Die Rechenwerke RW-Pioneer

Luca Anthony Schwarz / Die Rechenwerke

April 30, 2024, Hannover

General information

The purpose of this paper is to provide an overview over the RW-Pioneer computer system. All implementation details in this paper only apply to the physical RW-Pioneer system implementation based on discrete integrated circuits and components. The Verilog-based version utilizes the same instruction set, but performs operations in three instead of two clock cycles. The changes were required to make the model synthesizable on FPGAs.

Contents

1 Architecture specification

1.1 General architecture description

RW-Pioneer is a 4-bit computer system that is based on the Harvard computer architecture. All mathematical operations are performed on 4-bit wide sets of binary data in the ALU. Wider operations are possible by using the carry flag of the previous operation. This requires strict instruction ordering, as multiple additions or subtraction operations must be executed consecutively. RAM address space is split up into sixteen addresses. Each address corresponds to one 4-bit word of storage in RAM. Program memory consists of 256 bytes, with one instruction being encoded by two 4-bit words. The program counter (PC) can be altered by instructions to allow for jumps and branches. As memory access is only possible using immediate addresses, the system can not be considered general purpose and is not Turing-complete.

The architecture can be seen in the block diagram 1. The diagram shows all main components of the architecture with their respective connections and splits them into two groups (Step0 and Step1). Instructions are executed in two clock cycles. In the first cycle (Step0) the next instruction will be loaded into the instruction register and decoded by the instruction decoder which sets the control signal lines according to the loaded instruction. In the second cycle (Step1) the instruction is then executed and the step counter resets to zero while the program counter advances by 1 if no jump instruction was performed. Jump instructions write into the program counter directly without incrementing it by 1.

