

Computer architecture

S80 computer

Ver. 2.0

A retro Z80 with memory mapping

Filip Pynckels

September 21, 2019

Update October 19, 2019

Update April 14, 2020

Update May 31, 2020

Documents:

http://users.telenet.be/pynckels/2019-3-S80-retro-Z80-computer-with-stackable-segments_ver_2-0.pdf

http://users.telenet.be/pynckels/2019-2-S80-retro-Z80-computer-with-stackable-segments_ver_1-0.pdf

Sources, schematics, printed circuit boards:

http://users.telenet.be/pynckels/2019-3-S80-retro-Z80-computer-with-stackable-segments_ver_2-0.zip

http://users.telenet.be/pynckels/2019-2-S80-retro-Z80-computer-with-stackable-segments_ver_1-0.zip

Table of contents

Table of contents	2
List of figures	3
List of tables	3
Introduction	4
1 Architecture	7
1.1 The Princeton architecture.....	7
1.2 The Harvard architecture	7
1.3 The S80 version 2.0 architecture	8
2 Design choices and specifications	9
2.1 Design guidelines	9
2.2 PCB design rules	10
3 Processor 2.0 board.....	11
3.1 Processor 2.0 - Description.....	11
3.2 Processor 2.0 - Technical specifications	11
3.3 Processor 2.0 - Functional schematics.....	12
3.4 Processor 2.0 - PCB design	12
3.5 Processor 2.0 - Correction for the memory 2.0 board	13
4 Serial 2.0 board.....	16
4.1 Serial 2.0 - Description.....	16
4.2 Serial 2.0 - Technical specifications.....	16
4.3 Serial 2.0 - Functional schematics.....	17
4.4 Serial 2.0 - PCB design	18
5 Memory 2.0 board	19
5.1 Memory 2.0 - Description	19
5.2 Memory 2.0 - Find the bug.....	20
5.3 Memory 2.0 - Technical specifications	21
5.4 Memory 2.0 - Functional schematics	21
5.5 Memory 2.0 - Functional schematics - Testing.....	23
5.6 Memory 2.0 - Timing study.....	28
5.7 Memory 2.0 - Populated board test	28
5.8 Memory 2.0 - PCB design	30
6 Software development	31
7 Conclusion.....	32
Appendix 1 - VHDL memory addressing logic	33
Appendix 2 - VHDL memory addressing simulation results	39
Appendix 3 - VHDL memory addressing simulation results	40
Appendix 4 - Offline memory test - Controller program	41

List of figures

Figure 1 - S80 version 1.0	4
Figure 2 - S80 version 1.0 - Do it yourself kit	5
Figure 3 - Architecture - Princeton.....	7
Figure 4 - Architecture - Harvard.....	7
Figure 5 - Architecture - S80 version 2.0.....	8
Figure 6 - Design specifications - Sizes	9
Figure 7 - Design specifications - Headers and breadboard.....	10
Figure 8 - Processor 2.0 - Functional schematics.....	12
Figure 9 - Processor 2.0 - PCB.....	13
Figure 10 - Emulation - Reset signal (22 μ F, 1K Ω , 1K Ω).....	13
Figure 11 - Emulation - Reset signal (100 μ F, 4.7K Ω , 100 Ω).....	14
Figure 12 - Emulation - Reset signal (100 μ F, 4.7K Ω , 0 Ω).....	14
Figure 13 - Processor 2.0 – Reset signal (100 μ F, 4.7K Ω , 0 Ω).....	15
Figure 14 - Processor 2.0 – Power-on signal (100 μ F, 4.7K Ω , 0 Ω)	15
Figure 15 - Serial 2.0 - Functional schematics.....	17
Figure 16 - Serial 2.0 - PCB	18
Figure 17 - Serial 2.0 - Temperature test.....	18
Figure 18 - Memory 2.0 - First test board	20
Figure 19 - Breadboard test of the SN74LS670 component	20
Figure 20 - Memory 2.0 - Functional schematics	22
Figure 21 - Memory 2.0 - VHDL simulation schematic	23
Figure 22 - VHDL test of the memory addressing logic.....	23
Figure 23 - Breadboard test of the memory addressing logic	24
Figure 24 - Front side of the non-populated prototype board	24
Figure 25 - Backside of the non-wired prototype board	25
Figure 26 - Backside of the wired prototype board.....	25
Figure 27 - Memory 2.0 - Test board	26
Figure 28 - Memory 2.0 - Addressing logic analysis at power-on.....	26
Figure 29 - Memory 2.0 - Addressing logic analysis at reset.....	27
Figure 30 - Memory 2.0 - 74LS670 logic at power-on	27
Figure 31 - Memory 2.0 - 74LS670 logic at power-on	27
Figure 32 - Memory 2.0 - Offline controller tests.....	29
Figure 33 - Memory 2.0 - Controller test results.....	29
Figure 34 - Memory 2.0 - Full-stack memory test result	30
Figure 35 - Memory 2.0 - 514Kb memory PCB	30

List of tables

Table 1 - Processor 2.0 - Technical specifications.....	11
Table 2 - Serial board 2.0 - Technical specifications	17
Table 3 - Memory 2.0 - Technical specifications.....	21

Introduction

Welcome to the second document concerning the construction of an expandable Z80 retro computer. Our goal is to explain the design, building and programming of a **Z80 based computer with retro technology**. The S80 version 1 document (see link on page 1) described the construction of the “*S80 version 1.0*” (see *Figure 1*). Discussions with readers and resulting insights led to the design and construction of a “*S80 version 2.0*”.

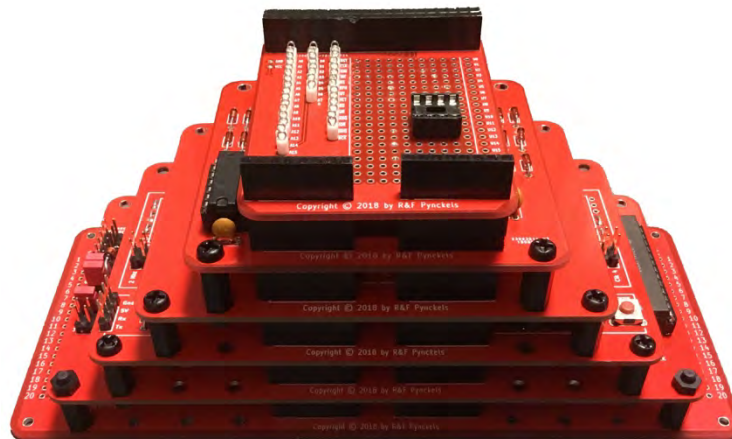


FIGURE 1 - S80 VERSION 1.0

Since the second version of the S80 and of this document are an immediate consequence of the first versions, we will build upon the information mentioned in the first document without repeating all of it. So, this document should be considered by the reader as an update. However, since the choice of an architecture is important when designing a computer, the chapter on choosing an architecture is repeated below.

We had a “*do it yourself kit*” (see *Figure 2*) for the first version. We are still in consideration if we will also constitute such a kit for the second version. This will for the most part depend on the questions we get in that direction.

We will explain why we changed some things, how we changed them, and what the resulting hard- and software is.

The “why” is still the same:

- There is an **increasing interest in retro technology**.
- The Internet of Things (IOT) is built with new technology, but the principles are still the same as those that were valid in the days of the first computers: little computers with limited power and band width that are majorly used for a single task at a time, are used to **extend ones computing environment**.
- **Be an education platform** that clarifies the end-to-end construction of a simple computer.

Where the goal of version 1 was more to split up the functionalities so to give each functionality its own board, this has changed in the second version. With version 2, we opt to split up the computer in its major architectural components. Small functionalities, like a reset circuit and the power input are put on the board that seems the most appropriate to host them.

Everything in this document is public domain under the MIT license and can be used to develop or extend the readers projects or ideas. We would consider it an honor if people would take this design and construct other modules for it. And we're ready to give any possible support if they want to do so.

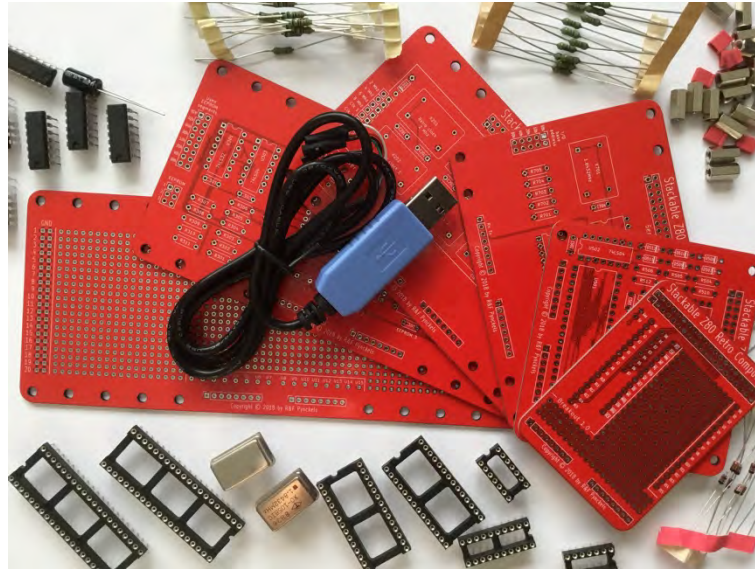


FIGURE 2 - S80 VERSION 1.0 - DO IT YOURSELF KIT

The fact that we put a copyright notice on most documents and deliverables does not mean that we want to protect the below ideas. On the contrary, it gives us the possibility to keep this entire biotope under the MIT and the open hardware license, without someone trying to make it proprietary.

We are also looking for people who want to extend this open idea/source/hardware platform:

- Change the designs and make them faster, compacter, more elegant, more comprehensive for beginners or more challenging for experts, make them more suited for educational purposes.
- Make additional hardware/boards (CP/M 4 MB memory board, Internet of things boards, parallel I/O board, board that can be used for educational purposes, ...).
- Adapt or write software for the S80. In the first place, I think a compact monitor program would be nice. And of course, the Microsoft NASCOM ROM BASIC Ver 4.7. Maybe also some Internet of Things programs for the hardware mentioned above.
- Write educational documentation about the S80 so to permit the S80 environment to be used in schools to explain analog and digital electronics, or to be used by people interested in getting a grasp of electronics and computers.
- Host an open source version management system (independent of establishment companies) or manage projects in such a system, where people can place their board designs, software developments, documentations, ... and work together on them.
- Propose other ways to make the S80 useful for people and/or society.
- Host a YouTube channel about the S80 and projects using it (tutorials on Z80, assembler, digital electronics, Internet of Things, ...).

- Promote the S80 environment as a fun and instructive environment that's entirely open hardware/software.

For the rest of this document, we will handle the three major parts of the second version of the S80: the processor board, the serial board and the memory board. The processor board and the serial board are already treated in the S80 version 1 document (see link on page 1). There is a minor adjustment on the processor board to accommodate the memory board. The serial board is not changed at all.

The reader should note that the architecture also mentions a “*laser board*”. Although the actual hardware- and software sources are already included in the sources zip file of the first version of the S80, and will be included in the second version too, this is an announcement of an upcoming document. Creating an extension board for the S80 is a subject on its own and will be treated in an upcoming document.

1 Architecture

1.1 The Princeton architecture

The **Princeton architecture** (also called the Von Neuman model, see *Figure 3*) is first described in the 1945 document **First Draft of a Report on the EDVAC**. It contains the first published description of the logical design of a computer using the stored-program concept. The document describes a design architecture for an electronic digital computer with as components:

- A **processing unit** that contains an arithmetic/logic unit and processor registers
- A **control unit** that contains an instruction register and program counter
- **Memory** that stores data and instructions
- External mass storage
- **Input and output** mechanisms

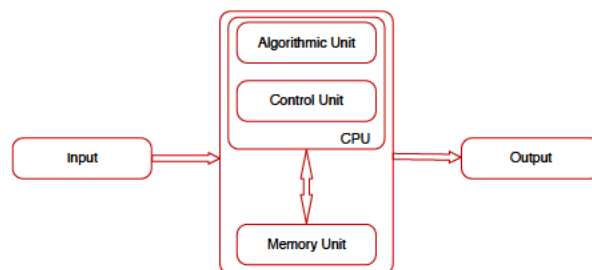


FIGURE 3 - ARCHITECTURE - PRINCETON

1.2 The Harvard architecture

The **Harvard architecture** (see *Figure 4*) is a computer architecture with physically separate storage and signal pathways for instructions and data. The name originates from the **Harvard Mark I** computer. These early machines had data storage entirely contained within the central processing unit and provided no access to the instruction storage as data.

