

spring 2023

RCA 2020

An RCA CDP1802 COSMAC revival project

Design, Build and Test Manual

RCA2020

An RCA CDP1802 COSMAC revival project

Designed by Richard van Harderwijk

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description/Changes** |
| V1 | Spring 2021 | Richard van Harderwijk | Initial Version |
| V2 | Spring 2023 | Richard van Harderwijk | Documenting takes a while… Corrected an error in the PCB design |
|  |  |  |  |

Content

[Abstract 4](#_Toc130325537)

[Introduction 5](#_Toc130325538)

[Design 6](#_Toc130325539)

[CPU-Unit 6](#_Toc130325540)

[ROM & RAM 6](#_Toc130325541)

[TTY-I/O 7](#_Toc130325542)

[Parallel 8-bit I/O 8](#_Toc130325543)

[I2C bus controller 9](#_Toc130325544)

[Schematics 11](#_Toc130325545)

[Build and Test 12](#_Toc130325546)

[CPU 12](#_Toc130325547)

[RAM and ROM 12](#_Toc130325548)

[TTY I/O and Utility (UT4b) 13](#_Toc130325549)

[Parallel 8-bit I/O 14](#_Toc130325550)

[I2C bus controller 15](#_Toc130325551)

[Annex A – References, Additional Documentation and Tools 17](#_Toc130325552)

[Annex B – Simple CPU test 19](#_Toc130325553)

[Annex C – PCB Layout 22](#_Toc130325554)

[Annex D - Jumper and TTY settings 24](#_Toc130325555)

[Annex D – Bill Of Materials 25](#_Toc130325556)

[Annex E – The OS - Utility UT4b 26](#_Toc130325557)

[ROM memory map 26](#_Toc130325558)

[UT4b change log 26](#_Toc130325559)

[Startup on reset code 26](#_Toc130325560)

[SCRT routines (Standard Call and Return Technique) 27](#_Toc130325561)

# Abstract

This document describes the design of a small microcomputer system based on the RCA 1802 COSMAC CPU. This 1970’s CPU seems still in production by Renesas [1] [2], also other manufactures compatible chips are available. (But better check eBay.)

The specifications of the system are:

* RCA 1802 CPU; 8 bit; up to 3,2 MHz clockspeed
* 64 kB RAM memory
* 8 kB ROM
* OS(ish): Utility to save, load, start programs; and examine registers and memory
* TTY serial terminal connection, software UART
* 8 bit parallel in/out
* I2C interface; and a clock/calender unit
* All signals available via connectors
* Single 5V via USB port

The design includes a PCB of 10 x 13 cm; and a build and test protocol per unit to facilitate error finding during build.

# Introduction

**Why would you want to build this system?**

Good question. The CPU is very old, from the 70’s and to my own surprise still seems to be in production by Renesas [1], [2].

My personal reason is this is one of my introductions to computers, I built a little system with this CPU in the early 80’s [6]. Having your own personal computer was expensive to almost impossible in those days. Fast forward; working in ICT for 30 years now, I wondered how my interest started 40 years ago. This regression project is the result of some browsing, acquiring a CPU on ebay and then ‘having to’ design a system around it.

Your reasons might be the simplicity. This simple system can help you understand modern computers in low level detail, because modern architecture is more or less the same although more extensive and advanced nowadays. Also you might have fun building some stuff yourself, the large size of the components make soldering relatively easy.

Ever programmed in assembler? The architecture of the CPU is easy to understand and well documented. Fun fact, the architecture was different from other CPU’s in those days (Intel 8008 etc): central part of the 1802 architecture are 16 registers of 16 bit. It resembles modern RISC architectures (a little bit). One of the big differences is that RISC architectures facilitate direct register to register operations, the 1802 performs them via the accumulator (called D-register) or stack.

In addition to the standard input/output, I added an I2C interface to make the system a bit more interesting and facilitate the connection of equipment.

**Stepwise build and test protocol**

To make construction manageable, the build is split in construction units with a test protocol per unit. First the basic CPU is constructed and tested; then memory is added; the TTY interface and a Utility to run and examine the computer. Finally, the parallel input/output and I2C interface is added.

**Reading guide**

If you are completely new to this CPU, best document to start is: RCA’s User manual; MPM-201 [4]. This explains the architecture of the CPU, the instruction set and programming techniques.

Books with more information and explanation are available on the web (Dutch and English, in a bit archaic language) [6] [7]. If you have a technical background, you can find technical details in the datasheet: [2]

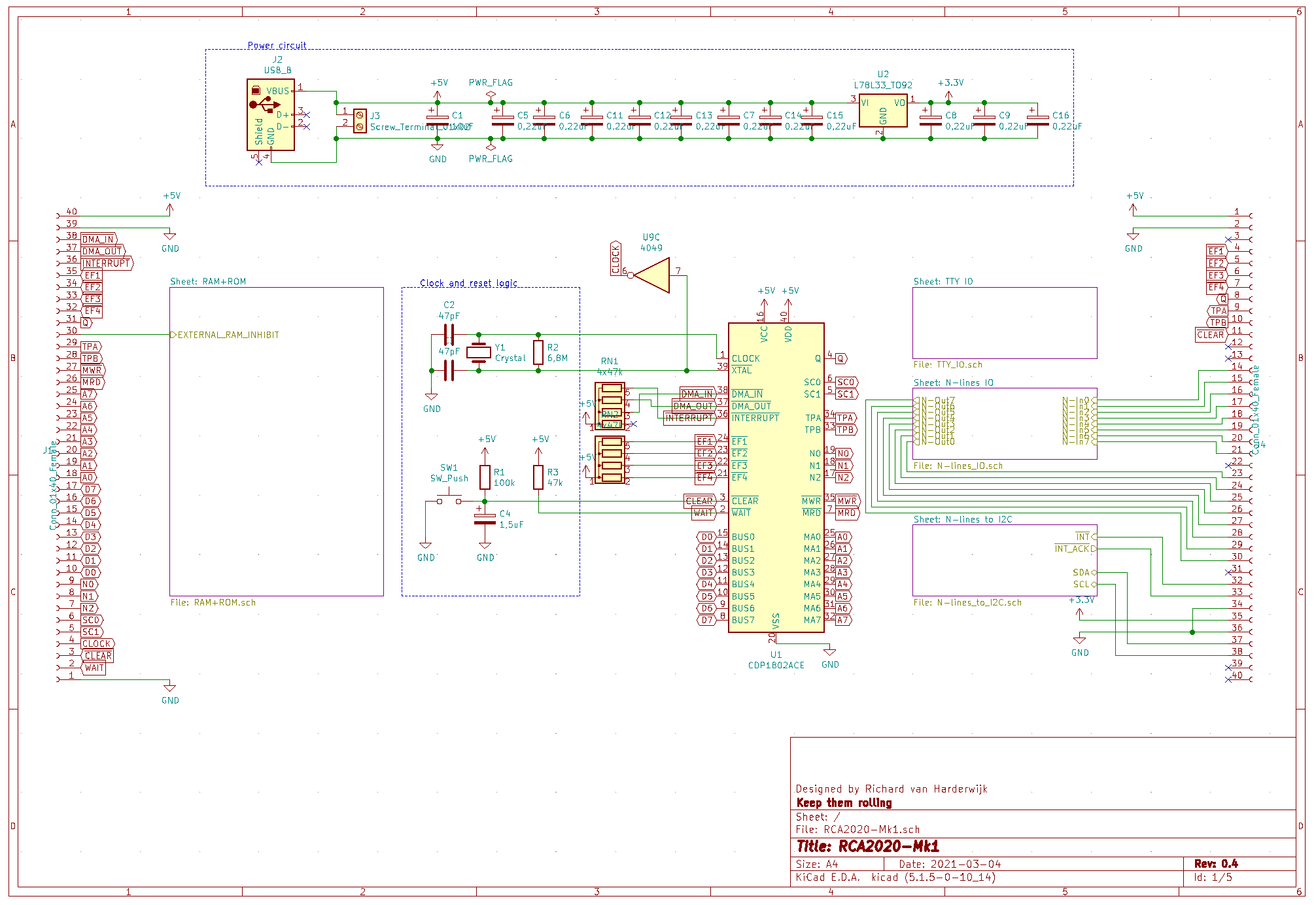
Then, after reading this document you have quite a good understanding of the system and will also be able to understand the other documents listed in: Annex A – References, Additional Documentation and Tools

