



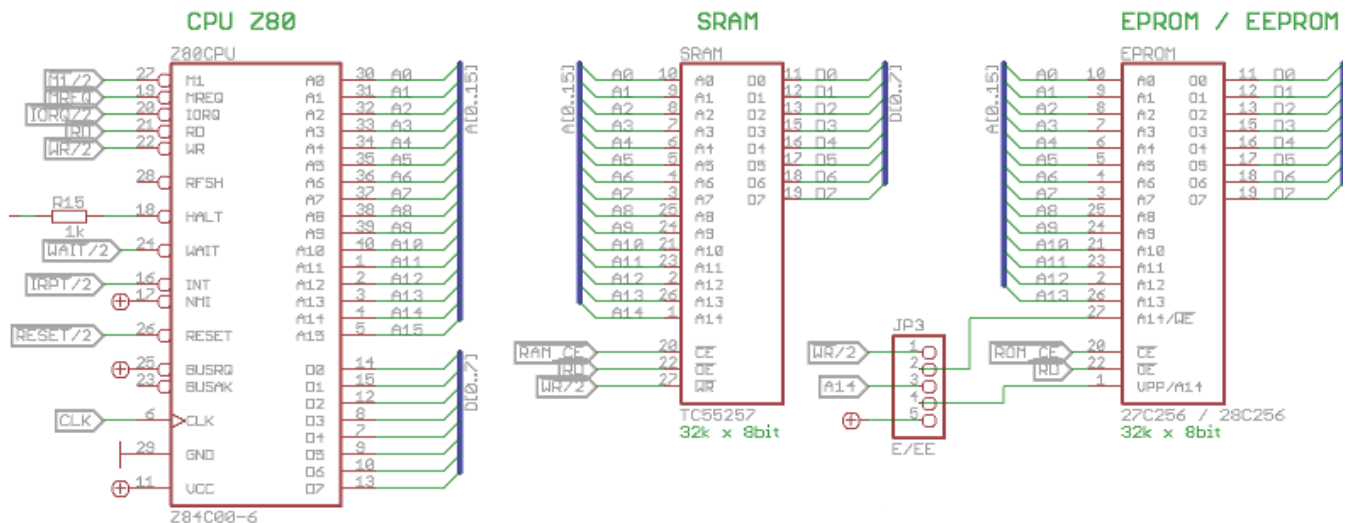
/pub/Develop/Hardware/K1-Bus Z80 CPU board with SRAM/

k1.spdns.de / [Develop](#) / [Hardware](#) / [K1-Bus Z80 CPU board with SRAM](#) /

Minimalistic Z80 System with SRAM and K1-Bus

The system consists of a CMOS Z80 CPU running at 6 MHz, one 32kB SRAM and one 32kB EPROM or EEPROM. It has no I/O except connection to a K1-bus. The K1-Bus allows attachment of 16 bit peripherals. K1-bus card selection is restricted to D0 .. D7.

Memory Map



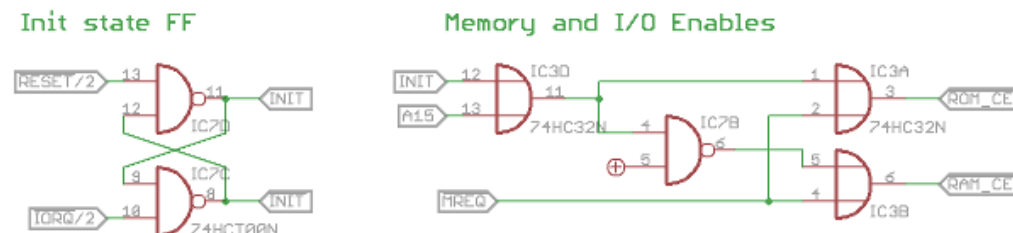
The Z80 uses separate instructions for I/O and memory access. Therefore the I/O address space is not part of the memory address space.

RAM is mapped to \$0000 .. \$7FFF and ROM is mapped to \$8000 .. \$FFFF. This allows modifying the RST vectors at run time.