Today, most processors implement such separate signal pathways for performance reasons, but actually implement a modified Harvard architecture, so they can support tasks like loading a program from disk storage as data and then executing it.

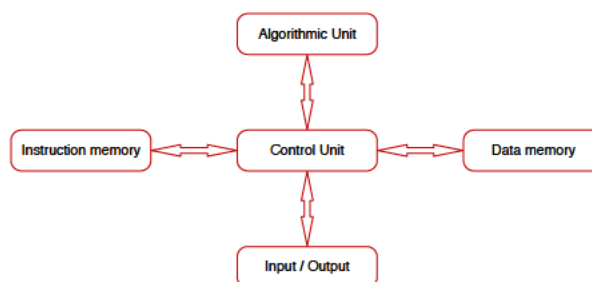


FIGURE 4 - ARCHITECTURE - HARVARD

1.3 The S80 version 2.0 architecture

The **architecture of the S80** is a combination and extension of the ideas of both the above models. We have chosen for an architecture with two busses. A **primary bus** with 40 lines (A, C, D, P) and a **secondary bus** with 16 lines (U) for later expansions.

- **A[0..15]** : address lines
- **D[0..7]** : data lines
- **C[0..13]** : control lines
- **P[0, 1]** : power lines (Gnd, Vcc)
- **U[0..15]** : expansion (user) lines

The number of address lines (16), data lines (8) and control lines (14) is an immediate consequence of the use of the Z80 processor and the number of lines it can manage. The number of expansion lines is an arbitrary choice, induced by design and construction choices.

The above is the same for version 1.0 as for version 2.0, making both versions 100% bus-compatible and almost 100% hardware- and software compatible.

A top-level schematic of the architecture of version 2.0 can be found in Figure 5. The bus lines (address, data, control, power, user) are further clarified in this top-level schematic.

The **processor**, the **memory** and the **serial** (input/output) components are comparable with the components from The Princeton architecture. However, to make a functioning computer, other functionalities (the **clock**, the **reset circuit** and the **power circuit**) must be present. These “added electronics are added to the new processor board.

The **small breakout** board and the **large breakout** board of the version 1.0 are compatible with the below discussed version 2.0 boards. These components permit the user to make test-components that can access the primary bus as well as the secondary bus.

The chosen architecture permits to add an unlimited number of primary/secondary bus-compatible components to be added, that can interact with already existing components by means of the bus lines. The only limitation is the inspiration of the component developer.

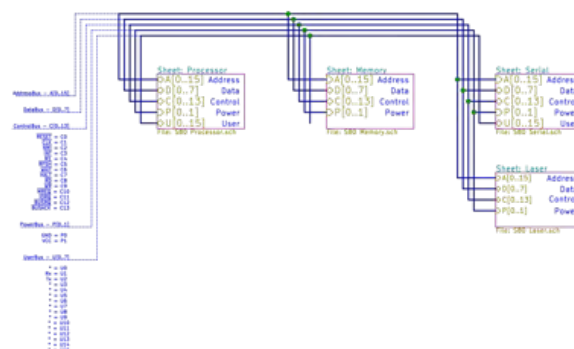


FIGURE 5 - ARCHITECTURE - S80 VERSION 2.0

2 Design choices and specifications

We are still using KiCad as design tool. We're not saying that it is the best tool, but it's good and it's free.

2.1 Design guidelines

There will be **6 different board sizes** (see *Figure 6*). The boards will be stackable without limit. The boards will form (eventually supported by spacers) a pyramid shape (see *Figure 1*). But, since the stacking of the boards is not limited, they can also form a diablo shape, a cube shape, a beam shape, etc. In fact, it should be elegant if not put in a case, and it should be easy to put it as compact as possible in a case for production purposes.

All components will be **through hole** except if the required functionality does not exist in a DIP package. But through-hole or not, all components will be at the top side of the boards, in order to guarantee a uniform and elegant view.

Boards created by third parties can have any shape, but it would be nice if they would also consider the default measures and the default side of the board to put their components on.

The printed circuit boards will be made available in **2 colors**: green and red. The red boards are the final builds of a version that are ready for “production”. The green boards are test-boards and will be marked as such. Both the green and the red boards will have white silk screens at both sides of the boards.

To guarantee that the designs stay under the MIT license, **a copyright will be mentioned** on the boards, so to give the designer the power to keep all his ideas in the public domain.

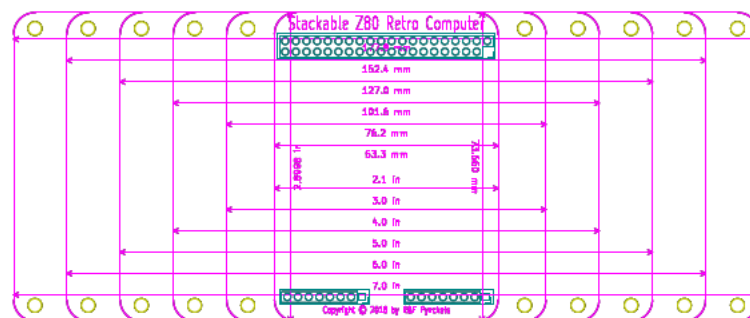


FIGURE 6 - DESIGN SPECIFICATIONS - SIZES

The busses of the design will be implemented with **stacking headers**. The sizes of the stackable headers (A, D, C) and (U) and distance between the stackable headers (U0..U7) and (U8..U15) must permit to put a normal breadboard between two layers of the S80 (see *Figure 7*) as to be able to do fast temporary tests that are part breadboard and part S80. The vertical distance between any two consecutive layers of the S80 must also permit to put a standard breadboard between them.



FIGURE 7 - DESIGN SPECIFICATIONS - HEADERS AND BREADBOARD

2.2 PCB design rules

The used PCB design rules are:

- 6 mil minimum **trace width**
- 6 mil minimum **trace spacing**
- 10 mil minimum **drill size**
- 5 mil **annular ring**
- **PCB thickness** of 1.6mm (0.063")
- 1oz **copper** on both sides (1.4 mil, 35μm)

The **spacer holes** must be copper surrounded and connected to the GND plane. This will guarantee that metal spacers, if used, can connect the ground planes of all boards. This is also guaranteed by the GND line of the primary bus, but the metal spacers can provide more capacitance and a Faraday effect between the boards.

3 Processor 2.0 board

3.1 Processor 2.0 - Description

The processor functionality is the same as discussed in *Appendix 1* of the *S80 version 1 document* (see link on page 1).

The second version of the processor board is smaller than its predecessor, it does not contain protection circuitry, it contains pull-up on the incoming bus lines, and it contains the clock/power/reset functionality. These changes make the board smaller and more stand-alone. In fact, the only thing missing to have a functioning Internet of Things (IoT) computer would be some ROM and eventually some RAM memory. For IoT, all I/O can be sent to the user bus or via either bus to another S80 board.

Since the input connector also contains the Rx and Tx signals, besides the GND and VCC signals, the processor board 2.0 must pass the Rx and Tx signals via the user bus to the serial board. In order to be **compatible with serial board 2.0**, the Rx signal must be passed through U1 and the Tx signal must be passed through U9. To be **compatible with serial board 1.0**, the Rx signal must be passed to U1 and the Tx signal must be passed to U2. In the latter case, a jumper cable must patch user bus U9 to user bus U2.

3.2 Processor 2.0 - Technical specifications

Table 1 gives an overview of the information needed to integrate the processor board in the S80 stack.


Author/Creator	Robin & Filip Pynckels
License	MIT License (open hardware and software)
Remark	This board is tested with Serial board 1.0 and Memory board 1.0
Status	Production boards are ready. Do it yourself kits are investigated.
PCB Manufacturing	 batch numbers: 2396351A-Y18-190119
Dimensions (W x H x D)	53.3mm (2.1") x 73.7mm (2.9") x 1.6mm (0.063")
Outgoing bus signals	A[0..15] (not buffered, can go in high impedance) C[4, 7..11] (not buffered, can go in high impedance) C5, C13 (not buffered, no high-impedance mode)
Incoming bus signals	C[0..3, 6, 12] (not clamped, accept high impedance mode) P[0, 1] (no protection build in)
Bidirectional bus signals	D[0..7] (not buffered)
User bus signals	U1 (Rx outgoing on U-bus & incoming on TTL conn pin 3) U9 (Tx incoming on U-bus & outgoing on TTL conn pin 4)
Reserved Memory addresses	-
Reserved IO addresses	-

TABLE 1 - PROCESSOR 2.0 - TECHNICAL SPECIFICATIONS

3.3 Processor 2.0 - Functional schematics

To give an overview, we join a rescaled version of the functional schematics of the processor circuit in *Figure 8*.

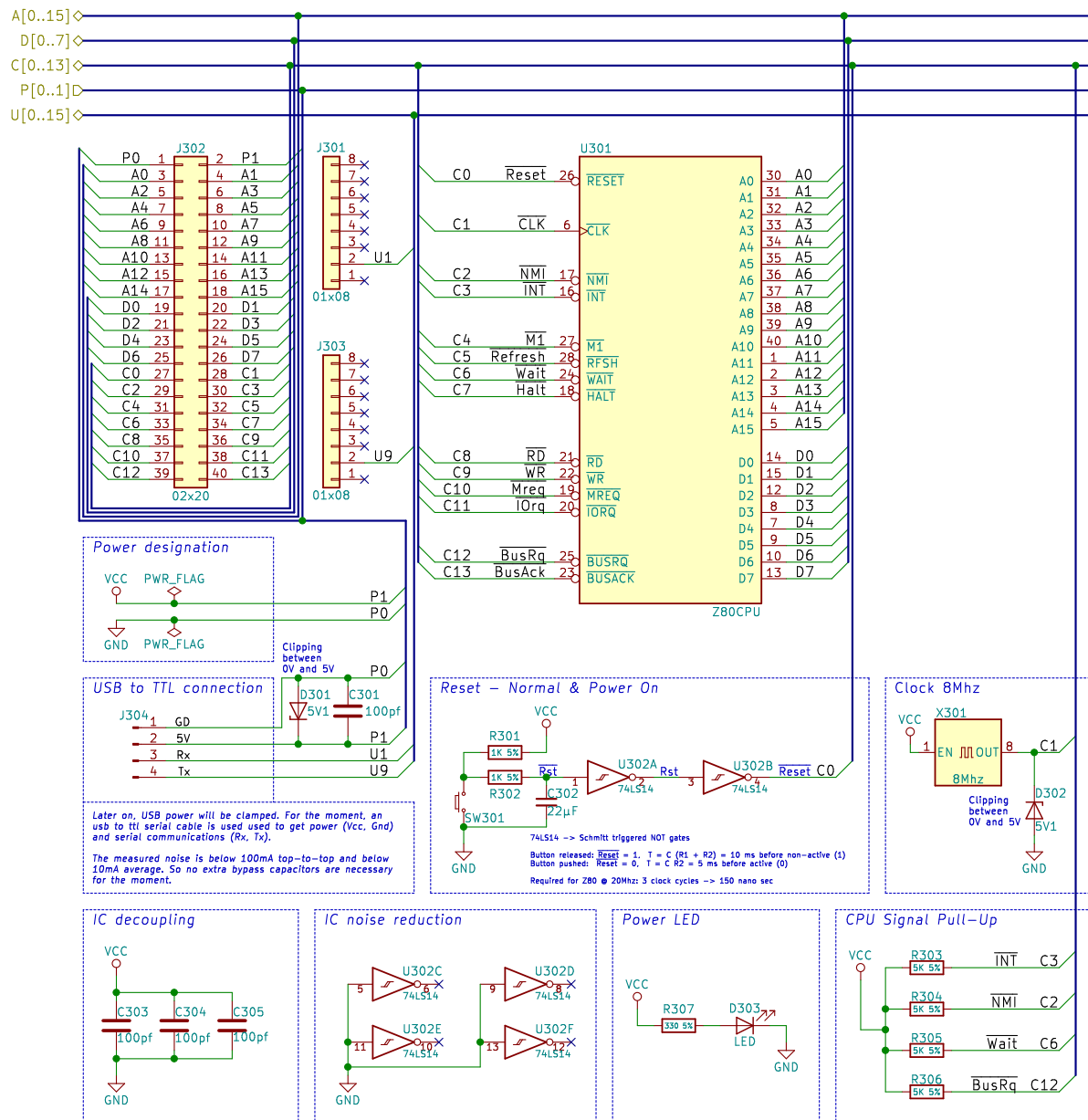


FIGURE 8 - PROCESSOR 2.0 - FUNCTIONAL SCHEMATICS

3.4 Processor 2.0 - PCB design

To give an overview, we join a rescaled version of the PCB design of the processor board in *Figure 9* (left image), and an impression of the non-populated and the populated final PCB in *Figure 9* (middle and right image). Please note that the below figures don't have the exact dimensions of the real PCB.