# Design

## CPU-Unit

Design considerations:

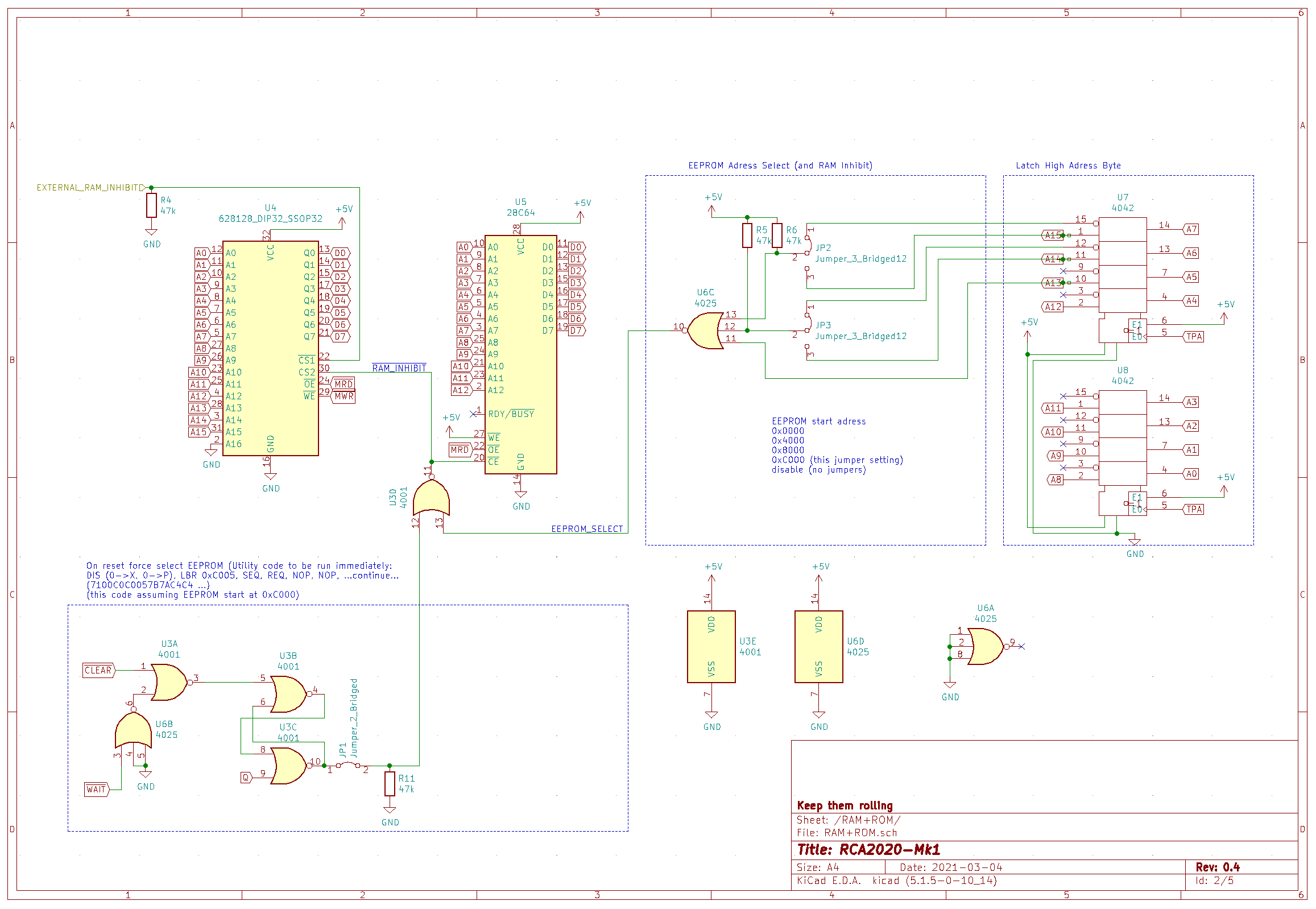
* Not much to consider, general guidelines in [2], [4]
* The CPU has some pull-up resistors; and a power up reset including a reset button.
* The clock circuit is CPU internal and has a buffer to the extension port (J1) [3].
* There is a USB-port and an additional screw terminal for 5V power. Power consumption is very low. In the build and test chapters there are some measurements. The ROM and RAM are the top power users. The CMOS 1802 uses little power, that is one of the reasons it was selected as the Galileo Jupiter satellite computer [8]



## ROM & RAM

Design considerations:

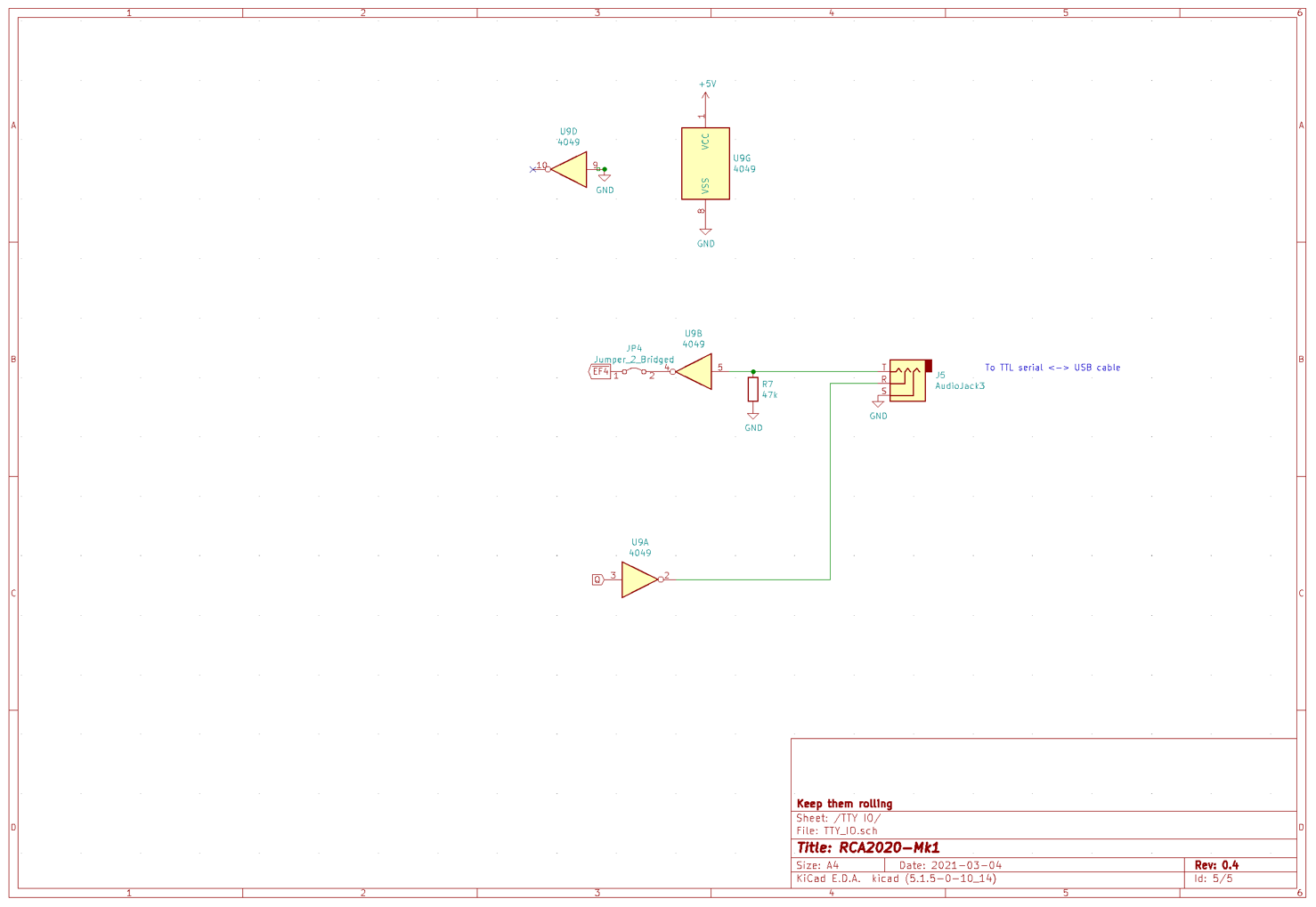
* The address bus of the 1802 is multiplexed, the two 4042’s (U7, U8) latch the high address.
* With modern chips it is easy to just have max RAM available. The selected RAM is limited to 64kB. The RAM is deselected if the ROM is selected (port U6C).
* No databus pullup necessary, always something is selected.
* ROM can start at 0x0000; 0x4000, 0x8000 (Elf compatibility [9]) and 0xC000 (Dutch Cosmicos compatibility [6] and to have 48 kB consecutive mem available). See: Annex D - Jumper and TTY settings
* A simple flipflop (U3B, U3C) force selects ROM on reset; the flipflop is reset by pulsing Q (inspiration from the Cosmicos [6] ). Only after completion a better idea came to mind, using address line A3 to reset.
* The second RAM select line is used for an External RAM inhibit, for example if you want to experiment with a memory mapped IO device.