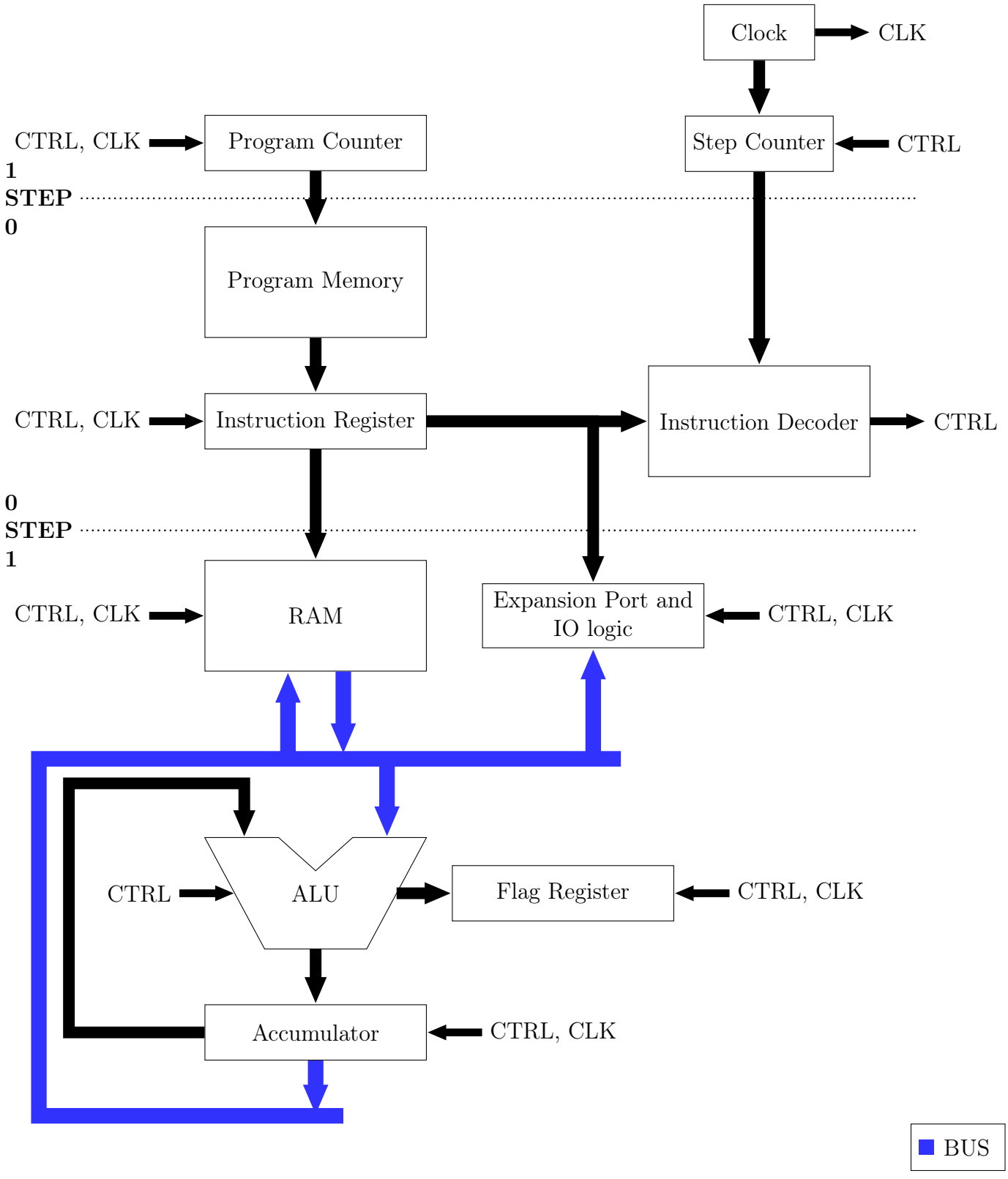


Figure 1: DLC1 architecture block diagram

1.2 Instruction set

Instructions are encoded into one byte of data. The most significant word is called instruction code (IC) and the least significant word is called instruction value (IV). The instruction code defines the instruction type, and the instruction value is the immediate parameter of that instruction.

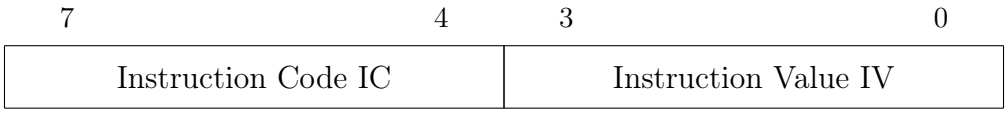


Figure 2: Instruction Bit Field

As four bits are used to encode an instruction's code, a maximum of 16 different instructions are supported. The implemented instructions are listed in table 1. Not all instructions require an instruction value. It is advised to use an instruction value of 0xF in such cases.

RW-PIONEER instruction set		
Instruction	IC	Functional description
LDI	0x0	ACC $\leftarrow$ IV
LDA	0x1	ACC $\leftarrow$ RAM[IV]
STOA	0x2	RAM[IV] $\leftarrow$ ACC
STOB	0x3	RAM[IV] $\leftarrow$ BUS
ADD	0x4	ACC $\leftarrow$ ACC + RAM[IV]
ADDI	0x5	ACC $\leftarrow$ ACC + IV
ADDC	0x6	ACC $\leftarrow$ ACC + RAM[IV] + CF
SUB	0x7	ACC $\leftarrow$ ACC − RAM[IV]
SUBI	0x8	ACC $\leftarrow$ ACC − IV
SUBC	0x9	ACC $\leftarrow$ ACC − RAM[IV] − CF
SJMP	0xA	PCL $\leftarrow$ IV
JMP	0xB	PCL $\leftarrow$ IV and PCH $\leftarrow$ ACC
BNEZ	0xC	PCL $\leftarrow$ IV and PCH $\leftarrow$ ACC if ZF = LOW
BEZ	0xD	PCL $\leftarrow$ IV and PCH $\leftarrow$ ACC if ZF = HIGH
HALT	0xE	Halt program execution (set HLT signal HIGH)
NOOP	0xF	Perform no operation

Table 1: RW-Pioneer instruction set

1.3 Micro operations

Instruction codes are interpreted by an instruction decoder. The decoder then supplies the ALU and the memory unit with appropriate control signals, so-called micro operations. Interestingly,

the memory unit controls the instruction and random access memories, as well as the program counter. The instruction counter is enabled or disabled by the CE signal, and the system is halted by the HLT signal. The ALU performs the mathematical operations and controls the accumulator and flag registers. The control signals are influenced by the step counter, as the system requires two cycles per instruction. In step 0 a new instruction is loaded in all cases. Only in step 1 the fetched instruction is interpreted and executed. Conditional branches are only executed, if the stored zero flag value ZF is a match for the given branch instruction. For example, to take the branch on an BNEZ instruction, the ZF signal is required to be LOW. The internal control signals and micro operations are listed in table 2.