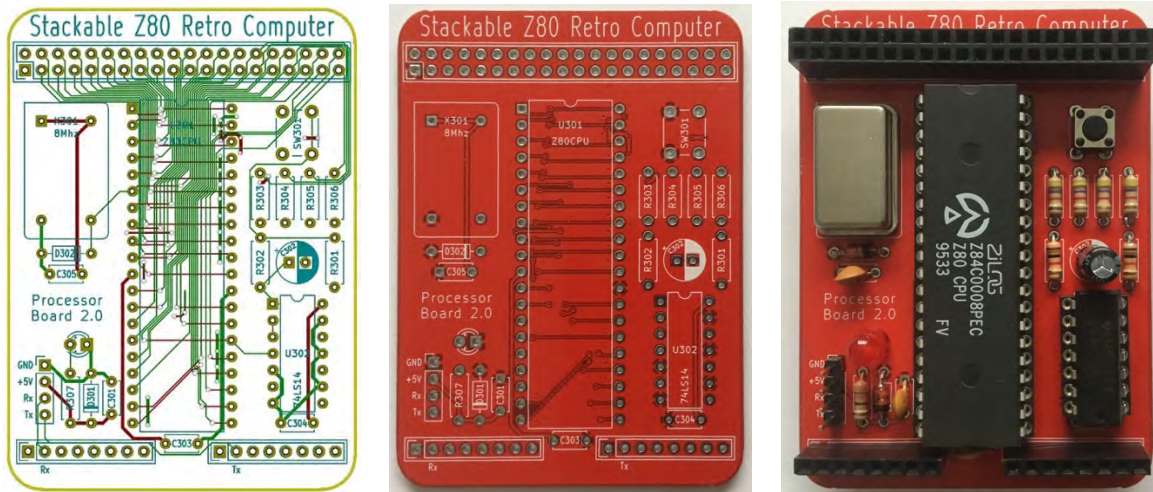


FIGURE 9 - PROCESSOR 2.0 - PCB

3.5 Processor 2.0 - Correction for the memory 2.0 board

We have done substantial testing to be sure that signals (in this case the reset signals) are crisp and that temperatures don't rise during an hour of intensive use of the board. We have also tested this board with the memory 1.0 board and concluded a 100% compatibility. However, for the memory 2.0 board, we need a little more startup time before the reset signal becomes inactive (high). This extra startup time will permit the memory 2.0 board to initialize itself.

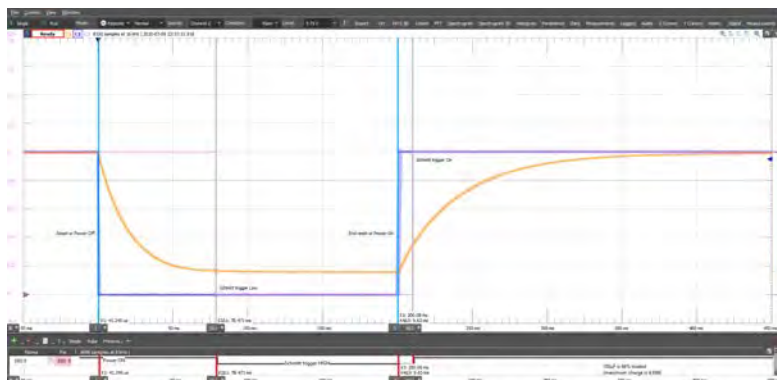


FIGURE 10 - EMULATION - RESET SIGNAL (22 μ F, 1K Ω , 1K Ω)

Figure 10 shows the emulated timing measurements without the proposed changes:

- State of **SW301** (closed = logic 1, open = logic 0) in blue.
- Signal **/RST** in orange.
- Signal **/RESET** in pink.

Note that it takes the double Schmitt trigger around 78ms after the reset button is pushed to trigger down. The reason is that the 74LS14 Schmitt triggers go low at 0.8 volt of their input signal (orange signal /RST). On the other side, it only takes the double Schmitt triggers 9.63ms to trigger up, since the 74LS14 Schmitt triggers go high at 1.6 volt of their input signal (orange signal /RST). The triggering voltages can be found in the 74LS14 datasheet.

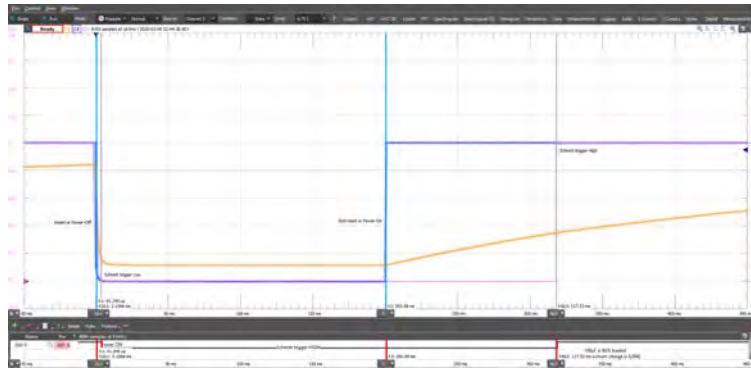


FIGURE 11 - EMULATION - RESET SIGNAL (100 μ F, 4.7K Ω , 100 Ω)

The solution to get a **larger up-triggering time interval** and to get a smaller down-triggering time interval (which is of no real concern for the memory board) is to **replace the 22 μ F capacitor C302 and the 1K Ω resistors R301 and R302**. For educational purposes, we have tested two slightly different configurations to show the reader the impact of a small change in the discharging resistors. The configurations are:

- C302=100 μ F, R301=4.7K Ω , R302= **100 Ω** (Figure 11)
- C302=100 μ F, R301=4.7K Ω , R302= **0 Ω** (Figure 12)

It should be clear, from Figure 11 and from Figure 12 that as well the down-triggering time interval as the up-triggering time interval are around **2.2ms faster** when coupling the capacitor C302 to the switch without an intermediate resistor R302. In both cases, this has **no significant impact** on the functioning of the S80. Since the values in Figure 12 are ever so slightly better and save us an extra resistor (in favor of a bridge wire), **we will opt for C302=100 μ F, R301=4.7K Ω , R302=0 Ω** .

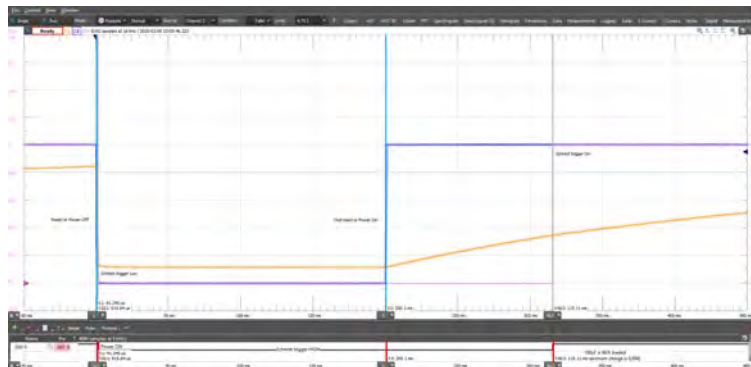


FIGURE 12 - EMULATION - RESET SIGNAL (100 μ F, 4.7K Ω , 0 Ω)

It should be clear that, if we want to use the Processor board for an IoT configuration with a memory board without memory mapping, there is no need for the proposed changes.

To have **less different components** in the *Do it yourself kit*, we also **replace R303, R304 R305 and R306 with 4.7K resistors**. This also gives the board a more uniform appearance.

After replacing the necessary components on the processor board, a real reset and power-on test were done. The results can be found in Figure 13 (*Reset test*) and in Figure 14 (*Power-on test*)



FIGURE 13 - PROCESSOR 2.0 – RESET SIGNAL (100 μ F, 4.7K Ω , 0 Ω)

The reset test shows a down-triggering time of 122 μ s. The up-triggering time is 152ms. Both numbers are better than the simulated times in the sense that the down-triggering is faster and that the up-triggering time is slower. Goal achieved!

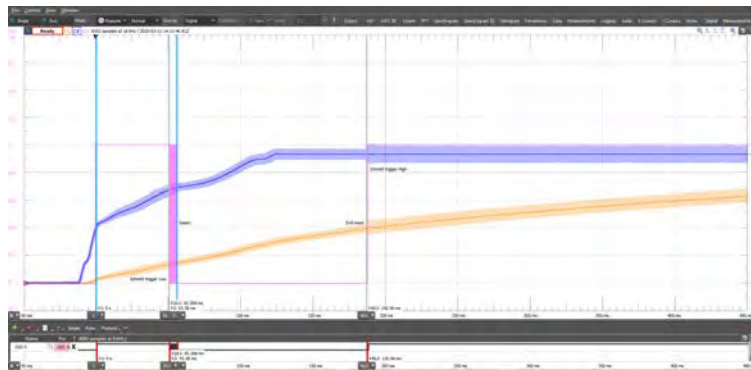


FIGURE 14 - PROCESSOR 2.0 – POWER-ON SIGNAL (100 μ F, 4.7K Ω , 0 Ω)

The power-on test shows results that deserve some explanation. We see the following signals:

- State of P1 (5V power line) in blue.
- Signal /RST in orange.
- Signal /RESET in pink.

It takes some time to get the power line to 5V. Some components start pulling power, resulting in a non-linear power increase. During this time, the 74LS14 IC's react a bit strange to the lack of sufficient power. First, the /RESET signal stays low until the power line reaches 2V. Then the /RESET line goes high until the power line reaches 3.5V. During the next 5ms, we see kind of bouncing signals, meaning that the 74LS14 can't make up its mind. Finally, the /RESET line goes down when the power line goes towards 5V. **From that moment on (56ms after power-on), the power board goes in reset state as it should.**

The /RESET signal is passed to all S80 boards to take the necessary action. Both the serial board and the memory board effectively react on a reset situation. The serial board initializes its 16C550 IC and the memory board reinitializes its memory mapping logic. And the get, during power-on, 131ms to do so. During a real reset phase (reset button pushed), they even get 152ms.

4 Serial 2.0 board

4.1 Serial 2.0 - Description

Just like for the processor board, we want to make a more compact board with the same functionalities.


Since the processor board's input connector also contains the Rx and Tx signals, besides the GND and VCC signals, the processor board 2.0 passes the Rx and Tx signals via the user bus to the serial board. In order to be compatible with processor board 2.0, the Rx signal must be passed through U1 and the Tx signal must be passed through U9.

Multiple serial boards 2.0 can be used in the same "hardware stack", since serial board 2.0 permits to choose which user bus line is used for the Rx signal (U0...U7) and which user bus line is used for the Tx signal (U8...U15). So, if each serial board is used for as well Rx as Tx, 8 serial boards can be used together. If each serial board is used only for Rx or Tx, 16 serial boards can be used together (8 for Rx signals, 8 for Tx signals).

Please note that one serial board 1.0 and multiple serial boards 2.0 (6 for Rx signals, 8 for Tx signals) can be used together. Since serial board 1.0 uses U2 (Rx) and U3 (Tx), one must be attentive that none of the serial boards 2.0 use the U2 nor U3 for Rx signals, since this would come into conflict with the used serial board 1.0

4.2 Serial 2.0 - Technical specifications

Table 2 gives an overview of most of the information needed to integrate the processor board in the S80 stack.