## TTY-I/O

Design considerations:

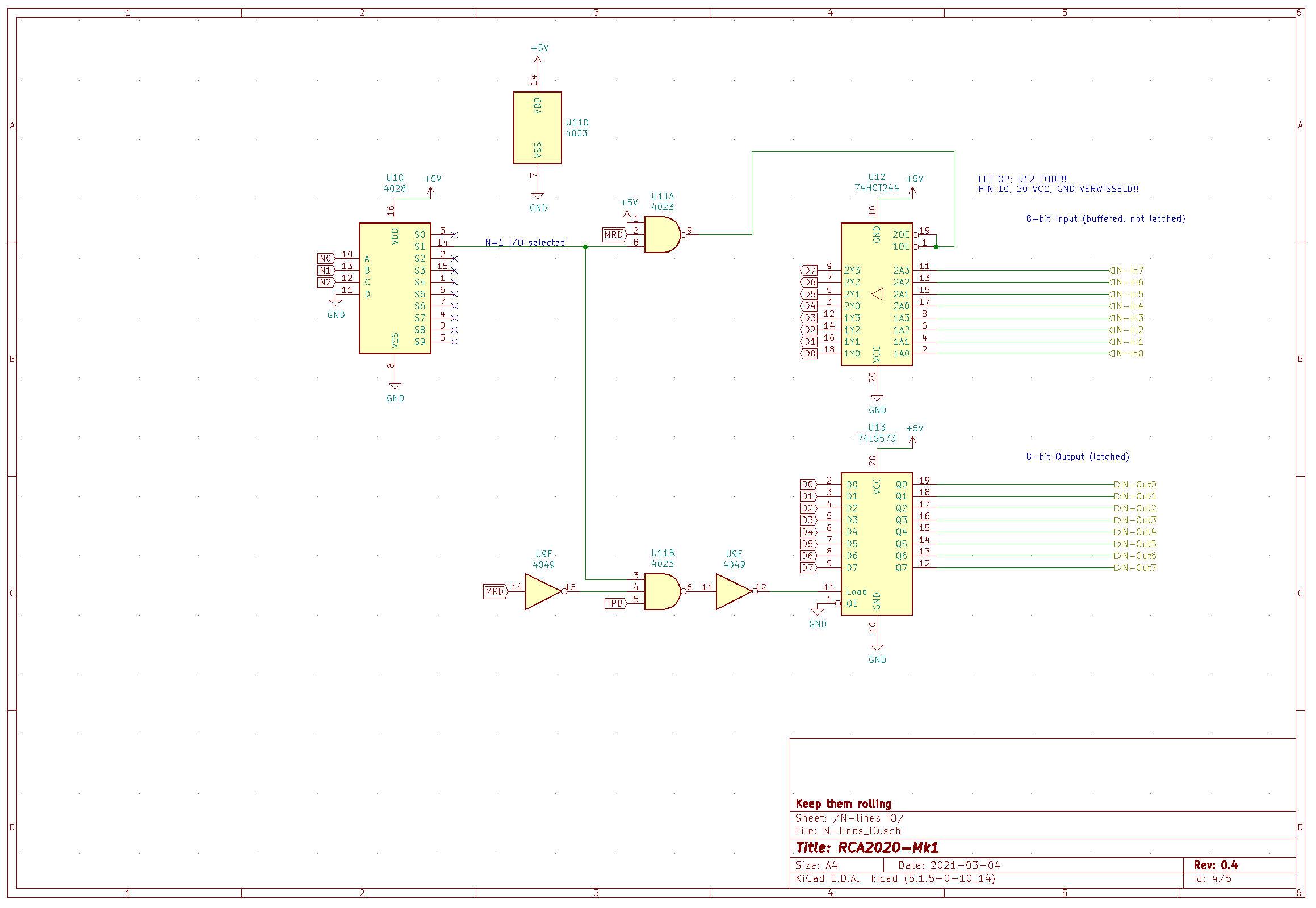
* Almost none, no RS232 necessary with a modern TTY->USB cable (I used one with the FTDI 232RL chipset).
* With jumper (JP4) you can switch off this circuit to use EF4 in another way



## Parallel 8-bit I/O

Design considerations:

* The output is latched, the input buffered.
* Input and output strobes are described in [4, figure 111 and 112]
* This part is optional