After reset the ROM is mapped to the whole address space because the CPU starts execution at address \$0000 and needs to find code there.

Pin header JP3 allows using an EPROM or an EEPROM. The EEPROM is writable by the Z80.

RAM / ROM select circuit

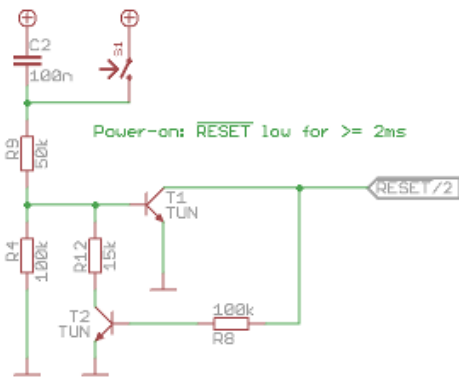


The \overline{RD} and \overline{WR} outputs of the CPU are directly connected to the RAM and ROM's corresponding inputs. Activation of the memory chips is done using their \overline{CE} input.

RAM is enabled when \overline{MREQ} is low and $\overline{A15}$ is low, ROM is enabled when \overline{MREQ} is low and $\overline{A15}$ is high.

There is a 'init state FF' which is set by \overline{RESET} and cleared by any I/O operation. Thus the FF is set after reset, will be cleared by the first I/O instruction and remain cleared throughout the rest of its life. While it is set it forces $\overline{A15}$ high in the RAM/ROM select circuit so that the CPU will always read from ROM.

Power-on Reset and Reset Switch



Reset circuit

The reset circuit was elaborated in great detail in my [hardware blog](#). It is a little bit load dependent but should work up to 5mA pull-up current, which is much more than is to be expected, unless you attach a LED here. ;-)

With the selected value of the timing capacitor the reset pulse will be approx. 2 ms.

I/O

The CPU board does not contain any peripherals, not even a timer interrupt. All Peripherals are expected to be connected at the K1-bus.

The K1-bus provides an I²C bus to attach EEPROMs with driver code and is 16 bit wide. Therefore two data latches are required for buffering data from the Z80's 8 bit bus to the upper half of the K1-bus. I/O therefore is divided into *I²C bus access*, *high-to-low* and *low-to-high* data latch access and *actual bus access*.

The K1-bus is specified for symmetrical signals (HC, AC not HCT or TTL) therefore some signals may not have required levels without help.

K1-bus **IRPT** is directly connected to the **IRPT** input of the CPU. The interrupt source is determined in software.

K1-bus **WAIT** is directly connected to the **WAIT** input of the CPU. Any peripheral board which can issue **WAIT** fast enough can be used with this CPU board. Since the Z80 is pretty slow this probably means all peripheral boards.

The K1-bus address lines **A0** to **A5** are directly connected to the corresponding CPU address lines. They are pulled up with 3.3kΩ resistors to help the CPU to pull them up: The CPU has very little driving capabilities and driving high is even less than driving low:

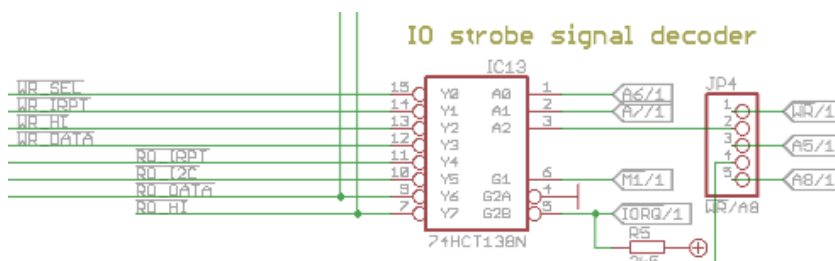
IOL = 2.0mA	@0.4V
IOH = -1.6mA	@2.4V (which is too low for HC inputs)
IOH = -250μA	@4.2V

Eventually 4k7Ω is a better choice. To be tested.