Author/Creator	Robin & Filip Pynckels
License	MIT License (open hardware and software)
Remark	The difference between the version 1 and version 2 board are on the one side its dimensions and on the other side the fact that a pin can be chosen on the User data bus for the Rx and the Tx signal. So, multiple serial boards version 2 can be used with different I/O addresses. This board is tested with Processor board 2.0, Memory board 1.0, in combination with one Serial board 1.0 and in combination with one Serial board 1.0 and multiple Serial boards 2.0.
Status	Production boards are ready. Do it yourself kits are investigated.
PCB Manufacturing	 batch numbers: 2396351A-Y19-190119
Dimensions (W x H x D)	76.2mm (3.0") x 73.7mm (2.9") x 1.6mm (0.063")
Outgoing bus signals	C3
Incoming bus signals	A[0..7] (checked on address range set by dipswitches) C0, C4, C8, C9, C11 (no protection build in) P[0, 1] (no protection build in)
Bidirectional bus signals	D[0..7] (buffered)
User bus signals	Rx on U[0..7] (outgoing on U-bus & TTL conn pin 3)

4.4 Serial 2.0 - PCB design

To give an overview, we join a rescaled version of the PCB design of the serial board in *Figure 16 (left image)* and an impression of the non-populated and the populated final PCB in *Figure 16 (center and right image)*. Please note that the below images don't have the exact dimensions of the real PCB's

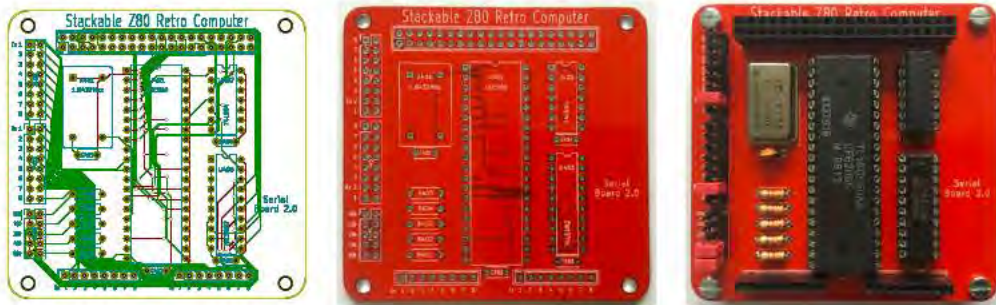


FIGURE 16 - SERIAL 2.0 - PCB

For clarity: the **upper 2x8 pin connector** with 1 jumper defines on which user bus line (U0..U7) the **Rx signal** is supposed to be. The **lower 2x8 pin connector** with 1 jumper defines on which user bus line (U8..U15) the **Tx signal** is supposed to be. The **2x6 pin connector** defines the **base I/O address** that the processor must use to address the serial processor on this board.

Below, the temperature reading of the serial 2.0 board after running the Towers of Hanoi program for 60 minutes. Which, in the meantime, proves that this board is software compatible with the version 1.0 hardware and software (the memory board 1.0 was used and no recompile was done to run the program).

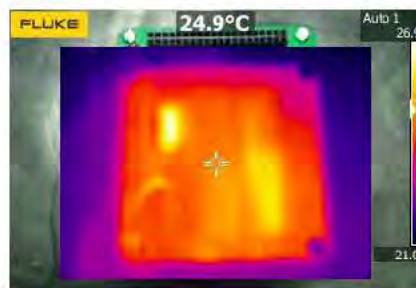


FIGURE 17 - SERIAL 2.0 - TEMPERATURE TEST

5 Memory 2.0 board

5.1 Memory 2.0 - Description

We saw the memory 2.0 board as a challenge, since we wanted to redesign the entire memory setup. More specific, we wanted to make a design that permits to make a PCB supporting up to 4Mb of bank-switched memory. Each memory IC's can be designed as well as an EEPROM or as a RAM of 256Kb. Note that, if the EEPROM and the RAM IC's you want to use have a different pin-out, you must decide during the design phase of this board where you will place EEPROM's or RAM's. This means that we can design a combination with only 256Kb EEPROM, but as well a combination with 4Mb of EEPROM and no RAM (for whatever reason we would have). To run CP/M, a memory setup will probably be something like 512Kb EEPROM and 1.5Mb of RAM, with the entire CP/M in EEPROM. Unless if you plan to load the CP/M software from a storage device, in which case you will probably have to make different design choices.

The bank switching logic permits 4 banks of 16Kb that can be mapped on the entire 4Mb of memory in steps of 16Kb. The way this is done is the following:

5.1.1 Bank switching - Set mapping

The mapping is done by using an OUT instruction to an I/O address range that is set on the memory card. The data written to an I/O address has the following format (in bits): cccmmmmm, where ccc (the 3 most significant bits) contain the memory chip number (0..7) and where mmmmm (the 5 least significant bits) contain the address lines 14-18 (the highest address lines) of the chosen memory chip.

Let's suppose that we have set the jumpers to use address range 0xA0..0xA3. Then:

- writing 0x01 to the address 0xA0 sets the mapping of the first memory bank (low 16Kb of the virtual address) to the second 16Kb of the first memory chip.
- writing 0xFF to address 0xA3 sets the mapping of the fourth memory bank (high 16Kb of the virtual address) to the last 16Kb of the last memory chip.

5.1.2 Bank switching - Get data from virtual address

Letting the Z80 processor read from the real memory is transparent for the Z80. The address bits 0..13 of the Z80 address constitute the address within the 16Kb bank. The address bits 14..15 constitute the bank number to use for memory mapping.

In the examples from *paragraph 5.1.1*, if the mapping registers have not been changed:

- moving a byte to virtual (Z80) address 0x0000 means that, in real memory, we move the byte to the first byte of the second 16Kb memory block of the first memory chip.
- reading a byte from virtual (Z80) address 0xC000 means that we read the first byte of the last 16Kb block of the last memory chip.

5.2 Memory 2.0 - Find the bug...

So far so good. However, when using the test setup in *Figure 18* (**First test PCB**), it was obvious that things didn't work out the way they were intended to. In fact, the S80 didn't start up at all. When changing the board with the memory board 1.0, everything was fine. Changing to the memory 2.0 board again gave the same error. So, the error was reproduceable, but neither the schematic nor the VHDL simulation that was done when designing this test board seemed to contain an error.

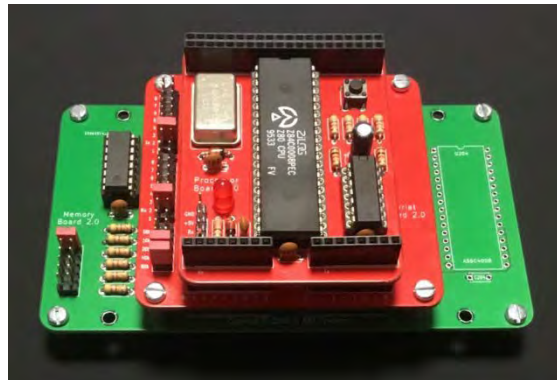


FIGURE 18 - MEMORY 2.0 - FIRST TEST BOARD

Digging into the signals generated by the processor 2.0 board didn't bring us further, but when we took a look at the real addresses generated by the addressing logic, we came to the conclusion that the initial address was not pointing to address 0. Backtracing the problem made clear that the initialization of the SN74LS670 (the 4 x 4bit RAM) was different in theory (datasheet) than in practice. More specific, at initial power-up, the RAM registers seem to contain the value 0b1111 for the full 100% of the times we have done the tests (several 100ths of times).

Plugging the SN74LS670 (the 4 x 4bit RAM) in a breadboard (see *Figure 19*) to test it at startup confirmed the above analysis.

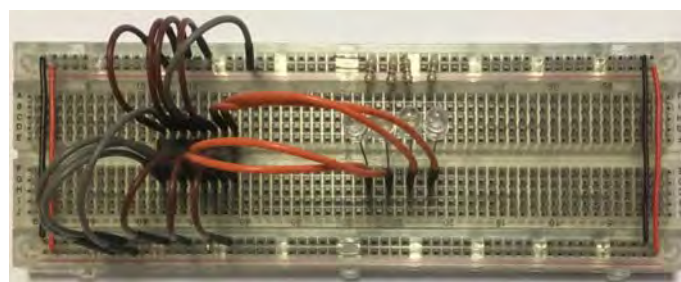


FIGURE 19 - BREADBOARD TEST OF THE SN74LS670 COMPONENT

However, we need the first register to contain 0b0000 at startup, so the computers first 16K of memory addresses are mapped to the first 16K real memory addresses (more specific to the start of the EEPROM memory). Unfortunately, a quick patch was not feasible, since the addressing logic had to be entirely redesigned. And that was that...

Theoretically: *“Problem solved”*.

Practically: *“Oh god, why art thou hating me...”*

5.3 Memory 2.0 - Technical specifications

Table 3 gives an overview of most of the needed specifications to integrate the memory board in the S80 stack. It also constitutes the functional analysis of the memory board to design.


Author/Creator	Robin & Filip Pynckels
License	MIT License (open hardware and software)
Status	Under development
PCB Manufacturing	 batch numbers: -
Dimensions (W x H x D)	127.0mm (5.0") x 73.7mm (2.9") x 1.6mm (0.063")
Outgoing bus signals	-
Incoming bus signals	A[0..15] (non-clamped, accepts high impedance mode if C[0, 8, 9, 10, 11] are not floating) C[0, 8, 9, 10, 11] (non-clamped, does not accept high impedance) P[0, 1] (no protection build in)
Bidirectional bus signals	D[0..7] (non-buffered, non-clamped, can go in high impedance, accepts high impedance mode if C[8, 9, 10] are not floating)
User bus signals	-
Reserved Memory addresses	0x0000 :: 0xFFFF
Reserved IO addresses	Address range of 4 addresses to write to the 4 memory mapping registers. The base address of the address range can be set with jumpers on the board. Advised address range is 0x00..0x03. These addresses are used by the default bootloader.
Initial state	The first memory address mapping register must be initialized to 0x00 so that the S80 can start on real memory address 0x000000.

TABLE 3 - MEMORY 2.0 - TECHNICAL SPECIFICATIONS

5.4 Memory 2.0 - Functional schematics

To give an overview, we join a functional schematic of the memory circuit in *Figure 20*.

Note that the **addressing logic is a new design compared to the schematics of the first test board** (see *Figure 18*). The functional schematics of this first test board will not be joined to this document, since they are erroneous due to the deviant behavior of the SN74LS670 chips.

Note also that **the redesigned memory logic needs a little more startup time** before the reset signal becomes inactive (high). This extra startup time permits the memory 2.0 board to initialize itself. More specific, to initialize the two SN74LS670 (the 4 x 4bit RAM) chips. **The entire discussion on providing more startup time between reset switch release (equivalent to power-on) and the processor starting to execute can be found in Paragraph 3.5.** More specific, *Figure 10*, *Figure 11* and *Figure 12* can be educational.

To be sure that this time, the functional schematics are also behaving correct during startup, **an extended functional testing cycle will be run through.**

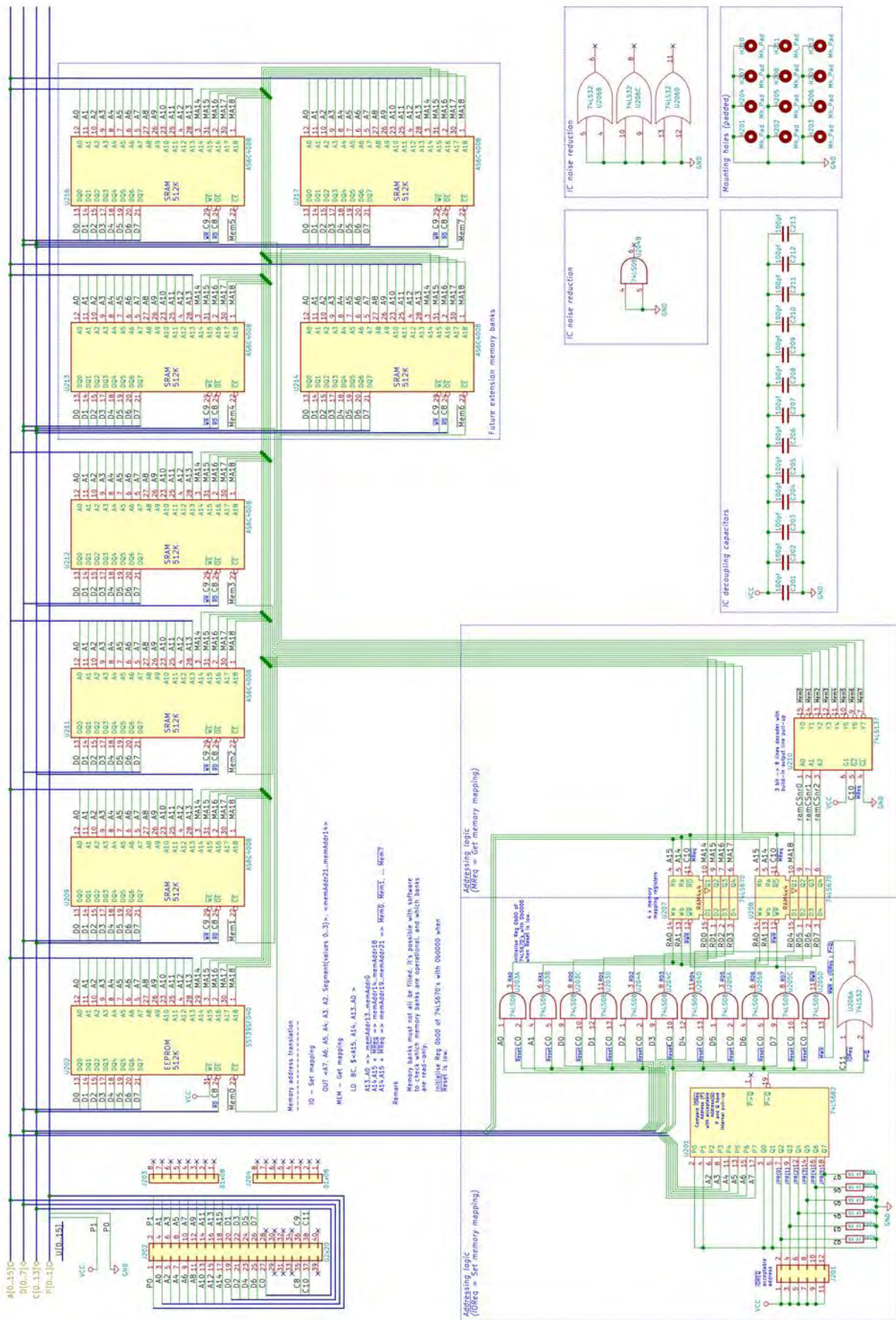


FIGURE 20 - MEMORY 2.0 - FUNCTIONAL SCHEMATICS

5.5 Memory 2.0 - Functional schematics - Testing

5.5.1 Functional schematics test - Simulation

Taking into account the results of *Paragraph 3.5*, we are now ready to **develop a software simulation of the memory addressing logic** in VHDL using ModelSim.