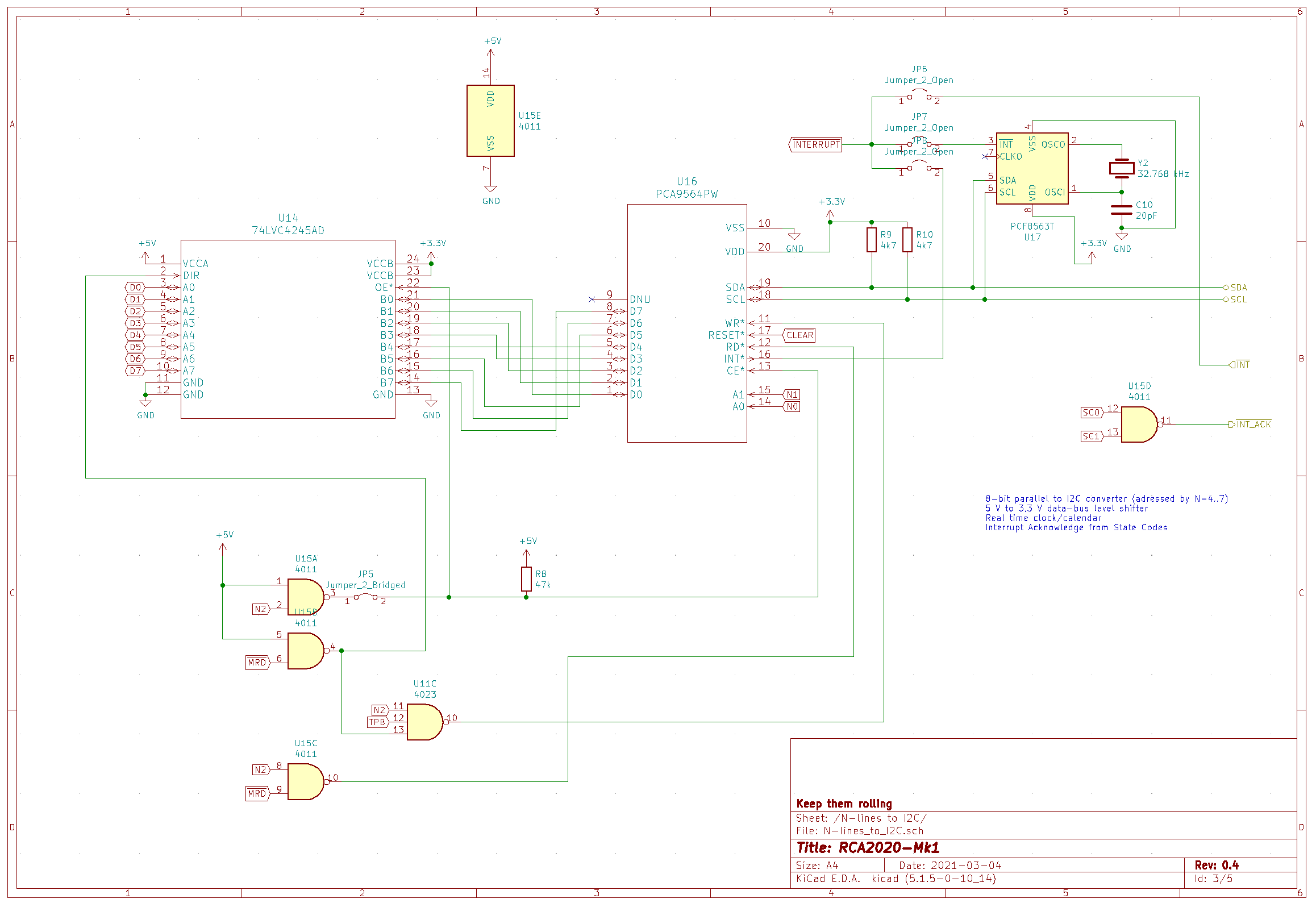
Error in connection U12 (pin 10, 20)

To be corrected in Mk2 version

## I2C bus controller

Design considerations:

* To make things more interesting, to add the possibilities for various chips and a clock/calendar, I added a parallel to I2C interface
* First selection was the PCF8584 I2C bus-controller, this one is 5V compatible. But the timing was incompatible; ao. the read pulse width is max 1 μS, while the 1802 produces a vastly longer one.
* The PCA9564 had compatible timing, but levels of 3.3V. This is solved with a bidirectional level shifter (74LVC4245) [10].
* With jumper JP5 the whole circuit can be disabled to have N=4 through N=7 for other use.
* The interrupt signals are open collector and can be wired together to the 1802
* The !Int\_Ack signal is generated by both state codes SC0 and SC1 high [4, figure 102]
* PCF8563 as the real time clock/calendar [11].
* And now the drivers… [12]
* This part is optional



# Schematics

Here, 90 degrees rotated? Or in the design chapter?

# Build and Test

## CPU

Components:

|  |  |  |  |
| --- | --- | --- | --- |
| PCB | Y1 | C1, C2, C3, C4 | J1, J2, J3 |
|  | R1, R2, R3 | C7, C11, C12 |  |
| U1 | RN1, RN2 | SW1 |  |

Mind isolation of metal parts, for example the bottom of J2 touching the PCB.

Afbeelding met tekst, elektronica, circuit

Automatisch gegenereerde beschrijving

Testing

* Before inserting the CPU in it’s socket, perform the usual checks like: visual inspection; is the power on the right pins; short circuits, etc.
* Test this part according to the CPU test described in: Annex B – Simple CPU test

Power consumption: 7 mW (Run: 1,31 mA Reset 0,83 mA @ 5,02V)

## RAM and ROM

Components:

|  |  |  |  |
| --- | --- | --- | --- |
| U3, U6, U7, U8 | R4, R5, R6, R11 |  |  |
| U4 | C6, C14, C15 |  |  |
| U5 | JP1, JP2, JP3 |  |  |

Afbeelding met tekst, elektronica, circuit

Automatisch gegenereerde beschrijving

I used sockets for all IC’s to facilitate hardware bugfixing (the Mk1 is my prototype).

Test:

* Insert IC’s U3, U6, U7, U8; do not insert the RAM and ROM yet
* Check RAM select on reset (no jumpers yet)
* Measure output flipflop (JP1)
* Set JP2, JP3 to 0x0000 to check EEPROM select on reset

It is a bit extensive (performed on the prototype), but quick and simple. Feel free to skip.

With the RAM and ROM inserted:

Power consumption: 20 mW (Run: 2,7 - 4 mA ; Reset 0,87 mA)

## TTY I/O and Utility (UT4b)

Components:

|  |  |  |  |
| --- | --- | --- | --- |
| U9 | J5 |  |  |
| R7 | JP4 |  |  |
| C5 |  |  |  |

Afbeelding met tekst, elektronica, circuit

Automatisch gegenereerde beschrijving

Test this with UT4b (Annex E – The OS - Utility UT4b) in the ROM and a TTL-USB cable to use your very powerful current computer as a 1970’s 300 baud dumb terminal (for settings see: Annex D - Jumper and TTY settings). The use and design of Utility UT4 is extensively described in [5] [13].

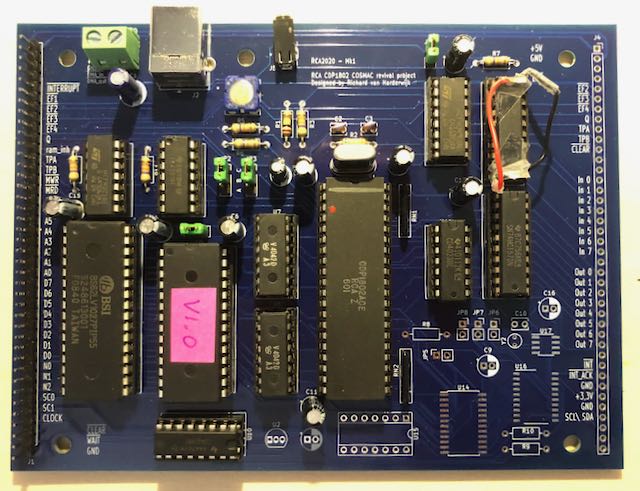
## Parallel 8-bit I/O

This part is optional.

Components:

|  |  |  |  |
| --- | --- | --- | --- |
| U10, U11, U12, U13 |  |  |  |
| C13 |  |  |  |
|  |  |  |  |

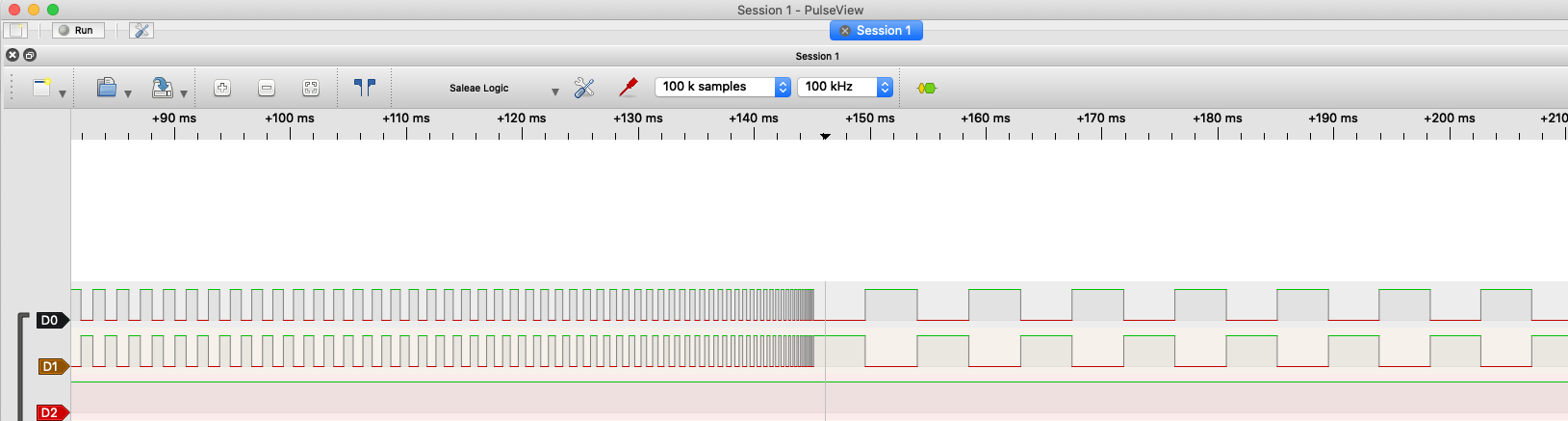
Remark: don’t install J4 yet, but after soldering the SO-IC’s