The K1-bus data lines **D0** to **D7** are directly connected to the CPU as well, which is an allowed design for very small systems. Expect problems with the third card added! They are pulled up with 3.3kΩ resistors to help the CPU to pull them up for the same reason as above. The K1-bus data lines **D8** to **D15** don't need pull-ups because they are driven by the 74HCT574 which has symmetrical outputs (though TTL inputs) and slightly higher driving capabilities as well.

K1-bus I/O

Any I/O instruction addresses the K1-bus. This is centered around a 74HCT138 3-to-8 decoder. It is enabled by **IORQ=0** && **M1=1**. It then activates one of 8 strobe lines, depending on **A6**, **A7** and **WR**. Pin headers JP4 allow to use **A6**, **A7** and **A8** instead and **IORQ** has a "strong" pull-up resistor. The reason for these circuit options are explained below.



The following strobe signals are generated by the 74HCT138:

WR_SEL

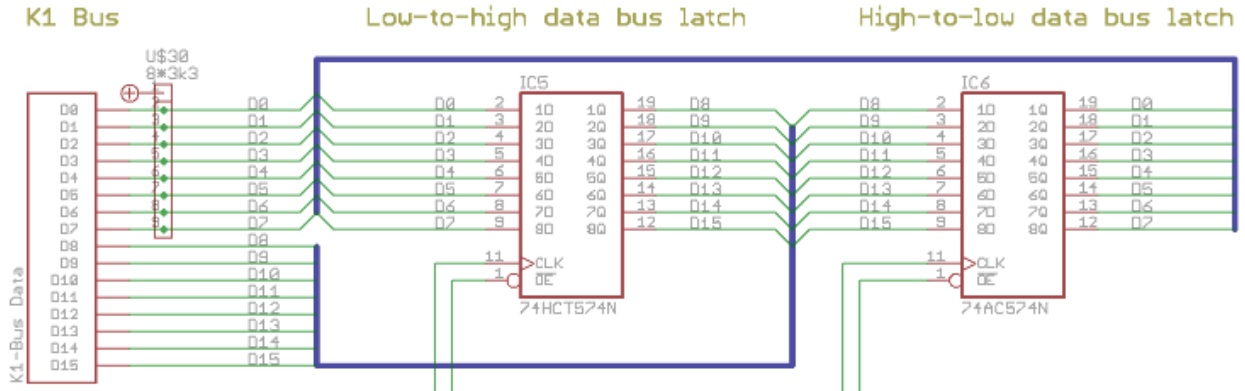
Signal on the K1-bus to select a card for subsequent I/O operations. I/O cycles on the K1-bus do not contain the address of the talked-to card, instead a card must be 'selected'.

RD_IRPT

Signal on the K1-bus to read in the interrupt states of all cards to detect which one actually activates the IRPT line.

WR_IRPT

Signal on the K1-bus to mask off interrupts on some or all card. Can be used to implement interrupts with priority levels.



RD_DATA

Read data from a K1-bus card. The lower data byte is read by the CPU directly while the upper data byte (if present) is latched into the high-to-low data latch.

WR_DATA

Write data to a K1-bus card. The lower data byte is supplied by the CPU directly while the upper data byte (if required) is supplied by the low-to-high data latch.

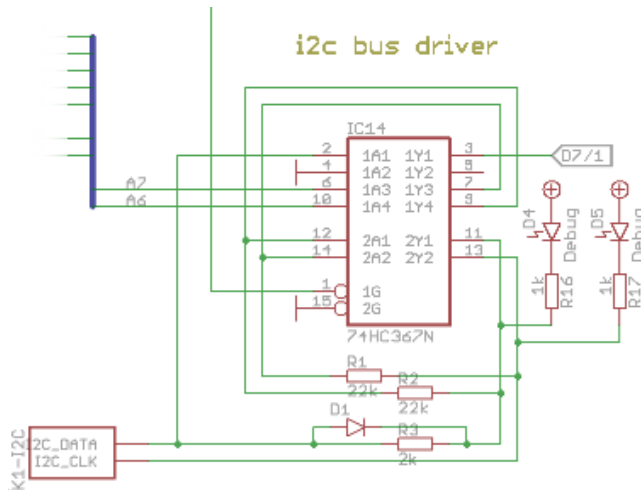
WR_HI

Write one byte of data into the low-to-high data latch for use in the next (probably immediately following)

WR_DATA cycle.