The overall structure of the simulation is shown in the RTL diagram of *Figure 21*.

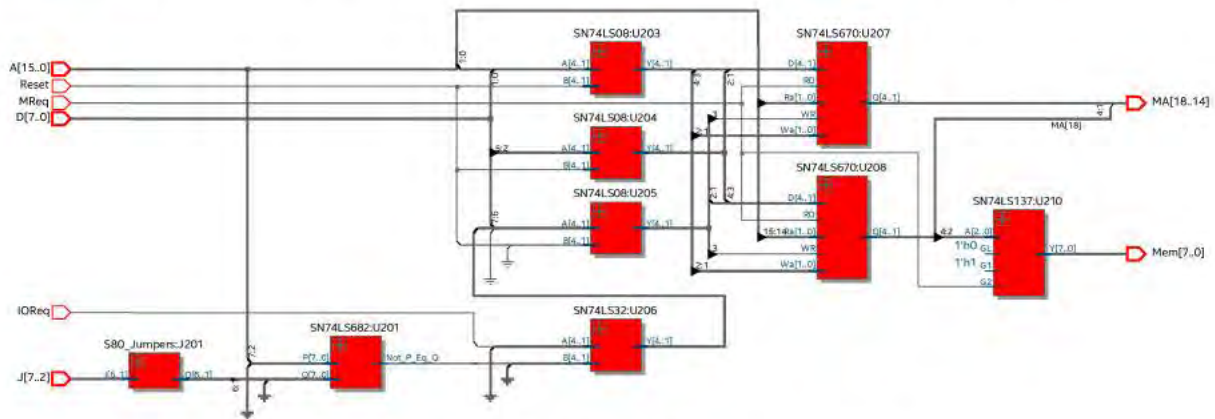


FIGURE 21 - MEMORY 2.0 - VHDL SIMULATION SCHEMATIC

All VHDL files, inclusive the test benches for each component and the ModelSim script files to launch the different simulations and tests are included in the resources zip file. The URL where this zip file can be found is mentioned on the title page of this document, at the bottom of the page (*Sources*). However, for illustrative reasons, the VHDL code for the above memory addressing logic simulation model is shown in *Appendix 1*, together with a schematic of the addressing logic circuit.

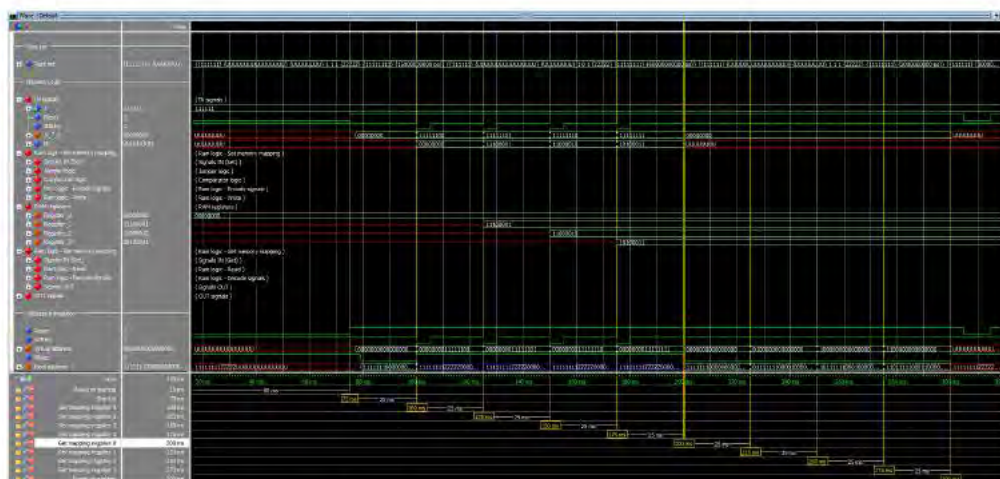


FIGURE 22 - VHDL TEST OF THE MEMORY ADDRESSING LOGIC

The **resulting tests (numeric, waves)** are **conclusive** in the sense that the redesigned memory addressing logic seems to do what it should do, even at startup, where we simulate a reset signal bump up to the “high” state, which isn’t even the case with the real reset signal. The resulting waves are shown in *Figure 22*, and in full-size in *Appendix 2*.

5.5.2 Functional schematics test - Breadboard

Since the **VHDL tests show positive results**, we can now build the **memory addressing logic on a breadboard** (see *Figure 23*) to assess how it behaves during startup and during a user-controlled reset. This also seems to give **results that are coherent** with the VHDL simulation, even for the indicative timing tests.

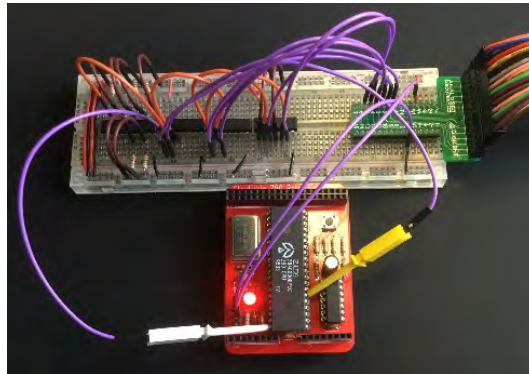


FIGURE 23 - BREADBOARD TEST OF THE MEMORY ADDRESSING LOGIC

5.5.3 Functional schematics test - Prototype

The next phase of the development of the memory 2.0 board will be the **construction of a wire-wrapped hardware prototype board** that will not only contain the addressing logic, but also an EEPROM and a RAM chip. This will permit us to assess the full functionality of the memory 2.0 board before ordering test PCB's.

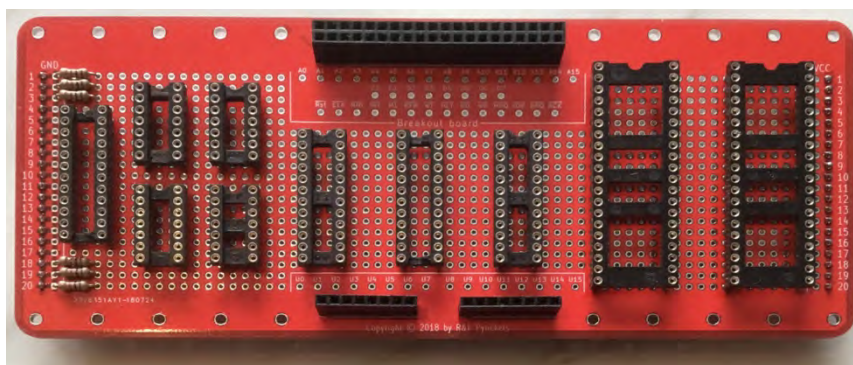


FIGURE 24 - FRONT SIDE OF THE NON-POPULATED PROTOTYPE BOARD

We chose to use the 7-inch breakout board we have developed for the S80 version 1. Information about this board (URL) can be found on the title page of this document, at the bottom of the page (*Documents*).

First, we have prepared the breakout board to be suitable for the intended test, by soldering the S80 compatible headers on it, by soldering 20x1 headers to connect IC's to respectively the Gnd and the Vcc signal and by plugging in wire wrapping sockets for all IC's. See *Figure 24* for a front side picture and *Figure 25* for a back-side picture of the prepared board.

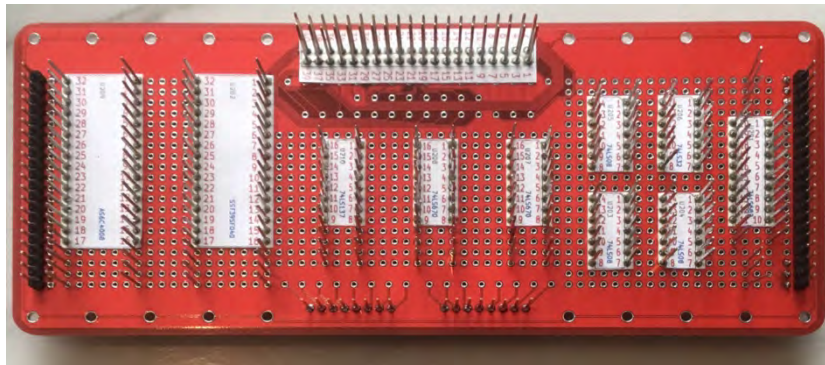


FIGURE 25 - BACKSIDE OF THE NON-WIRED PROTOTYPE BOARD

When using the wire wrapping technique, one of the problems is that one tends to make a lot of errors because one is working on the backside of the board, which makes that **all connections are mirrored left-right**. Connecting the wrong pin (a pin too high or too low) is also something that happens regularly. To prevent this kinds of mistakes, **we have created real-size mirrored pinouts of the IC's**, mentioning their number in the functional schematics, their specifications and their pin numbers. (see *Figure 25*).

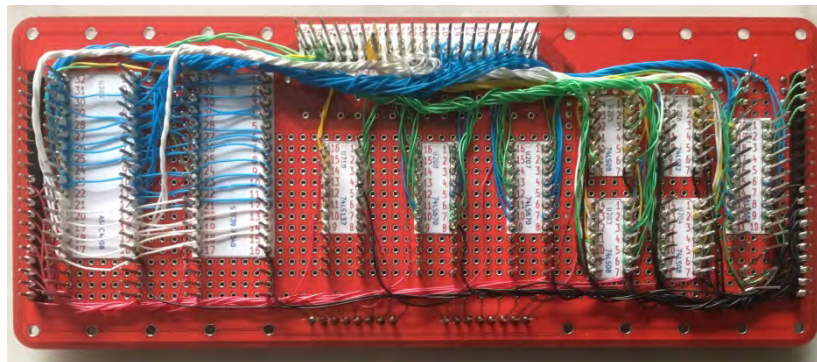


FIGURE 26 - BACKSIDE OF THE WIRED PROTOTYPE BOARD

A wired version, ready for prototype testing of the board is shown in *Figure 26*. Note that we have also **wire-wrapped the pull-down resistors, which is not a good thing to do** if one would use the wire-wrapping for a production board. For test purposes, this can do the trick if the wrapping is checked before doing the test.

A remark on the wiring technique is at its place here. As you can see, we have kind of **regrouped the address lines (blue wires), data lines (white wires), control lines (yellow and green wires) and power lines (red and black wires)**. The reason we've done this is that we want the reader to have an overview of the wiring. However, when measuring the signals on an oscilloscope, we can clearly see an interference pattern. More specific, **when looking at the blue wire bundle (or the white, green or yellow for that matter), we can see "coil" effects**. Indeed, the wires transport data, and hence produce pulses (which is not the case with the red and black wires that are 100% DC). The outer wires transport pulses around a core of inner wires, which creates in fact a disguised but still effective coil. **For our test board and relatively low speed signals, this is feasible**, but even in this case, we are at the limits of what can still give correct results.

The **better way to wire the lines is putting the wires all over the place while still taking care of not pulling wires in the area of regions sensitive to electro-magnetic interference**. These zones can be marked with a marker on the pinout images (see *Figure 26*).