See U12 (upper right corner), in the prototype board (Mk1) was one error: VCC + GND (pins 20 and 10) wrongly connected (whoops). In the photo you can see the correction with two wires. This is corrected in the Mk2 version.

Test:

* Check UT4b outputs 0x01 on Out1 on reset
* Write a small test program toggling Out1 with decreasing pulse-width

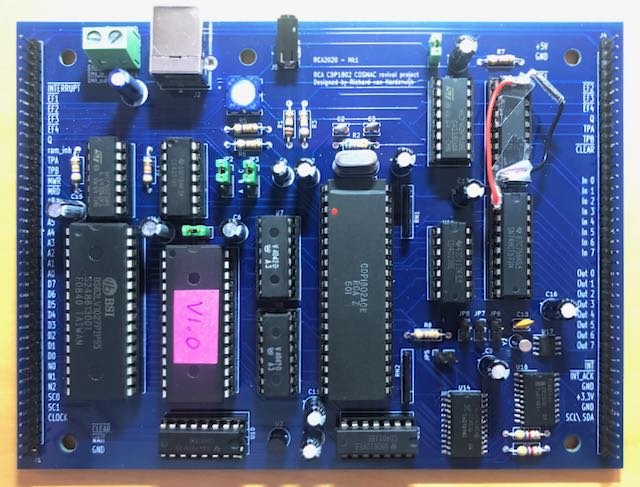


## I2C bus controller

This part is optional.

Components:

|  |  |  |  |
| --- | --- | --- | --- |
| U14, U16, U17 | Y2 | JP5, JP6, JP7, JP8 |  |
| U15 | C8, C9, C10, C16 | J4 |  |
| U2 | R8, R9, R10 |  |  |



Test? First drivers for the I2C controller. Volunteers?

# Annex A – References, Additional Documentation and Tools

**References**

[1] <https://www.renesas.com/us/en/products/space-harsh-environment/mil-std-883-products/mil-std-883-microprocessors-and-peripherals/cdp1802a-cmos-8-bit-microprocessors>

[2] Datasheet FN1441 - CDP1802AC/3 High-Reliability CMOS 8-Bit Microprocessor (Rev 3.00 October 17, 2008)

<https://www.renesas.com/us/en/document/dst/cdp1802ac3-datasheet?language=en&r=496496>

[3] Application Note AN6565 - Design of Clock Generators for Use with COSMAC Microprocessor CDP1802 (Rev.1.00 March 1997) <https://www.renesas.com/us/en/document/apn/an6565?language=en&r=496496>

[4] MPM-201 User Manual for the CDP1802 COSMAC Microprocessor <http://www.bitsavers.org/pdf/rca/1802/MPM-201A_CDP1802_User_Manual_1976.pdf>

[5] MPM-203 Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor <http://www.bitsavers.org/components/rca/cosmac/MPM-203_CDP1802_Evaluation_Kit_Manual_Sep76.pdf>

[6] <http://retro.hansotten.nl/radio-bulletin/1802-cosmicos/>

[7] Tom Swan – Programmer’s Guide to the 1802 - ISBN 0-8104-51B3-2. Seems to be downloadable on the web.

[8] <https://history.nasa.gov/computers/Ch6-3.html>

[9] <http://www.cosmacelf.com>

[10] NXP - Application Note AN2433/D; Rev. 0, 12/2002; 5V to 3V Design Considerations <https://www.nxp.com/docs/en/application-note/AN2433.pdf>

[11] PCF8563; Real time clock/calendar; Rev. 11 — 26 October 2015 <https://www.nxp.com/docs/en/data-sheet/PCF8563.pdf>

[12] Philips Application Note AN10148; PCA9564 – I2C-bus controller <https://www.nxp.com/docs/en/application-note/AN10148.pdf>

[13] BMP-802 Design ideas book

<http://www.cosmacelf.com/publications/books/design-ideas-book-for-the.pdf>

[14] Memory Application Briefs

<http://www.cosmacelf.com/publications/books/rca-microprocessor-memory.pdf>

**Additional Documentation**

**System 00**

If you want to go to an ever deeper level: ‘System 00’ is the start of the whole cosmac CPU family. Early 1970’s, this computer is made from discrete TTL components. Maximum understanding, insight (or confusion) guaranteed.

<http://www.retrotechnology.com/memship/cosmac_system_00.html>

<http://www.cosmacelf.com/publications/books/system-00-manual.pdf>

There are various high level languages available, for example this C compiler. (I have not tried this.)

<https://sites.google.com/site/lcc1802/>

**Tooling used**

A big thank you! to the people building and maintaining these tools:

KiCad (From schematics to PCB)

<https://www.kicad.org>

PulseView (Logic Analyzer)

<https://sigrok.org/wiki/PulseView>

HexFiend (Hex editor for MacOS)

<https://hexfiend.com>

EEPROM programmer (With an Arduino Mega and Windows app eewriter)

<http://danceswithferrets.org/geekblog/?page_id=903>

CoolTerm (Terminal emulator)

<http://freeware.the-meiers.org>

1802 online assembler

<https://www.asm80.com/onepage/asm1802.html>

# Annex B – Simple CPU test

So you have acquired an 1802, now there is a simple way to test it, with just a few components. I used my Arduino as a logic analyzer (later a € 10,- device with 24 MHz bandwidth). See photo, the necessary components are the power-on reset, clock circuit and some pull-up resistors.

