**C64 Keyboard Controlled Kernal Switch Rev. 0**

**Software Description v0.1**

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 c64ksksw.ino

The setup routine is initializing the pin modes than 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 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 ready back the row (as a byte, one bit per row). A bit value 0 (LOW) indicates exactly 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 11111BIN nor 00000BIN, 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.

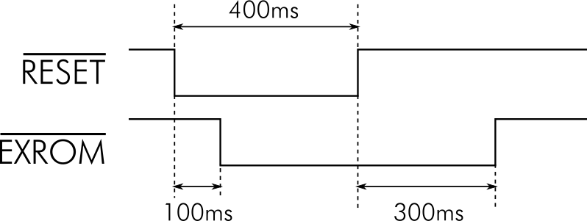


Figure 1:Timing of the (EXROM) reset

Next, the state of the RESTORE key is read. In case the bit is LOW, the restore key is pressed. A time count is started. If the RESTORE key is held between 1500 and 3000 main loop cycles (appr. 2-4 seconds), the C64 will be reset. If the RESTORE key is further held, an EXROM Reset is performed. The reset signal is issued for 400ms. The keyboard scan timeout is initialized again. The timing of the EXROM reset is not very critical, other timings have been tested and functioned as well.

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).

/\* Config uration \*/

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

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.