A second remark to make is that we have not used decoupling capacitors near the IC's power inputs. We can get away with this since our DC power lines are very stable (without fluctuations) and are not near the address, data and control wire bundles. But again, **choosing didactic simplicity over the better craftsmanship is a very slippery slope.**

5.5.4 Functional schematics test - Test board

Since the results of the wire-wrapped prototype board seemed satisfying, we made a **second test board**. *Figure 27* shows the test board with connections between one of the 74LS670 IC's and a logic analyzer. This was the setup to make *Figure 28*...*Figure 31*. Note that the processor board and the serial board are stacked under the green memory prototype board.

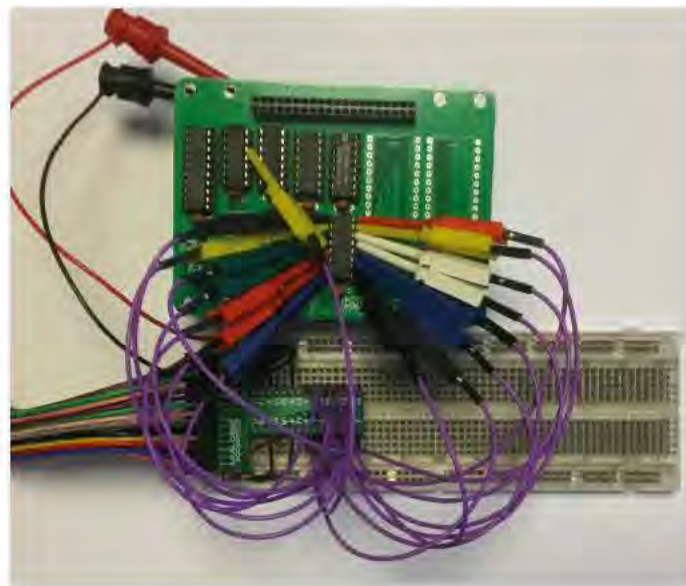


FIGURE 27 - MEMORY 2.0 - TEST BOARD

After some preliminary tests, the CPU board is plugged on the prototype board. Two scenarios must be tested: the startup of the CPU and a regular reset cycle. In both cases, we must assess that the **input 0b0000 (on pins D1..D4)** of **register 0b00 (on pins Wa and Wb)** of **both the 74LS670 IC's** are written by means of an **active low WR (0b0 on pin /WR)**. The best place to measure if this is the case are the lines RA0, RA1, RD0..RD7 and /RWR as they can be found on the functional schematics (see *Figure 20*).

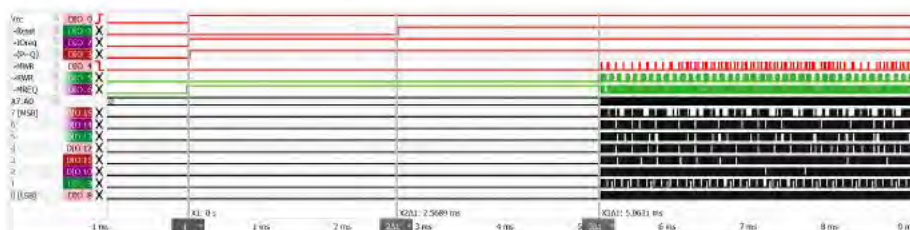


FIGURE 28 - MEMORY 2.0 - ADDRESSING LOGIC ANALYSIS AT POWER-ON

But first, we do a **general assessment of the Vcc, /Reset, /IOreq, /MWR, /RWR and A0..A7 lines** to see if the addressing logic behaves as expected. *Figure 28* shows us the mentioned lines during a power-on phase. *Figure 29* shows the same lines during a normal reset phase.

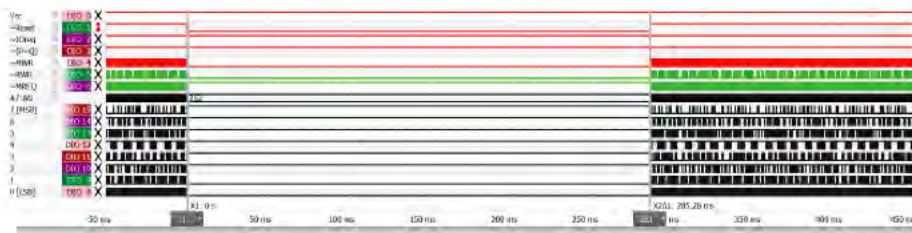


FIGURE 29 - MEMORY 2.0 - ADDRESSING LOGIC ANALYSIS AT RESET

During a reset cycle, the /RWR line stays low during 285ms. Which is a good sign. During a power-on cycle, the /RWR line stays low during 5ms. This can be ok (since, in such a cycle, we do not have to take into account the port delays of the 74LS682 IC, only the port delays of the 74LS32 IC in series with the 74LS08 IC's and the 74LS670 IC's is of importance. We will go further into the timing of the power-on cycle in *Paragraph 5.6*.

The lines that need **further investigation** are the **Vcc line**, the **/RESET line** and the **74LS670 IC's /GW, Wa, Wb, D1..D4** for the register configuration side, and **/GR, Ra, Rb, Q1..Q4** for the register reading side.

Figure 30 shows the **digital line values during the power-on cycle**. A larger image for detailed study can be found in *Appendix 3*. shows the digital line values during a reset cycle. Note that the power-on cycle and a reset cycle are so similar that extra images for studying a reset cycle would be overkill.

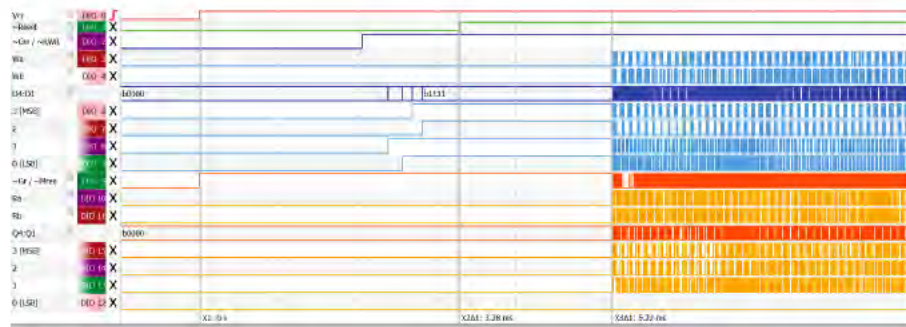


FIGURE 30 - MEMORY 2.0 - 74LS670 LOGIC AT POWER-ON

The lines that still need **further study** are the **Vcc line**, the **/RESET line** and the **74LS670 IC's /GW, /GR, R(Ra, Rb) and Q1..Q4** lines.

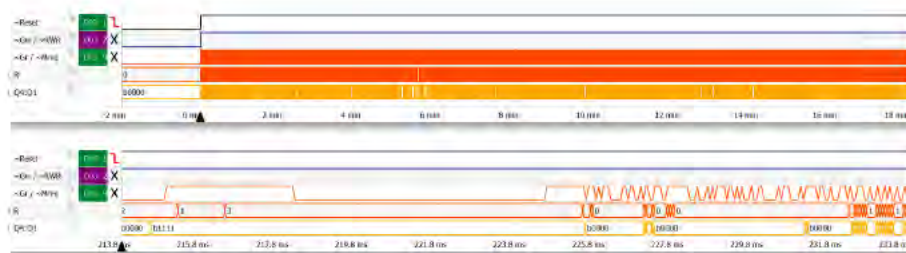


FIGURE 31 - MEMORY 2.0 - 74LS670 LOGIC AT POWER-ON

Figure 31 shows the same lines during a longer time interval, respectively in a shorter more detailed time interval. A larger image of *Figure 31* can be found in *Appendix 3* for detailed study. Functionally, *Figure 31* show that, when the /GR line is down, and the R(Ra,Rb) lines are 0, the first register is initialized with value 0. So, at first glance the test board seems to function.

5.6 Memory 2.0 - Timing study

Besides a functional analysis of the Memory 2.0 board, we also need to assess that the timing of the addressing logic is compatible with the timing of the Z80 processor. We could start dividing the power-on cycle and a reset cycle in their different intervals and throw all kinds of nice (or less nice) formulas on paper. But why would we. *Figure 28...Figure 31* show us the time intervals without calculations. The time intervals seem to be adequate to skip further timing studies.

5.7 Memory 2.0 - Populated board test

As *Figure 27* shows that the only two IC's that need to be added to the test board are an EEPROM and a RAM. The RAM IC can be plugged in as-is. The EEPROM on the other hand needs to be programmed with a small program to give the S80 the chance to start up in an orderly way. The program we will use is:

org	0x0000	
Code_Low:		; Code address 0x0000
nop		
nop		
nop		
jp	Code_High	
defs	0xFA, 0xFF	; Fill with 0xFF bytes
Code_High:		; Code address 0x0100
nop		
nop		
nop		
jp	Code_Low	

5.7.1 Populated board test – Offline controller test

The first series of tests are offline tests. By offline, we mean that the memory board will not be connected to other S80 boards, but only to a test-controller board. As test controller, we use the *Arduino Mega 2560*. This controller was chosen due to the large number of I/O pins it has. *Figure 32* shows the setup. On top of the image, we show the Arduino. In the middle of the image, the Memory 2.0 Board is shown, and at the bottom of the image, the used signal tracer is shown.

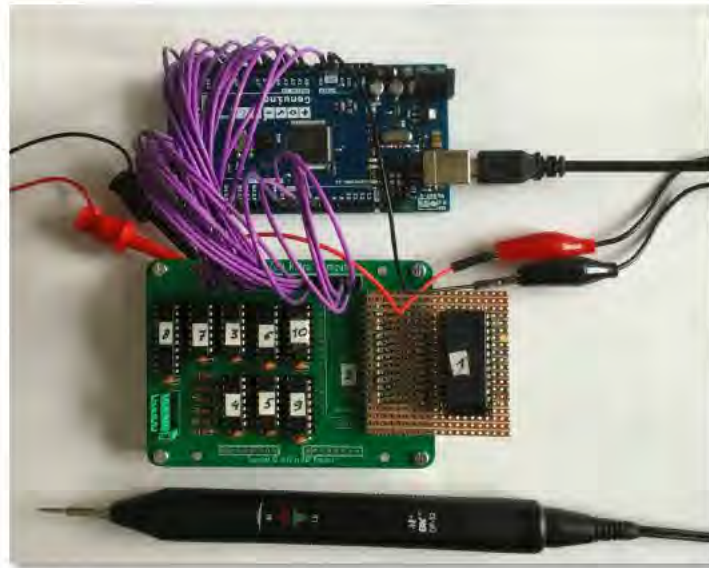


FIGURE 32 - MEMORY 2.0 - OFFLINE CONTROLLER TESTS

The program used in the processor is to be found in the sources zip file referenced at the first page of this document. This program contains **11 different tests and is the implementation of a test matrix** that is shown in in *Appendix 4*. The result of the execution of test 11 is shown below (*Figure 33*). **This test emulates a reset cycle and afterwards, addresses all 2K memory addresses** to show them with their content if that content is not 0xFF. The results are conclusive.

```

COM5 - PuTTY
Memory 514Kb Test Matrix
-----
Test to execute:
  Test[11] - Preparing
  Test[11] - Ready for measuring

0000 : 00
0001 : 00
0002 : 00
0003 : C3
0004 : 00
0005 : 01
0100 : 00
0101 : 00
0102 : 00
0103 : C3
0104 : 00
0105 : 00

Test to execute:

```

FIGURE 33 - MEMORY 2.0 - CONTROLLER TEST RESULTS

5.7.2 Populated board test – Integrated test

So, now comes the final test. The **Memory 2.0 board will be plugged in as-is on an S80 stack composed of an S80 Processor 2.0 board and a Serial 2.0 board**. All that rests is to measure the address- and the data lines to see how they behave. In theory, everything can be measured with mini grabber clips plugged onto the Z80 processor IC.