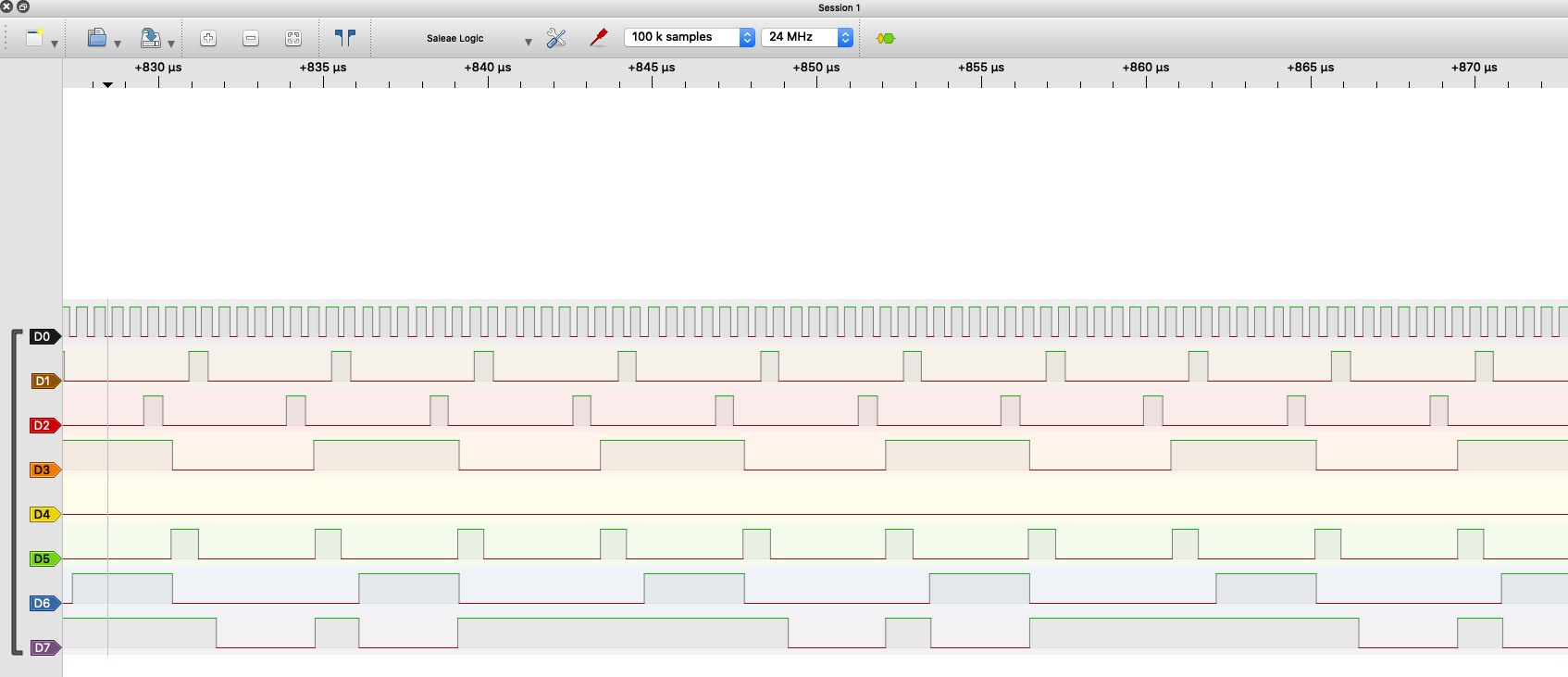
Afbeelding met elektronica

Automatisch gegenereerde beschrijvingConnections:

* Pin 40 (Vdd) and 16 (Vcc) on +5V
* Pin 20 (Vss) on Gnd
* !Wait on +5V
* !Clear (power-on reset) and clock circuit: according to schematics CPU-Unit
* !DMA-in, !DMA-out, !Interrupt, !EF1-4 on +5V
* The databus via resistors (22-47k) pull-up on +5V (an 8 resistor bank on the photo)
* Extra elco’s on the power bus

Connect your logic analyzer with: Channel 0-7 = Clock, TPA, TPB, SC0, SC1, MRD-not, MA0, MA1

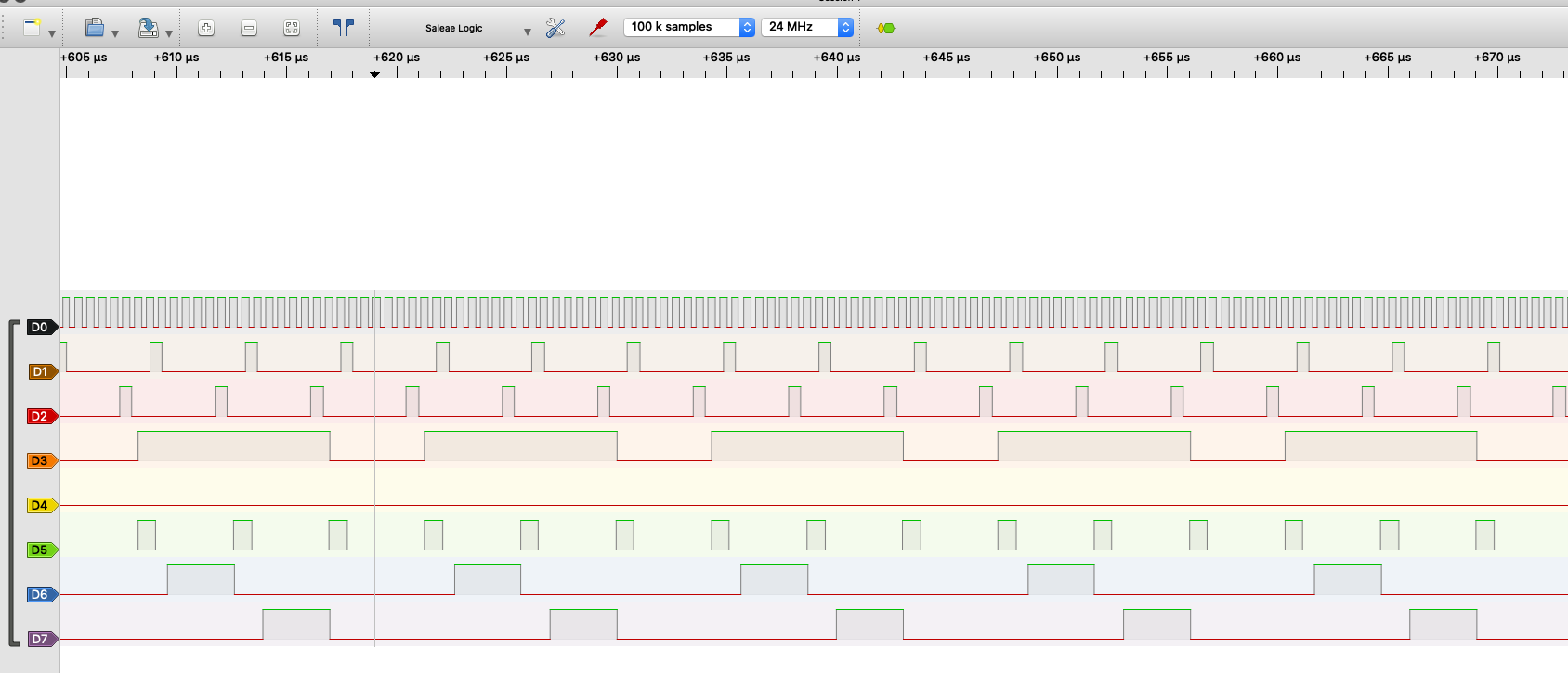
If your 1802 chip is ok, you will see something like this.



Conclusions:

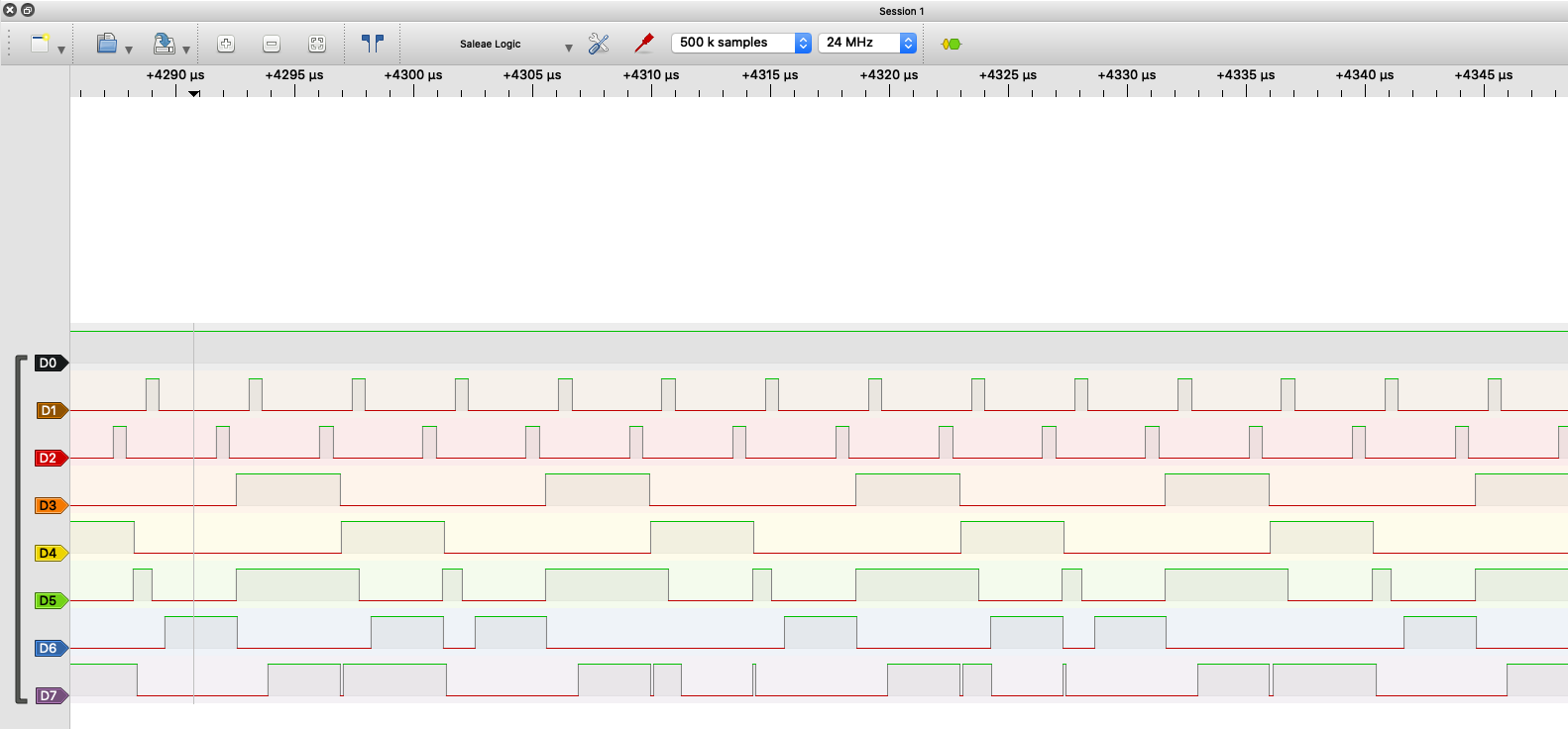
* There is a clock signal and TPA and TPB signals. This means the internal clock circuitry works.
* The state codes show alternating fetch-execute codes. This means the CPU is loading and executing instructions (0xFF = Subtract Memory Immediate with 0xFF). The state machine and control logic work.
* The !MRD signal shows fetching the instruction and the immediate operand.
* The address lines show addresses running through the ‘memory’ with the high byte at TPA. The registers, incrementor and memory-mux of the CPU work.
* If this all works, you can be confident the whole chip works
* When running this test, A7 wiggles with ~ 2 Hz (This is the highest address bit-A15; this is best seen with an analog multi meter)
* The Arduino logic analyzer has a max bandwith of ~ 2 MHz, so there might be no clear clock and aliasing of the TPA and TPB signals

While playing at it, with the databus resistors, connect them differently, pull-up and pull-down as 0xC0 (=long branch, one fetch and two execute cycles)



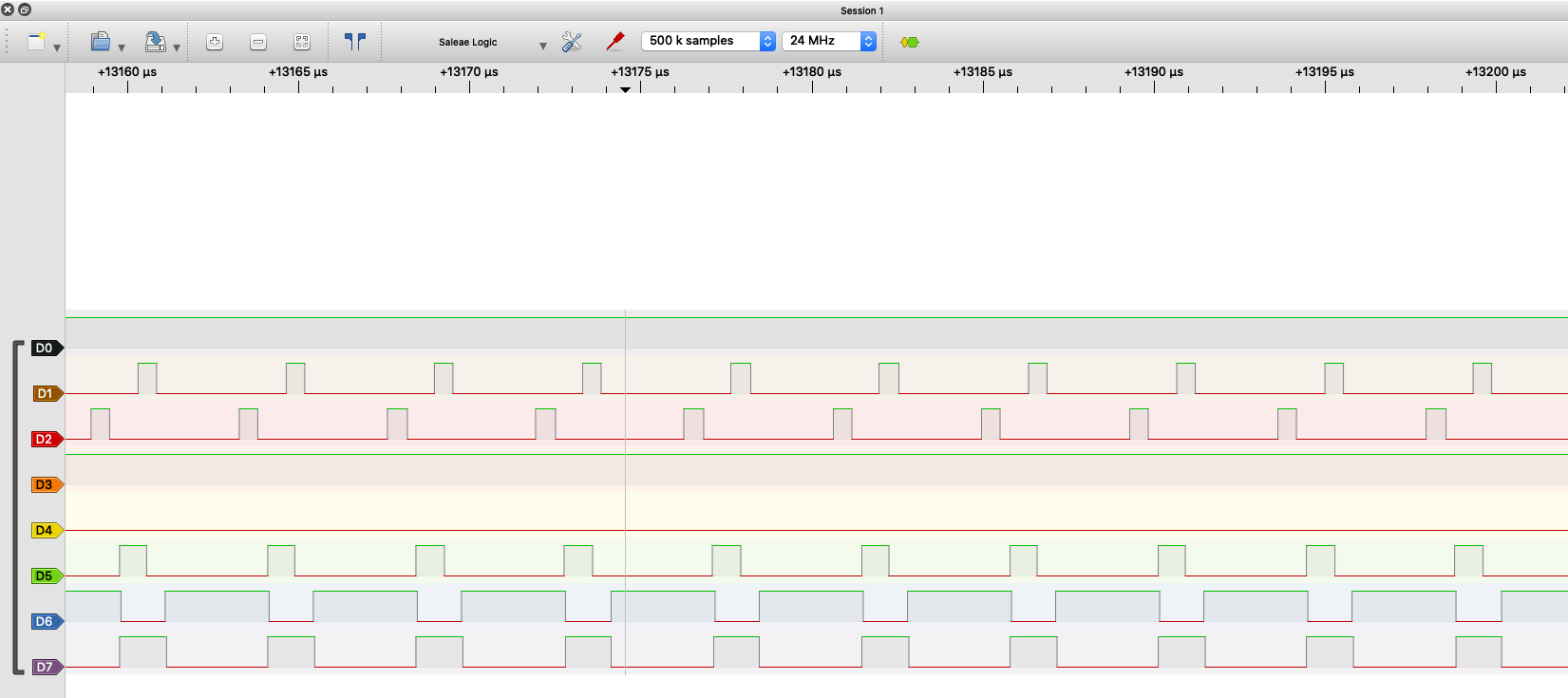
An interesting trick is to connect !DMA-Out to SC1. See [13] and [14-ICAN6704]

You will get consecutive fetch-execute-DMA cycles, see the state codes:



If your 1802 does all this, it works.

As a bonus play, I let the databus floating (no resistors) to get the wildest data input in the CPU. It stumbled on the 0x00 (=IDLE) instruction, and that is just what it did:



When I=0 and N=O, the microprocessor repeats execute (SI) cycles until an I/O request. You see the state codes stay in S1 and the address stays the same.