RW-Pioneer instruction to micro instruction look-up table						
Instruction	Step	ZF	ALU	MEM	CE	HLT
X	0	X	ALU_NOOP	MEM_LDINSTRC	L	L
LDI	1	X	ALU_BUSTOACC	MEM_IVTOBUS	H	L
LDA	1	X	ALU_BUSTOACC	MEM_RAMTOBUS	H	L
STOA	1	X	ALU_ACCTOBUS	MEM_BUSTORAM	H	L
STOB	1	X	ALU_NOOP	MEM_BUSTORAM	H	L
ADD	1	X	ALU_ADD	MEM_RAMTOBUS	H	L
ADDI	1	X	ALU_ADD	MEM_IVTOBUS	H	L
ADDC	1	X	ALU_ADDC	MEM_RAMTOBUS	H	L
SUB	1	X	ALU_SUB	MEM_RAMTOBUS	H	L
SUBI	1	X	ALU_SUB	MEM_IVTOBUS	H	L
SUBC	1	X	ALU_SUBC	MEM_RAMTOBUS	H	L
SJMP	1	X	ALU_NOOP	MEM_SJMP	L	L
JMP	1	X	ALU_NOOP	MEM_JMP	L	L
BNEZ	1	L	ALU_NOOP	MEM_JMP	L	L
BNEZ	1	H	ALU_NOOP	MEM_NOOP	H	L
BEZ	1	L	ALU_NOOP	MEM_NOOP	H	L
BEZ	1	H	ALU_NOOP	MEM_JMP	L	L
HALT	1	X	ALU_NOOP	MEM_NOOP	L	H
NOOP	1	X	ALU_NOOP	MEM_NOOP	H	L

Table 2: RW-Pioneer instruction to micro instruction look-up table

The ALU is supplied with a single micro operation per step by the instruction decoder. The micro operation is supplied as a 3-bit wide signal (AIC), which is then interpreted by the ALU as described in table 3. The ALU micro instruction decoder directly controls ALU operation by using six internal control signals. The signals AI0 and AI1 define, whether an addition or a subtraction is performed and if this operation uses the currently stored carry flag. $\overline{\text{ACCTBUS}}$ controls if the accumulator value is to be sent on the system bus. SSACC defines the accumulator source. This can be the system bus, or the result of the ALU. $\overline{\text{EACC}}$ disables the accumulator register and $\overline{\text{FI}}$ disables storing the zero and the carry flags for an operation.

The memory unit is supplied with a single micro operation per step by the instruction

RW-Pioneer ALU micro instructions							
Micro instruction	AIC	AI1	AI0	$\overline{\text{ACCTBUS}}$	SSACC	$\overline{\text{EACC}}$	$\overline{\text{FI}}$
ALU_ADD	0x0	L	L	H	L	L	L
ALU_ADDC	0x1	L	H	H	L	L	L
ALU_SUB	0x2	H	L	H	L	L	L
ALU_SUBC	0x3	H	H	H	L	L	L
ALU_BUSTOACC	0x4	H	H	H	H	L	H
ALU_ACCTOBUS	0x5	H	H	L	H	H	H
ALU_NOOP	0x6	H	H	H	H	H	H
ALU_NOOP	0x7	H	H	H	H	H	H

Table 3: RW-Pioneer ALU micro instructions

decoder. The micro operation is supplied as a 3-bit wide signal (MIC), which is then interpreted by the memory unit as described in table 4. The memory unit micro instruction decoder directly controls memory, bus and program counter operations by using six internal control signals. $\overline{\text{JL}}$ disables loading the least significant four bits of the program counter from the instruction value of the current instruction. $\overline{\text{JH}}$ disables loading the most significant four bits of the program counter from the accumulator. $\overline{\text{LI}}$ disables loading the next instruction. This is only used in step 0 to load the next instruction. $\overline{\text{WERAM}}$ sets the write enable signal of RAM. $\overline{\text{CERAM}}$ sets the chip enable signal of RAM. $\overline{\text{IVTBUS}}$ disables putting the instruction value of the current instruction to the system bus.