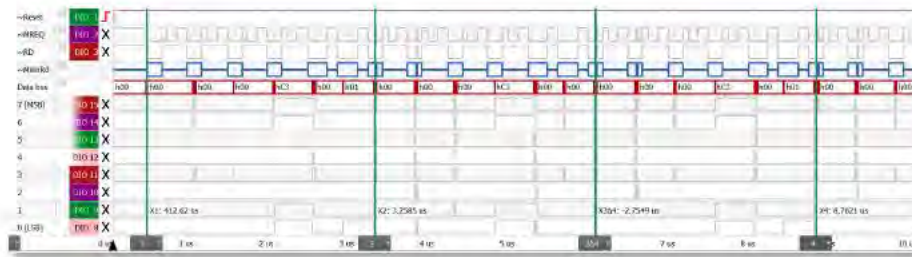


Figure 34 shows the results of the full-stack integrated test. Once the /RESET line is inactive (high), the Z80 CPU jumps to address 0x0000 and executes the three NOP operations. Then a jump to address 0x0100 is executed (JP 0x0100 translates to 0xC3 0x00 0x01). Again three NOP operations are executed and a jump back to address 0x0000 is executed (JP 0x0000 translates to 0xC3 0x00 0x00). And the cycle restarts, meaning that the Z80 reads all instructions from the EEPROM and keeps executing them in a timely fashion. Or with other words: The integrated test is a success.

A second more extended full-stack test listens to the serial port. Every character that enters will be send out to the serial port again. This test guarantees us **that the *Processor 2.0 board*, the *Memory 2.0 board* and the *Serial 2.0 board* can work together** in perfect harmony. The programs *S80_serial_test_output.asm* and *S80_serial_test_echo.asm* can be found in the **software resources file** of which one can find the reference on the title page of this document.

A last full-stack test **maps all four Z80 memory segments to the first EEPROM segment** and checks if it can read the same information from all memory segments. This program can be found in the **software resources file** of which one can find the reference on the title page of this document.

The **Memory 1.0 board** is for the full **100%** compatible with the **Processor 2.0** and with the **Serial 2.0 board**, so, the reader who doesn't want to use a Memory 2.0 PCB can still use the rest of the S80 version 2.0 with a Memory 1.0 board. An image of the PCB design of the Memory 2.0 can be found below (see *Figure 35*).

FIGURE 35 - MEMORY 2.0 - 514KB MEMORY PCB

6 Software development

The development environment for the S80 2.0 is the same as the development environment for the S80 1.0, and even most of the software can be used on both configurations. Just to be complete, we repeat below the software and hardware we have used, but we wish the reader to note that there are much more possible options than the ones mentioned. From our side, we have chosen the development environment that we feel comfortable with:

- **Operating system:** Linux, Mac OSx or Windows
- **Hardware simulation:** ModelSim, Quartus Prime (for Windows)
- **Terminal simulation:** PuTTY, RealTerm, Terminal window (for Linux and Mac)
- **Editor:** Programmers Notepad, Atom Editor, Syntra Small, Xcode
- **Cross-compiler:** z88dk (z80asm, zcc, appmake)
- **EEPROM programmer:** MiniPro TL866 Universal Programmer

7 Conclusion

Tests have shown that the Processor 2.0 board and the Serial 2.0 board are functioning as they should. A 100 μ F capacitor should replace the 22 μ F capacitor on the Processor 2.0 board to accommodate the Memory 2.0 board initialization.

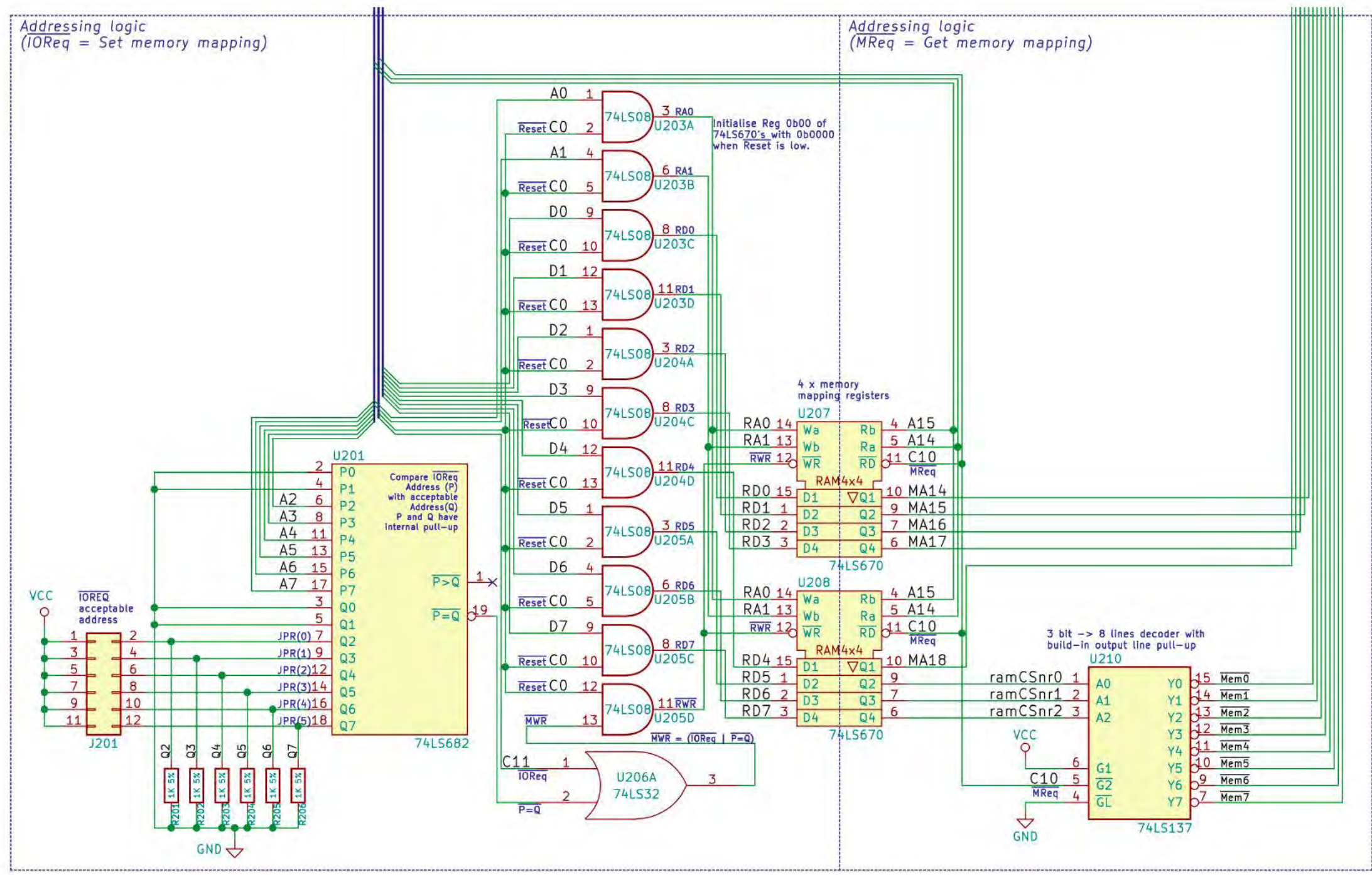
Further boards will be developed as examples of the possibilities of this educational project. The information on these boards will be included in a next version of this document.

Software will also be developed as an example of the possibilities of the S80 2.0 (memory mapping, ...). This will also be added to a next version of this document.

Below, the reader can find further reference material. There is also a zip file with all actually available hardware and software resources. A reference to this zip file is to be found at the bottom of the first page of this document.

Appendix 1 - VHDL memory addressing logic

To facilitate the reading of the below code, we also include the memory addressing logic schematic, Since the same component names and connection names are used.



```

1. -----
2. --
3. -- Project: S80
4. -- File:   S80_Memory_Logic.vhd
5. -- Version: 1.1
6. --
7. -- Author : Filip Pynckels
8. --
9. -- Created: 2018 12 02
10. -- Adapted: 2019 08 26
11. --
12. -- Build & Simulation tools:
13. --
14. --     Quartus Prime 18.1.0
15. --     Modelsim
16. --
17. -- Functionality:
18. --
19. --     S80 memory logic component
20. --
21. -----
22.
23.
24. -----
25. -- L I B R A R I E S
26. -----
27.
28. library IEEE;
29.     use IEEE.STD_LOGIC_1164.all;
30.
31.
32. -----
33. -- E N T I T Y   S 8 0 _ M E M O R Y _ L O G I C
34. -----
35.
36. entity S80_Memory_Logic is
37.
38.     port(
39.         J      : in STD_LOGIC_VECTOR( 7 downto 2);      -- Jumper settings: board IO base address
40.         A      : in STD_LOGIC_VECTOR(15 downto 0);      -- Z80 address lines
41.         D      : in STD_LOGIC_VECTOR( 7 downto 0);      -- Z80 data lines
42.         IOReq   : in STD_LOGIC;                        -- Z80 IOReq line
43.         MReq    : in STD_LOGIC;                        -- Z80 MReq line
44.         Reset   : in STD_LOGIC;                        -- Reset logic line
45.
46.         MA      : out STD_LOGIC_VECTOR(18 downto 14);   -- Address lines to EEPROM and SRAM's
47.         Mem     : out STD_LOGIC_VECTOR( 7 downto 0)     -- CE (active low) lines of each EEPROM/SRAM
48.     );
49.
50.     end S80_Memory_Logic;
51.
52.
53. -----
54. -- A R C H I T E C T U R E   S 8 0 _ M E M O R Y _ L O G I C
55. -----
56.
57. architecture S80_Memory_Logic of S80_Memory_Logic is
58.
59.
60.     -----
61.     -- Jumper(s) with eventual pulldown or pullup.
62.     -----
63.

```

```

64.      component S80_Jumpers is
65.
66.          generic(
67.              NR      : integer := 6 ;           -- Number of jumpers
68.              PROC    : integer := -1000        -- Processing of signal
69.          );
70.
71.          port(
72.              -- Ingoing line(s)
73.              I : in STD_LOGIC_VECTOR(NR downto 1); -- Value
74.              -- Outgoing line(s)
75.              O : out STD_LOGIC_VECTOR(NR downto 1) -- Result
76.          );
77.
78.      end component;
79.
80.
81.      -----
82.      -- Quad 2-input positive AND gate
83.      -----
84.
85.      component SN74LS08 is
86.
87.          port(
88.              -- Ingoing line(s)
89.              A : in STD_LOGIC_VECTOR(4 downto 1); -- Value 1
90.              B : in STD LOGIC VECTOR(4 downto 1); -- Value 2
91.              -- Outgoing line(s)
92.              Y : out STD_LOGIC_VECTOR(4 downto 1) -- Result
93.          );
94.
95.      end component;
96.
97.
98.      -----
99.      -- Quad 2-input positive OR gate
100.     -----
101.
102.     component SN74LS32 is
103.
104.         port(
105.             -- Ingoing line(s)
106.             A : in STD_LOGIC_VECTOR(4 downto 1); -- Value 1
107.             B : in STD LOGIC VECTOR(4 downto 1); -- Value 2
108.             -- Outgoing line(s)
109.             Y : out STD_LOGIC_VECTOR(4 downto 1) -- Result
110.         );
111.
112.     end component;
113.
114.
115.     -----
116.     -- 3 bit to 8 lines decoder
117.     -----
118.
119.     component SN74LS137 is
120.
121.         port(
122.             -- Ingoing line(s)
123.             A : in STD_LOGIC_VECTOR(2 downto 0); -- 3 bit number
124.             G1 : in STD_LOGIC;                   -- Output enable (active high)
125.             G2 : in STD_LOGIC;                   -- Output enable (active low)
126.             GL : in STD_LOGIC;                   -- Output enable (active low)
127.             -- Outgoing line(s)
128.             Y : out STD_LOGIC_VECTOR(7 downto 0) -- Decoded line (active low)
129.         );