# Annex C – PCB Layout

Needs to be updated (U12) -> Mk2 version

Not in scale

Silk screen – component placement

Afbeelding met tekst, elektronica, circuit

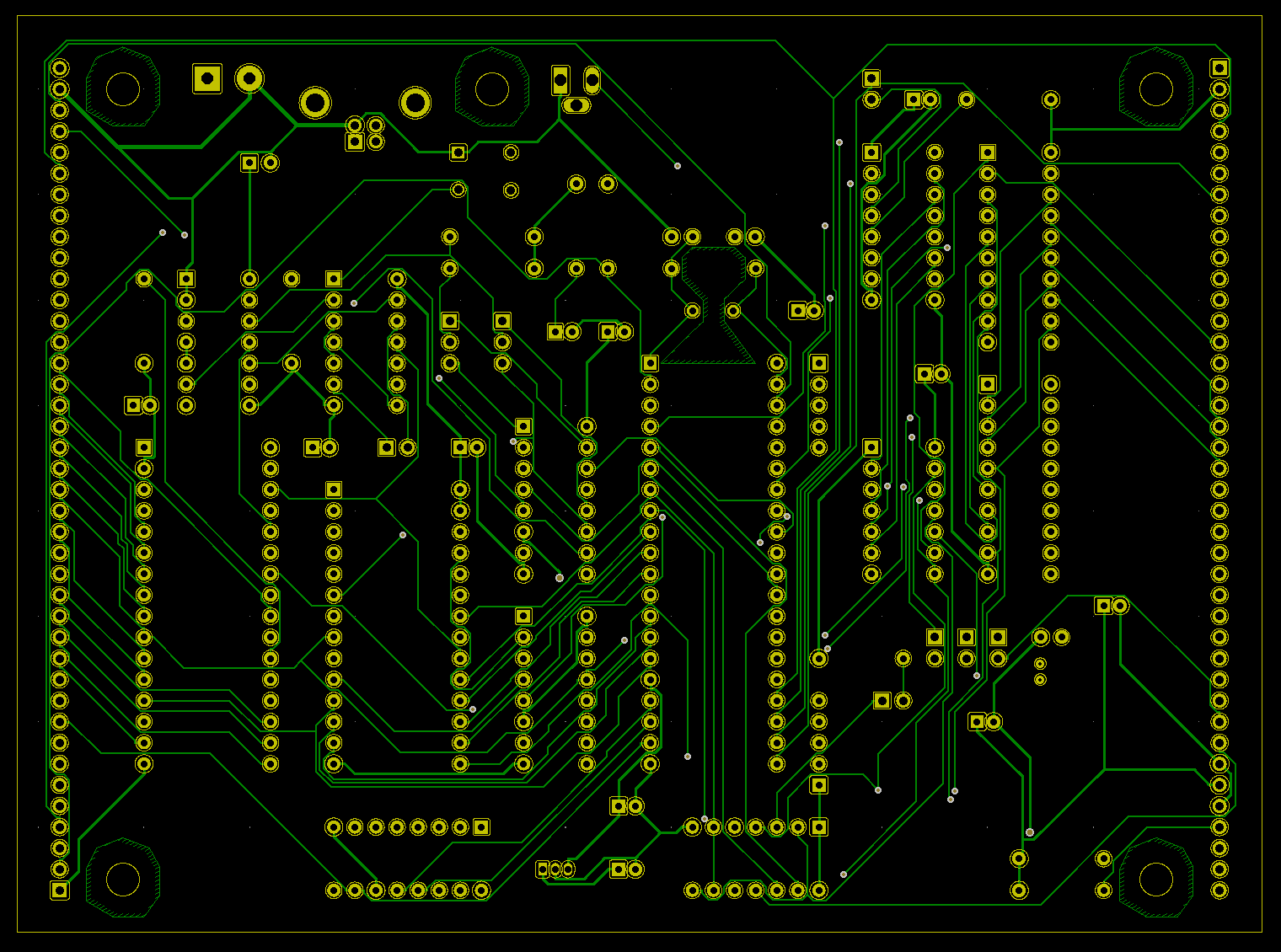
Automatisch gegenereerde beschrijving

Front Copper layer

Afbeelding met tekst, elektronica

Automatisch gegenereerde beschrijving

Back copper layer (copper protection zones to be removed from picture)



# Annex D - Jumper and TTY settings

**JP2, JP3** setting (for ROM start address)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | JP2-AB | JP2-BC | JP3-AB | JP3-BC |
| 0x0000 | √ |  | √ |  |
| 0x4000 | √ |  |  | √ |
| 0x8000 |  | √ | √ |  |
| 0xC000 |  | √ |  | √ |

**JP1**

Connect for run ROM on reset (not necessary if ROM start = 0x0000)

Remove to not use autostart in ROM

Note, ROM code needs to toggle Q (SEQ, REQ) to reset flipflop

**JP4**

Connect for TTY input to EF4

Remove to release EF4 for other use

**JP5**

To select I2C devices. Remove to deselect.

**JP6, JP7, JP8**

To connect (open-collector) the I2C controller; the I2C clock; and the J4 interface connector interrupts to the 1802 interrupt. Depending on your application and use.

**TTY**

CoolTerm settings:

Serial Port

300 Baud

8 databit, no parity, 1 stopbit

Flow Control: XON

Terminal Options:

Enter key emulation: CR

Remove High Bit from 8-bit characters

# Annex D – Bill Of Materials

Add PCB, TTY-USB cable, jumper things, (1802 remark ebay) etc to BOM



# Annex E – The OS - Utility UT4b

Version 1 of the RCA2020Mk1 ‘OS’

## ROM memory map

8kB EEPROM, to be placed at 0xC000-0xDFFF

|  |  |  |  |
| --- | --- | --- | --- |
| **Mem map** | **ROM start** | **ROM end** |  |
| 0xC000 | 0x0000 | 00FF | On reset->run code. Room for (long branch to) other initialization code. End with branch to UT4b (256 Bytes) |
| C100 | 0100 | 02FF | Start of UT4b (½ kB) |
| C300 | 0300 | 03FF | Start of utility codes (SCRT ao) (256 Bytes) |
| C400 | 0400 | 1FFF | Free (7kB of ROM) (0xC000-0xDFFF) |

## UT4b change log

Changes to original UT4 [5] [13]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Original address** | **UT4b address** | **Old values** | **New values** | **Comments** |
| 8001 | C101 | F880B0 | F8C1B0 | Load start page (C1 instead of 80) |
| 8004 | C104 | F88CB1 | F8E0B1 | Register save on reset address (E000-E01F) |
| 8033 | C133 | 6101 | No change | Outputs 0x01 on N=1 at reset |
| 80CF | C1CF | C081F8 | C0C2F8 | Long Branch to finish error |
| 80EC | C1EC | 7000 | 7100 | Start user prog with IE=0 |
| 814B | C24B | 6780 | C4C4 | Replace Output on N=7 with two NOP’s (Output was for paper tape reader, is now I2C) |
| 8159 | C259 | 6740 | C4C4 | Replace Output on N=7 with two NOP’s |
| 81FC | C2FC | C08039 | C0C139 | Long branch to start |

UT4 is an example of a bit-banging serial communication interface (software UART). Funky 70’s code, worth exploring. All in just ½ kB.

## Startup on reset code

On reset this code force selects the EEPROM (Utility code to be run immediately:

C000 .ORG $C000

C000 71 00 DIS 00

C002 C0 C0 05 LBR 0xC005

C005 7B SEQ

C006 7A REQ

C007 C4 NOP

C008 C4 NOP

C009 C0 C1 04 LBR 0xC104

* This code assuming EEPROM mapped at 0xC000
* Interrupt disabled (your code needs to initialize and enable interrupts)
* The SEQ, REQ pulse Q and reset the flipflop (But with connecting the flipflop to A3, this is not necessary anymore, see ROM & RAM design.)
* After the two NOP’s and before the LBR to UT4b other initialization code (or LBR’s to) can be added. (I2C initialization code, etc).

## SCRT routines (Standard Call and Return Technique)

See [4]

Mapped to 0xC300

.ORG $C300

SEP R3

SEX R2 ;(enter Call here)

GHI R6

STXD

GLO R6

STXD

GHI R3

PHI R6

GLO R3

PLO R6

LDA R6

PHI R3

LDA R6

PLO R3

BR 0x00

SEP R3

GHI R6 ;(enter Return here)

PHI R3

GLO R6

PLO R3

SEX R2

INC R2

LDXA

PLO R6

LDX

PHI R6

BR 0x10