RW-Pioneer memory micro instructions							
Micro instruction	MIC	$\overline{\text{JL}}$	$\overline{\text{JH}}$	$\overline{\text{LI}}$	$\overline{\text{WERAM}}$	$\overline{\text{CERAM}}$	$\overline{\text{IVTBUS}}$
MEM_SJMP	0x0	L	H	H	H	H	H
MEM_JMP	0x1	L	L	H	H	H	H
MEM_BUSTORAM	0x2	H	H	H	L	L	H
MEM_RAMTOBUS	0x3	H	H	H	H	L	H
MEM_IVTOBUS	0x4	H	H	H	H	H	L
MEM_LDINSTRC	0x5	H	H	L	H	H	H
MEM_NOOP	0x6	H	H	H	H	H	H
MEM_NOOP	0x7	H	H	H	H	H	H

Table 4: RW-Pioneer memory micro instructions

2 Hardware components

RW-Pioneer was designed with simplicity as the primary goal. Therefore, commonly available components were chosen for physical implementation. For timing, the 555 timer ICs were used. Program memory and the instruction decoder's look-up table use the SST39SD010A IC. It is a cheap parallel FLASH IC that is also easy to program using microcontrollers. Both memories

should never be directly soldered to the PCB, as sockets make programming the memory ICs way easier. The RAM is realized by the CY7C164 chip, which is the only rare component in the whole design. All logic functions are implemented with the 74HC-Series and 74HCT-Series logic chips. 74LS ICs are not supported, but might still work.

Passive components can be chosen freely as long as the correct values are used. For example, the 1N4001 diodes may also be replaced by 1N4007 diodes or comparable ones if necessary, as long as the forward voltage drop remains at roughly 0.7 volts. The resistor values for all resistors used in conjunction with LEDs may be chosen freely as long as the resistor value is high enough. When using different color LEDs different values for different colors might be used to keep a balance of apparent brightness between them.

3 Expansion port

The RW-Pioneer provides an expansion port (also called user port) to communicate with external circuitry. The pin assignments of the expansion port are listed in table 5. The port allows access to the instruction value of the current instruction and also to the data on the system bus. Instructions interfacing with system RAM can be identified by reading the values of the R, W and IOR signals. Access is also provided to the MR, HLT and CLK signals. Sending data and receiving data over the port is done via memory mapping. When the system writes data to memory W will be HIGH and when reading R will be HIGH. IOR is HIGH when main memory is accessed in the address range $11xx_2$. All signals provide by the port are 5V CMOS compatible. The expansion port allows for powering the connected external device, if the power requirement does not exceed the capabilities of the system's power supply.

RW-Pioneer expansion port pin assignment	
Pins	Function
1, 2, 19, 20	GND
15, 17	VCC
4, 6, 8, 10	IV of current instruction from LSB to MSB
3, 5, 7, 9	BUS from LSB to MSB
11	RE. HIGH when reading from RAM
12	MR. HIGH if reset signal provided
13	WR. HIGH when writing to RAM
14	HLT. HIGH when halt signal is provided
16	IOR. HIGH when RAM is accessed from addresses 1100_2 to 1111_2
18	CLK. Direct link to the CLK signal

Table 5: RW-Pioneer expansion port pin assignment

3.1 Sending data over the expansion port

Sending data should be implemented via memory mapping. To showcase this a 4-bit register (74HC173) is connected to the expansion port which is mapped to RAM address 1111_2. The CLK signal is connected to the CLK input of the 4-bit register. The enable signal (E) (active

HIGH) is connected to logic that computes $E = IV_0 \wedge IV_1 \wedge IOR$ and the input data lines (D_0 to D_3) of the register are connected to BUS (BUS_0 to BUS_3). When the system executes the STOA instruction with IV = 1111_2, the values in the accumulator will be written to RAM at address IV. The input E of the register will transition to HIGH, so the register is enabled. Then the CLK signal transitions to HIGH and the register will store the value that was on the BUS (which was also written to RAM). In conclusion, this setup links the register connected via the expansion port to a RAM address. Expansions like a simple LCD can be controlled in this way.