```

```

130.
131.     end component;
132.
133.
134.     -----
135.     -- 4 x 4bit RAM
136.     -----
137.
138.     component SN74LS670 is
139.
140.         port(
141.             -- Ingoing line(s)
142.             Wa : in STD_LOGIC_VECTOR(1 downto 0);      -- Write address (0..3)
143.             WR : in STD_LOGIC;                          -- Active low write pulse
144.             D  : in STD_LOGIC_VECTOR(4 downto 1);      -- Content to write
145.             Ra : in STD_LOGIC_VECTOR(1 downto 0);      -- Read address (0..3)
146.             RD : in STD_LOGIC;                          -- Active low read pulse
147.             -- Outgoing line(s)
148.             Q  : out STD_LOGIC_VECTOR(4 downto 1)      -- Read content
149.         );
150.
151.     end component;
152.
153.
154.     -----
155.     -- 8 bit comparator with active low output for P=Q and P>Q
156.     -----
157.
158.     component SN74LS682 is
159.
160.         port(
161.             -- Ingoing line(s)
162.             P  : in STD_LOGIC_VECTOR(7 downto 0);      -- Value 1
163.             Q  : in STD_LOGIC_VECTOR(7 downto 0);      -- Value 2
164.             -- Outgoing line(s)
165.             Not_P_Eq_Q : out STD_LOGIC;                -- Active low P=Q
166.             Not_P_Gt_Q : out STD_LOGIC;                -- Active low P>Q
167.         );
168.
169.     end component;
170.
171.
172.     -----
173.     -- Internal connections
174.     -----
175.
176.     signal JPR      : STD_LOGIC_VECTOR(5 downto 0);    -- Signal between S80_Jumpers and SN74LS682
177.     signal notPeqQ  : STD_LOGIC;                      -- Signal between SN74LS682 and SN74LS32
178.     signal notMWR    : STD_LOGIC;                     -- Signal between SN74LS32 and SN74LS08
179.     signal notRWR    : STD_LOGIC;                     -- Signal between SN74LS08 and SN74LS670
180.     signal RA        : STD_LOGIC_VECTOR(1 downto 0);   -- Signal between SN74LS08 and SN74LS670
181.     signal RD        : STD_LOGIC_VECTOR(7 downto 0);   -- Signal between SN74LS08 and SN74LS670
182.     signal ramCSnr   : STD_LOGIC_VECTOR(2 downto 0);   -- Signal between SN74LS670 and SN74LS137
183.
184.     signal openPins : STD_LOGIC_VECTOR(3 downto 0);    -- Solve VHDL "partial open pins on bus" problem
185.
186.
187.     -----
188.     -- Architecture
189.     -----
190.
191.     begin
192.
193.         -- Set memory mapping logic
194.
195.         J201: S80_Jumpers

```



```

196.
197.     port map (
198.         -- Ingoing line(s)
199.         I => J,                                -- Value
200.         -- Outgoing pins
201.         O => JPR                                -- 1K pulled up result
202.     );
203.
204.     U201: SN74LS682
205.
206.     port map (
207.         -- Ingoing line(s)
208.         P(7 downto 2) => A(7 downto 2),          -- Value 1
209.         P(1 downto 0) => (others => '0'),
210.         Q(7 downto 2) => JPR,                    -- Value 2
211.         Q(1 downto 0) => (others => '0'),
212.         -- Outgoing pins
213.         Not_P_Eq_Q    => notPeqQ,                -- Comparison result
214.         Not_P_Gt_Q    => openPins(0)
215.     );
216.
217.     U206: SN74LS32
218.
219.     port map (
220.         -- Ingoing line(s)
221.         A(1)          => IOReq,                  -- Value 1
222.         A(4 downto 2) => (others => '0'),          -- Noise reduction
223.         B(1)          => notPeqQ,                -- Value 2
224.         B(4 downto 2) => (others => '0'),          -- Noise reduction
225.         -- Outgoing line(s)
226.         Y(1)          => notMWR,                 -- OR result
227.         Y(4 downto 2) => openPins(2 downto 0)     -- Open pin
228.     );
229.
230.     U203: SN74LS08
231.
232.     port map (
233.         -- Ingoing line(s)
234.         A(2 downto 1) => A(1 downto 0),          -- Value 1
235.         A(4 downto 3) => D(1 downto 0),
236.         B => (others => Reset),                  -- Value 2
237.         -- Outgoing line(s)
238.         Y(2 downto 1) => RA(1 downto 0),          -- AND result
239.         Y(4 downto 3) => RD(1 downto 0)
240.     );
241.
242.     U204: SN74LS08
243.
244.     port map (
245.         -- Ingoing line(s)
246.         A(4 downto 1) => D(5 downto 2),          -- Value 1
247.         B              => (others => Reset),      -- Value 2
248.         -- Outgoing line(s)
249.         Y(4 downto 1) => RD(5 downto 2)          -- AND result
250.     );
251.
252.     U205: SN74LS08
253.
254.     port map (
255.         -- Ingoing line(s)
256.         A(2 downto 1) => D(7 downto 6),          -- Value 1
257.         A(3)          => notMWR,
258.         A(4)          => '0',                    -- Noise reduction
259.         B(3 downto 1) => (others => Reset),      -- Value 2
260.         B(4)          => '0',                    -- Noise reduction
261.         -- Outgoing line(s)

```

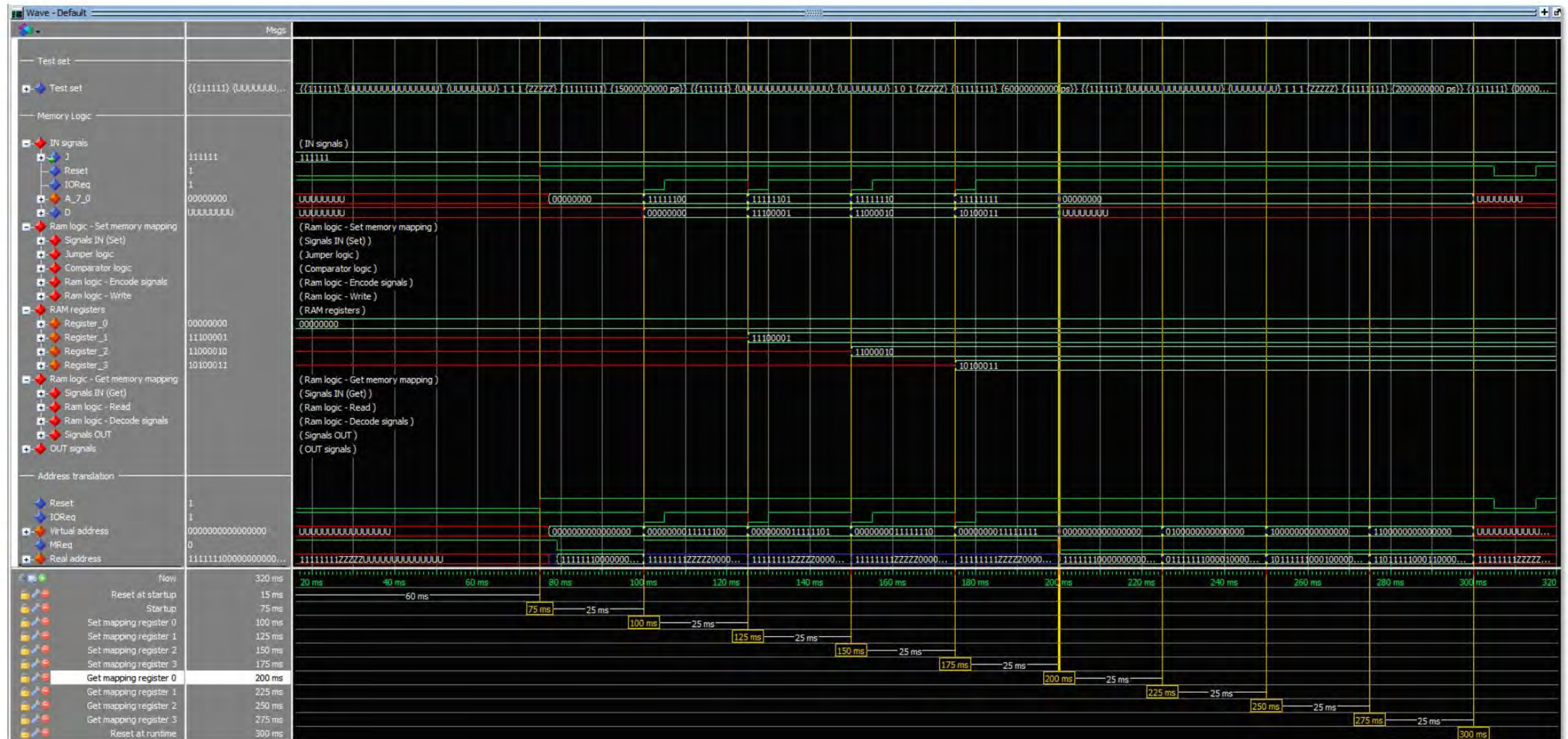
```

262.          Y(2 downto 1) => RD(7 downto 6),          -- OR result
263.          Y(3)          => notRWR,
264.          Y(4)          => openPins(3)              -- Open pin
265.      );
266.
267.
268.      -- Memory mapping storage
269.
270.      U207: SN74LS670
271.
272.      port map (
273.          -- Ingoing line(s)
274.          Wa => RA(1 downto 0),          -- Write address (0..3)
275.          WR => notRWR,                  -- Active low write pulse
276.          D  => RD(3 downto 0),          -- Content to write
277.          Ra => A(15 downto 14),          -- Read address (0..3)
278.          RD => MReq,                    -- Active low read pulse
279.          -- Outgoing line(s)
280.          Q  => MA(17 downto 14)          -- Read content
281.      );
282.
283.      U208: SN74LS670
284.
285.      port map (
286.          -- Ingoing line(s)
287.          Wa => RA(1 downto 0),          -- Write address (0..3)
288.          WR => notRWR,                  -- Active low write pulse
289.          D  => RD(7 downto 4),          -- Content to write
290.          Ra => A(15 downto 14),          -- Read address (0..3)
291.          RD => MReq,                    -- Active low read pulse
292.          -- Outgoing line(s)
293.          Q(1) => MA(18),                -- Read content
294.          Q(4 downto 2) => ramCSnr
295.      );
296.
297.
298.      -- Get memory mapping logic
299.
300.      U210: SN74LS137
301.
302.      port map(
303.          -- Ingoing line(s)
304.          A  => ramCSnr,                  -- 3 bit number
305.          G1 => '1',                      -- Output enable (active high)
306.          G2 => MReq,                      -- Output enable (active low)
307.          GL => '0',                      -- Output enable (active low)
308.          -- Outgoing line(s)
309.          Y  => Mem                        -- Decoded line (active low)
310.      );
311.
312.
313.      end S80_Memory_Logic;

```

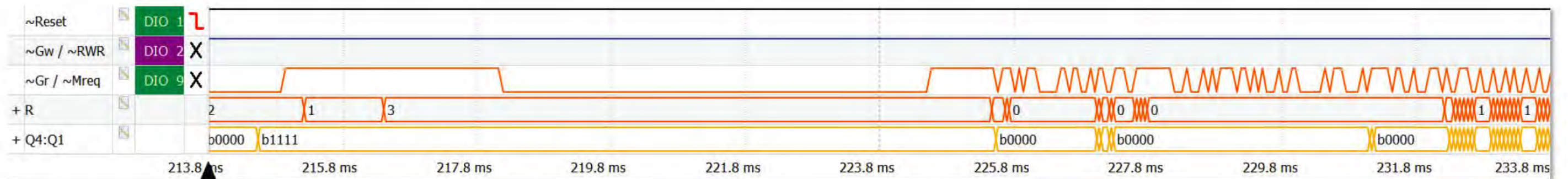
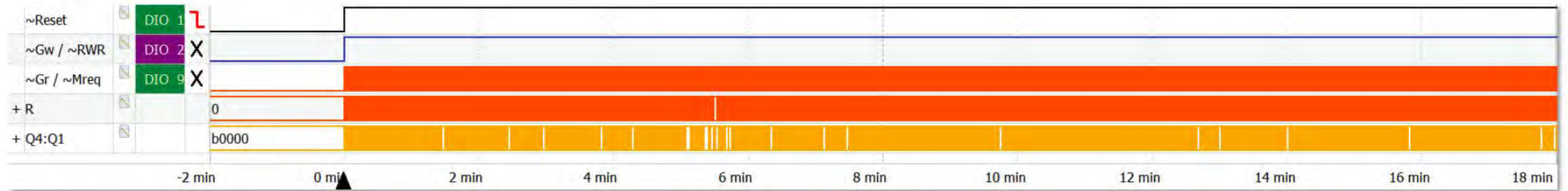
Appendix 2 - VHDL memory addressing simulation results

The below image is also shown in *Figure 22*, but for reasons of readability, we include a large version of this figure below.



Appendix 3 - VHDL memory addressing simulation results

The below images are also shown in *Figure 30*, but for reasons of readability, we include a large version of these images below. The first image shows the line values during a longer time interval. The second image shows the line values in detail during a small time interval.



Appendix 4 - Offline memory test - Controller program

The below table shows the test matrix for the offline memory controller tests. The test matrix contains as well the input for each test and each test line, as the results of the test with eventual remarks.

[illegible]

Remark 1: JPR(5):JPR(0) = 80h (0b100000)

Remark 2: Method to change address or data lines: deactivate IOreq, set address and data lines, reactivate loreq or other control signal as necessary for the following test line.

Remark 3: Test 11 shows that there seems to be a problem with the IOREQ and the MREQ address lines of the 74LS670's