RD_HI

Read the upper data byte from the last RD_DATA cycle from the high-to-low data latch.



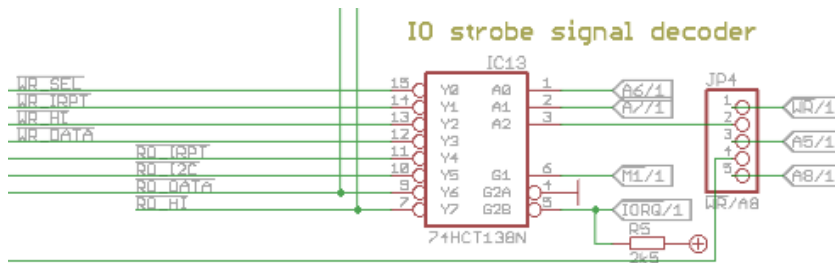
RD_I2C

Read or write to the I2C bus. Both is done by reading. The I2C data and I2C clock lines are set by A6 and A7, the value from the I2C data line is read on D7. The circuit is taken directly from my [K1-bus page](#) and was the only idea i had to use only one IC.

Errata: A6 and A7 cannot be used, because they are already in use to select the 74HC367! (They are used as select inputs to the strobe decoder.) Instead A3 and A4 are used.

The two LEDs are for debugging. It took me some time to find a place for some debugging lights on a CPU board without any i/o circuitry (except K1-bus) on it. :-)

74HCT138 enable:



A Z80 I/O bus cycle is strobed with **IORQ** by the CPU, so this is the first signal used to enable the signal decoder. But the CPU issues this signal for interrupt acknowledge cycles as well. This can be distinguished by the **M1** signal, which is activated for interrupts but not for I/O cycles, so **M1** must be high to detect an I/O cycle. **M1** starts 2.5 cycles (that is: long) before **IORQ** and remains approx. 10ns longer active (**M1**: 80ns max. and **IORQ** 70ns max. after $T3\uparrow$. These are marked with reference number 20 and 52 in the timing chart below) so it should be possible to use **M1** directly to mask the **IORQ** signal.