3.2 Receiving data over the expansion port

Receiving data should be implemented via memory mapping like sending data. The difference to sending is, that receiving data requires the STOB instruction, to first store the data at a given address in RAM. Clever memory mapping schemes would allow for direct accessing of external data, but the RW-Pioneer does not have such functionality. To showcase receiving data, a 4-bit register (74HC173) is connected to the expansion port, which is mapped to address memory address 1111_2. The CLK signal is connected to the CLK input of the 4-bit register. The output enable signal OE (active HIGH) is connected to logic that computes $OE = IV_0 \wedge IV_1 \wedge IOR$ and the output data lines (Q_0 to Q_3) of the register are connected to BUS (BUS_0 to BUS_3). When the system executes the STOB instruction with IV = 1111_2, the values on the BUS will be written to RAM at address IV. The OE input of the register will transition to HIGH, so the register will write its stored value to the BUS. As a result, RAM will store the register's stored value at the address 1111_2. In conclusion, this setup links the register connected via the expansion port to a RAM address. Expansions like a very simple sensor module can be realized using this memory mapping method.

4 Expansion modules

4.1 HEX-Module

The HEX-Module provides the system with four seven-segment displays to display four-digit hexadecimal values. Each digit's value is dependent on the value the accompanying register stores. The registers are directly writable by the RW-Pioneer.

4.2 1602-Module

The 1602-Module expands the system with a 1602-LCD, that is completely controllable by the RW-Pioneer. With the module, it is quite easy to display small texts and numbers.

5 Programming

The system is programmed by writing instructions to the instruction memory chip. A simple memory programmer called Memory-Programmer is provided with the RW-Pioneer to fulfill this task. The programmer employs an ATmega328 microcontroller in the form of an Arduino Nano and is controlled by a USB host through a simple host software. The host computer simply specifies a hex file to program to memory, and the memory programmer will complete this task in a few seconds.

