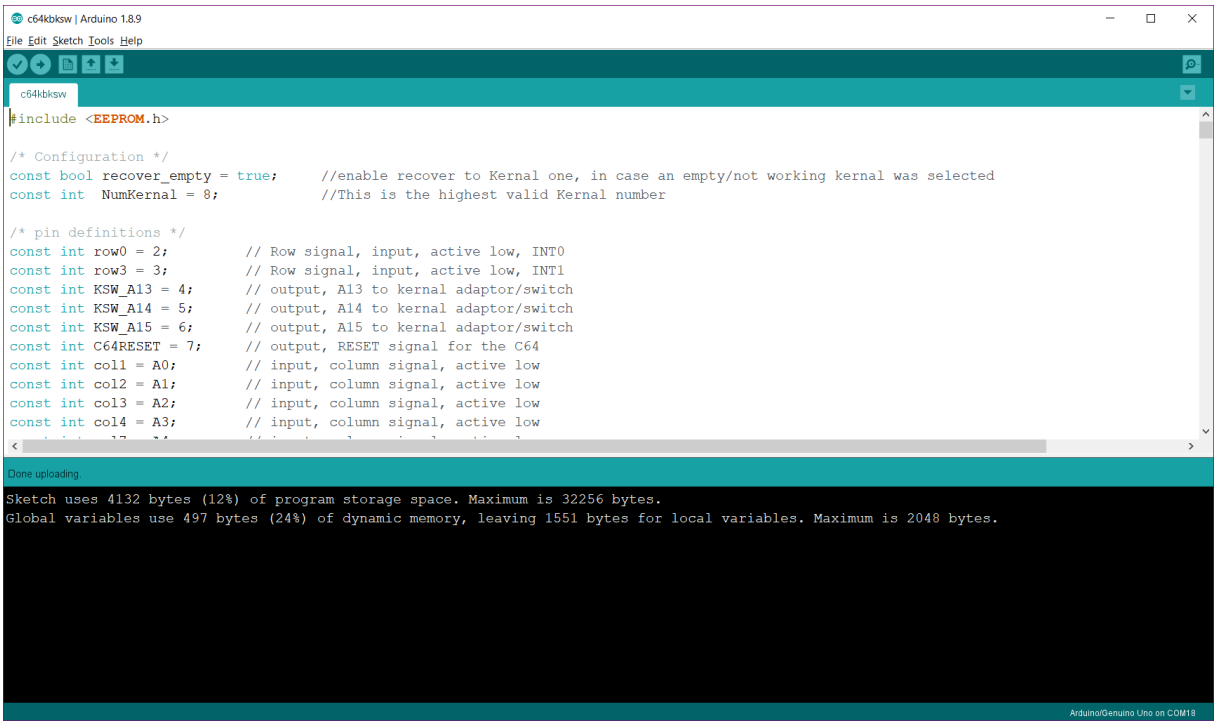


Figure 13: Compiling and uploading the sketch

C64 Keyboard Controlled Kernal Switch Rev. 1
Bill of Material Rev. 1.0

Pos.	Qty	Value	Footprint	Ref.-No.	Comment
1	1	128-2-01-01	2 Layer	PCB Rev. 1	2 layer, Cu 35µ, HASL, 101.2mm x 20.0mm, 1.6mm FR4
2		copper structure on pcb		CP1	no part, open Cutpad, close for ASSY250469
3	1	1x20	1X20	J5	pin header 1x20pin, 2.54mm pitch
4	1	1x7	1X07	J4	pin header 1x7pin, 2.54mm pitch
5	5	100n	0805	C1, C2, C3, C6, C7	SMD capacitor
6	4	10k	0805	R1, R2, R4, R5	SMD resistor, 5% or better
7	1	1M	0805	R3	SMD resistor, 5% or better
8	3	1k	0805	R6, R7, R10	SMD resistor, 5% or better
9	2	22p	0805	C4, C5	SMD capacitor
10	1	ATMEGA328P-AU	QFP80P900X900X120-32N	IC1	Atmel/Micro Chip, e.g. Farnell 2443176
11	1	BC846B	SOT23	Q1	SMD transistor
12	1	16 MHz/20pF quarz	HC49SLF	X1	e.g. Fox Electronics, FOXSLF/160-20, Farnell 2063950
13	1	LL4148	SOD80C	D1	SMD diode
14	1	M20-7862042	M20-7862042	J1	Harwin, e.g. Farnell 1256661
15	1	WSL 6G	2X03WV	J2	box pin header, 2x3pin, 2.54mm pitch
16	1	1x5	1X05	J3	pin header 1x5pin, 2.54mm pitch
17	2	220R	0805	R8, R9	SMD resistor, 5% or better

Rev. History

Pos.	**Rev. 0 -> Rev. 1**
1	PCB revision
4	Now 7 pin
8	R10 is new
17	new

C64 Keyboard Controlled Kernal Switch Rev. 1
Bill of Material Rev. 1.1

Pos.	Qty	Value	Footprint	Ref.-No.	Comment
1	1	128-2-01-01	2 Layer	PCB Rev. 1	2 layer, Cu 35µ, HASL, 101.2mm x 20.0mm, 1.6mm FR4
2		copper structure on pcb		CP1	no part, open Cutpad, close for ASSY250469
3	1	1x20	1X20	J5	pin header 1x20pin, 2.54mm pitch
4	1	1x7	1X07	J4	pin header 1x7pin, 2.54mm pitch
5	5	100n	0805	C1, C2, C3, C6, C7	SMD capacitor
6	4	10k	0805	R1, R2, R4, R5	SMD resistor, 5% or better
7	1	1M	0805	R3	SMD resistor, 5% or better
8	2	1k	0805	R6, R7	SMD resistor, 5% or better
9	2	22p	0805	C4, C5	SMD capacitor
10	1	ATMEGA328P-AU	QFP80P900X900X120-32N	IC1	Atmel/Micro Chip, e.g. Farnell 2443176
11	1	BC846B	SOT23	Q1	SMD transistor
12	1	16 MHz/20pF quarz	HC49SLF	X1	e.g. Fox Electronics, FOXSLF/160-20, Farnell 2063950
13	1	LL4148	SOD80C	D1	SMD diode
14	1	M20-7862042	M20-7862042	J1	Harwin, e.g. Farnell 1256661
15	1	WSL 6G	2X03WV	J2	box pin header, 2x3pin, 2.54mm pitch
16	1	1x5	1X05	J3	pin header 1x5pin, 2.54mm pitch
17	1	220R	0805	R8	SMD resistor, 5% or better
18	1	330R	0805	R9	SMD resistor, 5% or better
19	1	n/a	0805	R10	not placed

Rev. History

Pos.	**Rev. 0 -> Rev. 1.1**
1	PCB revision
4	Now 7 pin
17	new
18	new
19	new