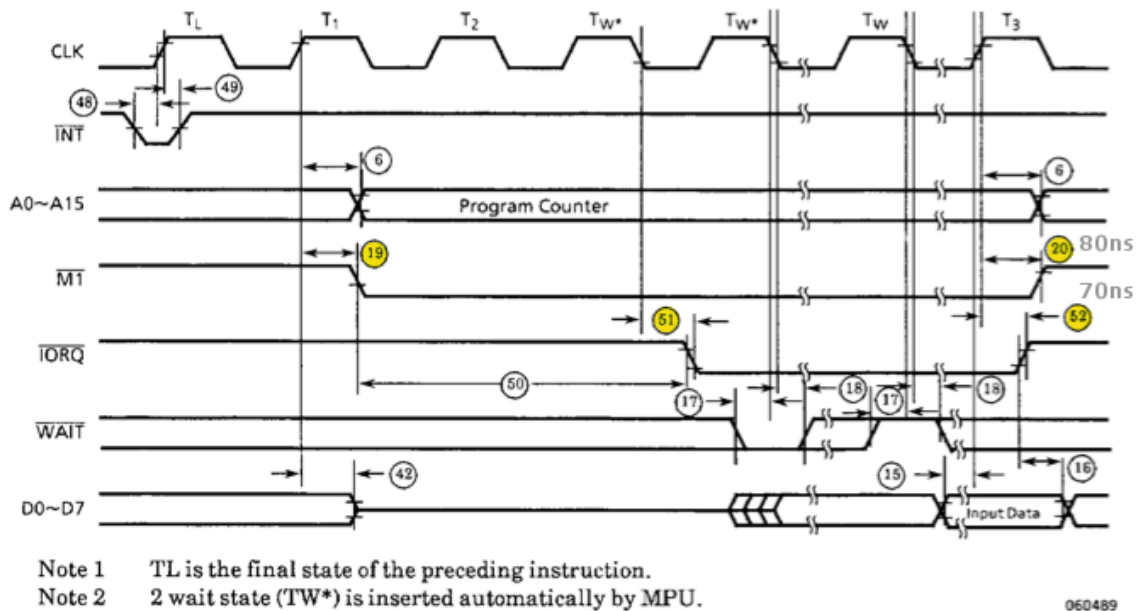


Figure 4.4 Interrupt Request/Acknowledge Cycle

Next is a timing problem: **WR** (and **RD**) and **IORQ** go up simultaneously: 70ns max. after $T3\downarrow$ as can be seen in the I/O timing chart below. This may result in spurious pulses at arbitrary outputs of the 74HCT138.
note: $T3$ in an I/O cycle is one cycle earlier than $T3$ in an int ack cycle: int ack inserts 2 automatic wait cycles before $T3$ and I/O only one.

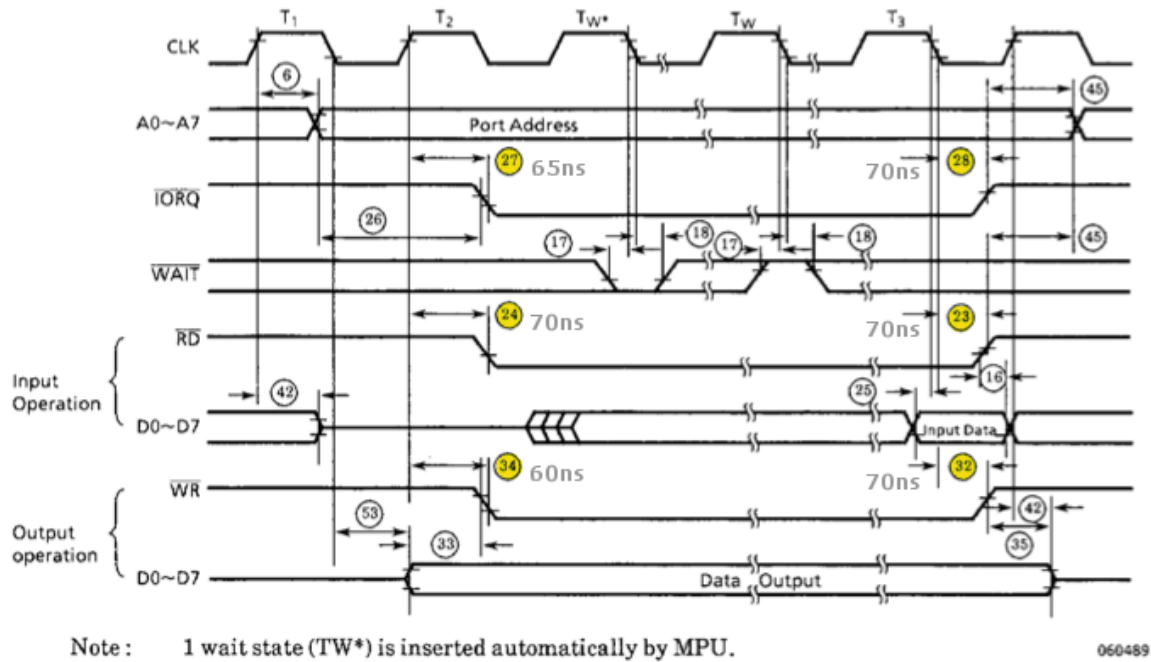


Figure 4.3 Input/Output Cycle





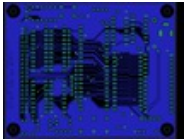
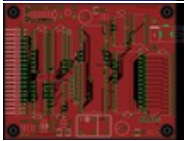

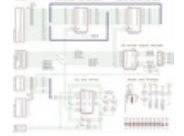

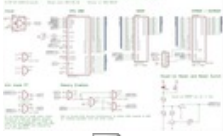








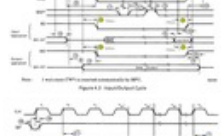

There are some ideas to overcome this:

- Make the **IORQ** signal end earlier:
Problem: edges can be delayed and not be brought forward. So some very clever circuitry would be needed to do this.
- Make the **WR** signal longer:
Adding a delay to the signal would do the job. But it would also delay the starting edge of the signal: One HCT gate adds approx. 10ns, so the front edge would at least move from 60ns to 70ns, which is now after the 65ns of the **IORQ** signal. Whatever we do with the **WR** signal, it will always at least add one gate latency to the front edge as well! So this is no easy solution either.
- Delay both signals to have time for front and end shaping:
Unfortunately we don't have any time available for delaying the end of the **IORQ** signal: in an output cycle data is only guaranteed to be stable for 30ns after **IORQ** goes up (time #35) which is just enough to propagate **IORQ** through the '138 to the strobe signal (HCT: typ. 19ns, 40ns max, HC: typ. 17ns, 30ns max)
- Use an address line instead of **WR**:
This is what i do in the Z80 reference design on my K1-bus webpage. This is safe in respect to no spikes on the strobe lines but has a small risk of bus collisions, e.g. if a program crashes. Since all lower 8 address lines are currently used up, we'd need **A8** as well, which will make block I/O opcodes impossible to use. But this restriction could be limited to K1-bus cards which actually use all 6 address lines if we reassign the address lines, so that the rarely used K1-bus **A5** will be driven by Z80 **A8**. I will use this design as a fallback for safety. This is what pin-header JP4 in the "IO strobe signal decoder" image above is for.
- Use a "strong" pull-up:
A method i have already used on my K1 CPU is "signal shaping" with "strong" pull-ups: If you add a low pull-up or pull-down resistor which draws "high" current, it will aid signal flipping into the pull-up's direction and delay signal flipping in the other. Applying a "strong" pull-up to the **IORQ** signal will slightly delay the starting edge of the pulse and slightly bring forward the ending edge. So what is "strong" here? The CMOS Z80 can only sink 2.0mA @ 0.4V, so a pull-up which supplies just this will be a reasonable choice: This is approx. 2.5kΩ. Disadvantage: The amount of time which can be shifted this way is tiny: some few ns only, but they may just be enough here.

Besides **WR** (or **A8**, if nothing else works) **A6** and **A7** are chosen for strobe selection, because they are in the low address half, making all I/O instructions usable and they are the only unused address lines because **A0** to **A5** are already used for addressing K1-bus card registers.

Archive

Name	Letzte Änderung	Länge
 build-qt.591.clang_64_kit-debug/	2017-10-05 15:42	26
build-qt.591.clang_64_kit-release/	2017-08-06 13:50	27

	Data sheets/	2015-01-31 12:06	16
	Emulator/	2017-10-20 17:21	15
	Images/	2016-04-10 16:55	9
	V1.0/	2015-01-31 12:38	9
	V1.1 board rear.png size: 835 × 628	2015-01-31 12:36	239161
	V1.1 board top.png size: 835 × 628	2015-01-31 12:35	314525
	V1.1 circuit io.pdf	2015-01-31 12:25	82454
	V1.1 circuit io.png size: 946 × 808	2015-01-31 12:39	293171
	V1.1 circuit main.pdf	2015-01-31 12:24	96882
	V1.1 circuit main.png size: 1191 × 732	2015-01-31 12:39	381993
	V1.1 part list.txt	2014-11-09 15:43	2752
	V1.1 placement.pdf	2014-11-09 15:34	47641
	V1.1 placement.png size: 628 × 472	2014-11-09 15:33	14801
	V1.1.brd	2015-01-31 12:11	185950
	V1.1.sch	2015-01-31 16:54	397226
	V2.0.b#1	2016-04-03 12:34	186221
	V2.0.brd	2016-04-04 20:36	186223
	V2.0.sch	2015-06-23 20:48	397071
	Z80 I:O cycle.png size: 693 × 445	2014-08-05 21:57	111670
	Z80 Int Ack cycle.png size: 682 × 418	2014-08-05 21:57	101547

powered by [vipsi](#) - your friendly VIP Script Interpreter