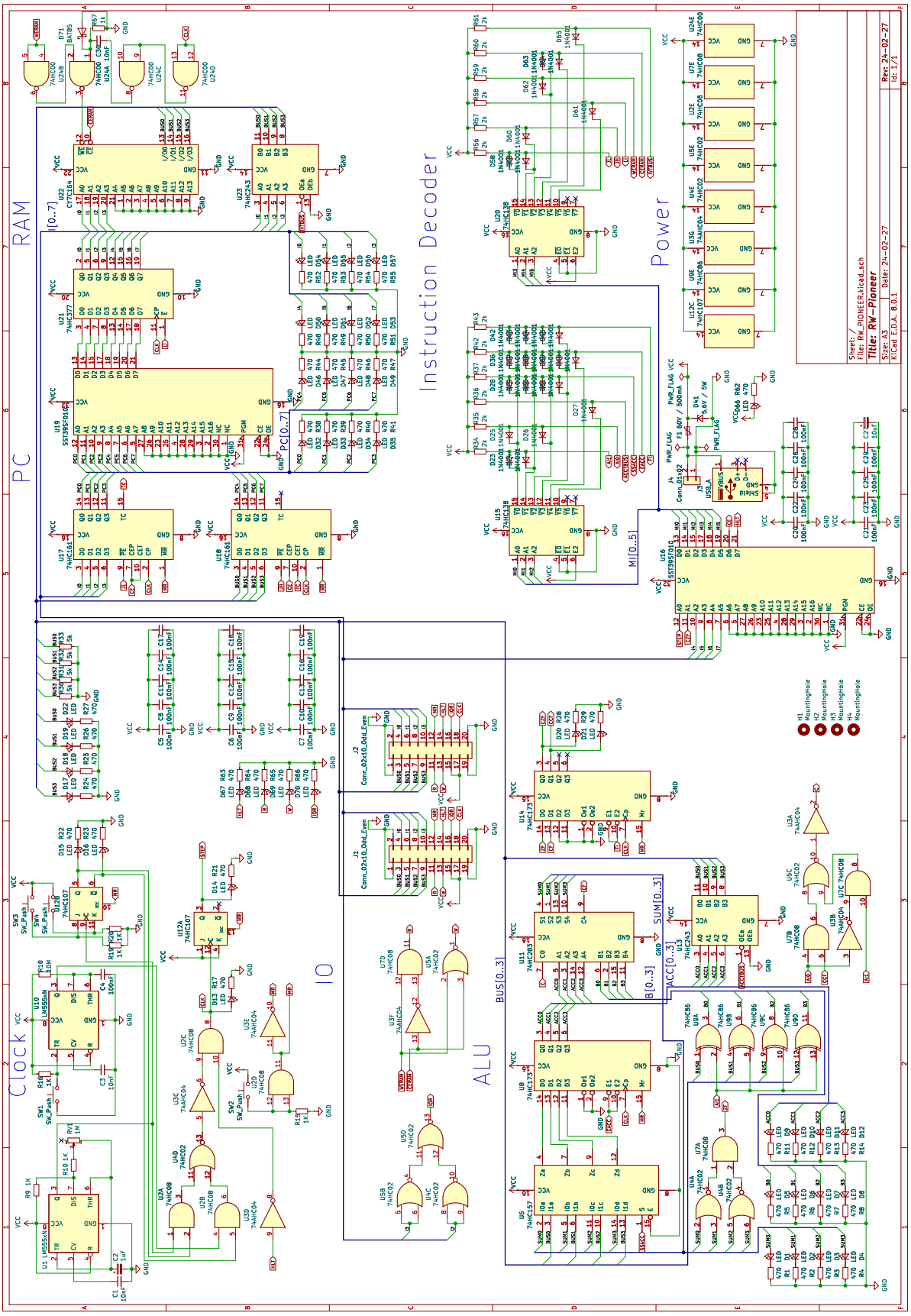
6 Emulation

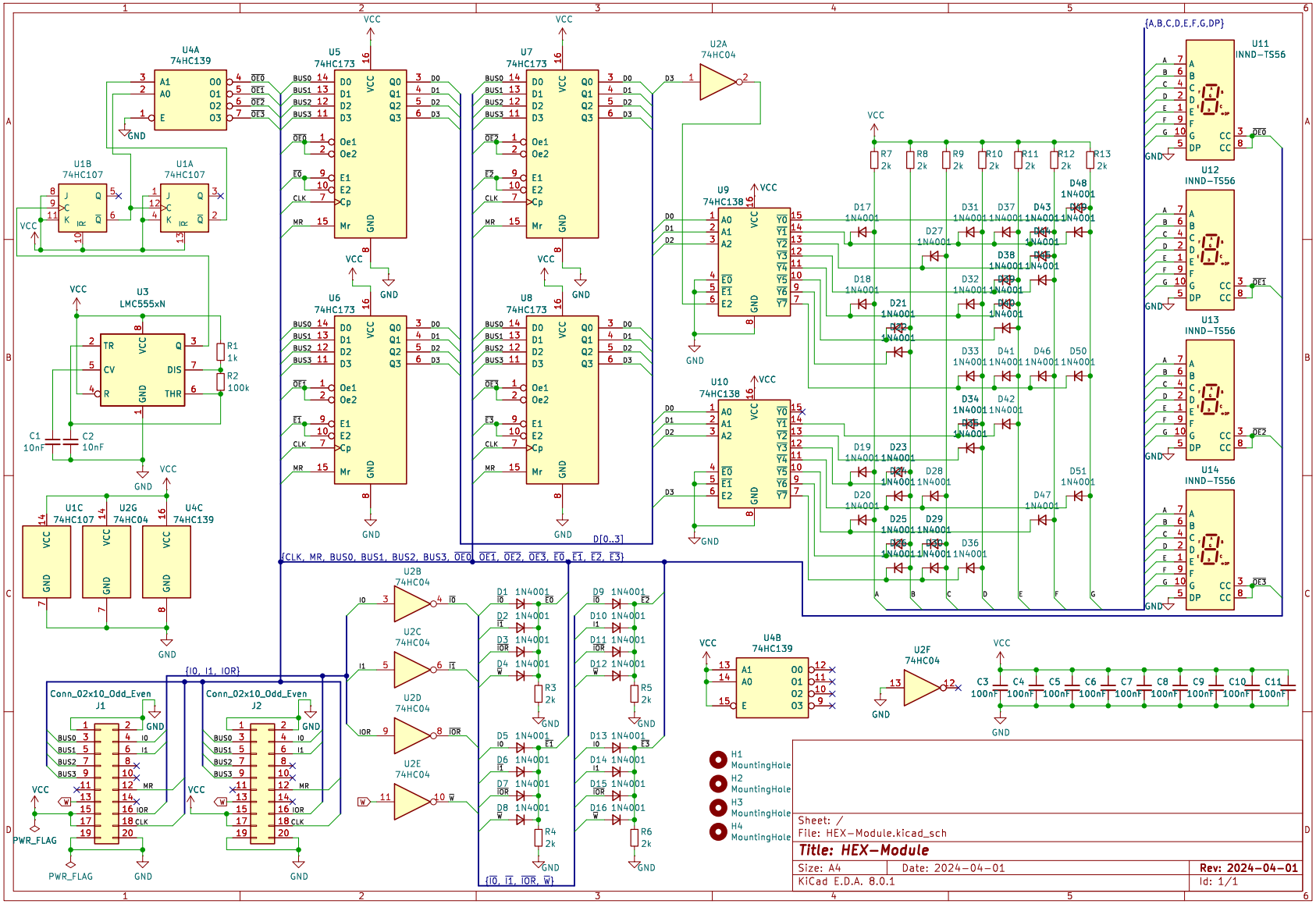
A simple emulation software called RWEMU for the RW-Pioneer is supplied alongside schematics and this document. With this emulator, programs can be tested before running them on a physical design. This helps to eliminate errors introduced by the programmer but can also be used to verify the physical design by comparing emulation and measured/observed results.

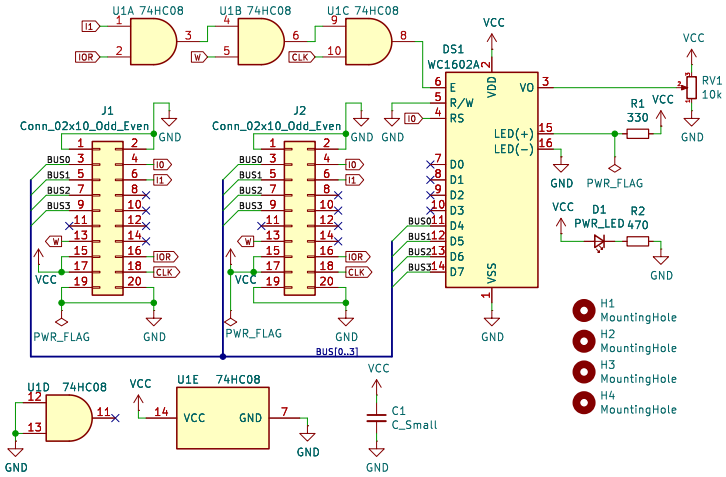
7 Verilog model

Parts of the RW-Pioneer system are modeled in the Verilog HDL. This model is supplied alongside this document and implements all essential components, except the manual clocking mechanism and the expansion port. The model and the physical implementation are binary compatible.

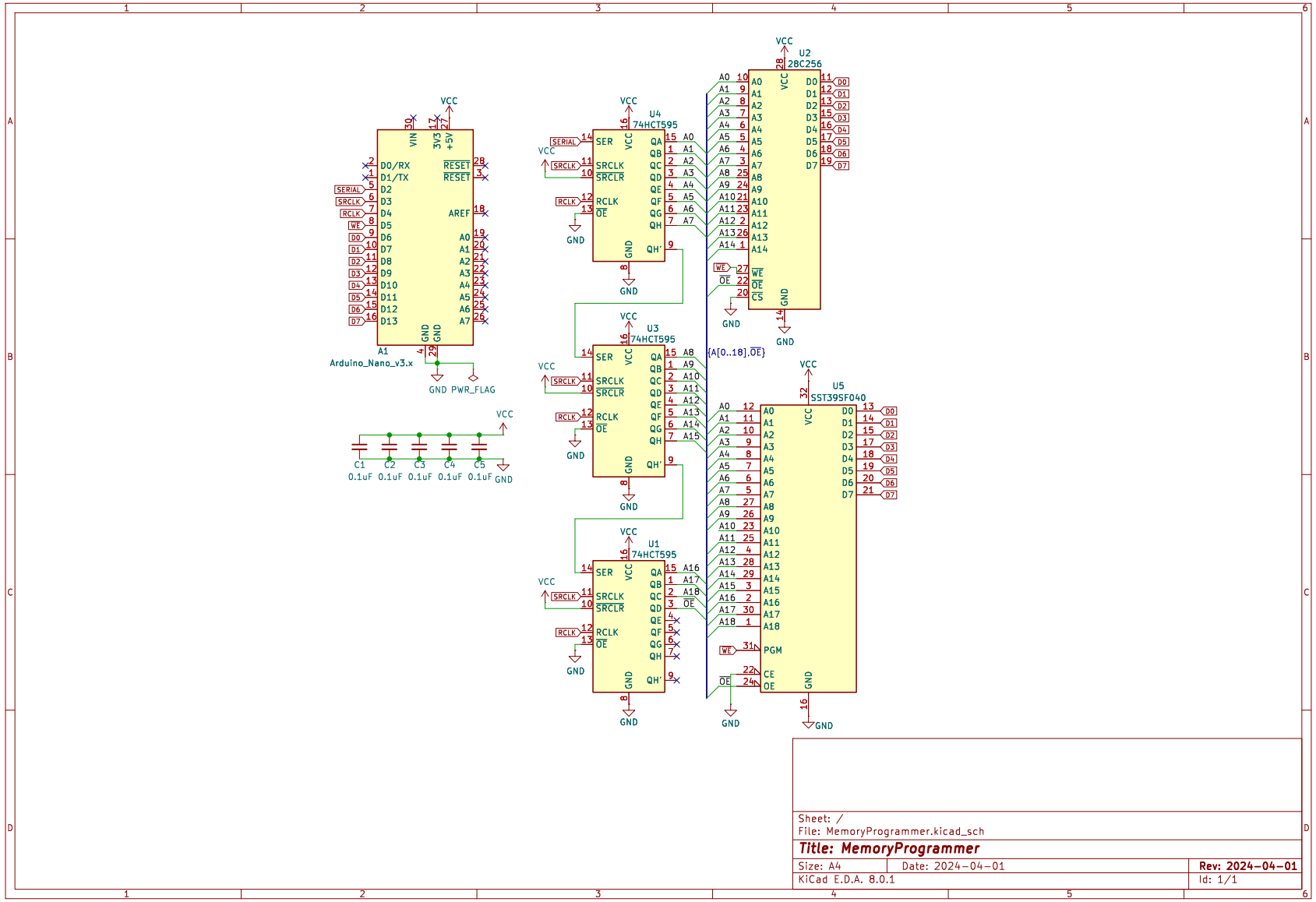
8 Schematics




U4A
74HC139
U5
74HC173
U7
74HC173
U2A
74HC04
U1B
74HC107
U1A
74HC107
U3
LMC555xN
U6
74HC173
U8
74HC173
U9
74HC138
U10
74HC138
U11
INND-TS56
U12
INND-TS56
U13
INND-TS56
U14
INND-TS56
U1C
74HC107
U2G
74HC04
U4C
74HC139
U2B
74HC04
U2C
74HC04
U2D
74HC04
U2E
74HC04
U4B
74HC139
U2F
74HC04
J1
Conn_02x10_Odd_Even
J2
Conn_02x10_Odd_Even
{A,B,C,D,E,F,G,DP}
D[0..3]
{CLK, MR, BUS0, BUS1, BUS2, BUS3, OE0, OE1, OE2, OE3, E0, E1, E2, E3}
{I0, I1, IOR}
{I0, I1, IOR, W}
VCC
GND
PWR_FLAG
H1 MountingHole
H2 MountingHole
H3 MountingHole
H4 MountingHole
Sheet: /
File: HEX-Module.kicad_sch
Title: HEX-Module
Size: A4
Date: 2024-04-01
Rev: 2024-04-01
KiCad E.D.A. 8.0.1
Id: 1/1



Sheet: /	
File: 1602-Module.kicad_sch	
Title: 1602-Module	
Size: A4	Date: 2024-04-01
KiCad E.D.A. 8.0.1	**Rev: 2024-04-01**
	Id: 1/1


A1
Arduino_Nano_v3.x
U1
74HCT595
U2
28C256
U3
74HCT595
U4
74HCT595
U5
SST39SF040
C1 0.1uF
C2 0.1uF
C3 0.1uF
C4 0.1uF
C5 0.1uF
VCC
GND
PWR_FLAG
SERIAL
SRCLK
RCLK
WE
{A[0..18],OE}
Sheet: /
File: MemoryProgrammer.kicad_sch
Title: MemoryProgrammer
Size: A4
Date: 2024-04-01
Rev: 2024-04-01
KiCad E.D.A. 8.0.1
Id: 1/1