Hardware-Systeme im SoSe 2020 Protokoll
zum Praktikum

VISCY-Prozessor- Teil 3

Gruppenmitglieder: **V**annessa **K**emeni **K**abiwo
Thibaut **T**EMKENG
Niko **M**aximiliam **M**azanec

1 Vollständiges Steuerwerk.

1.1 Zustandsübergangsdiagramm.

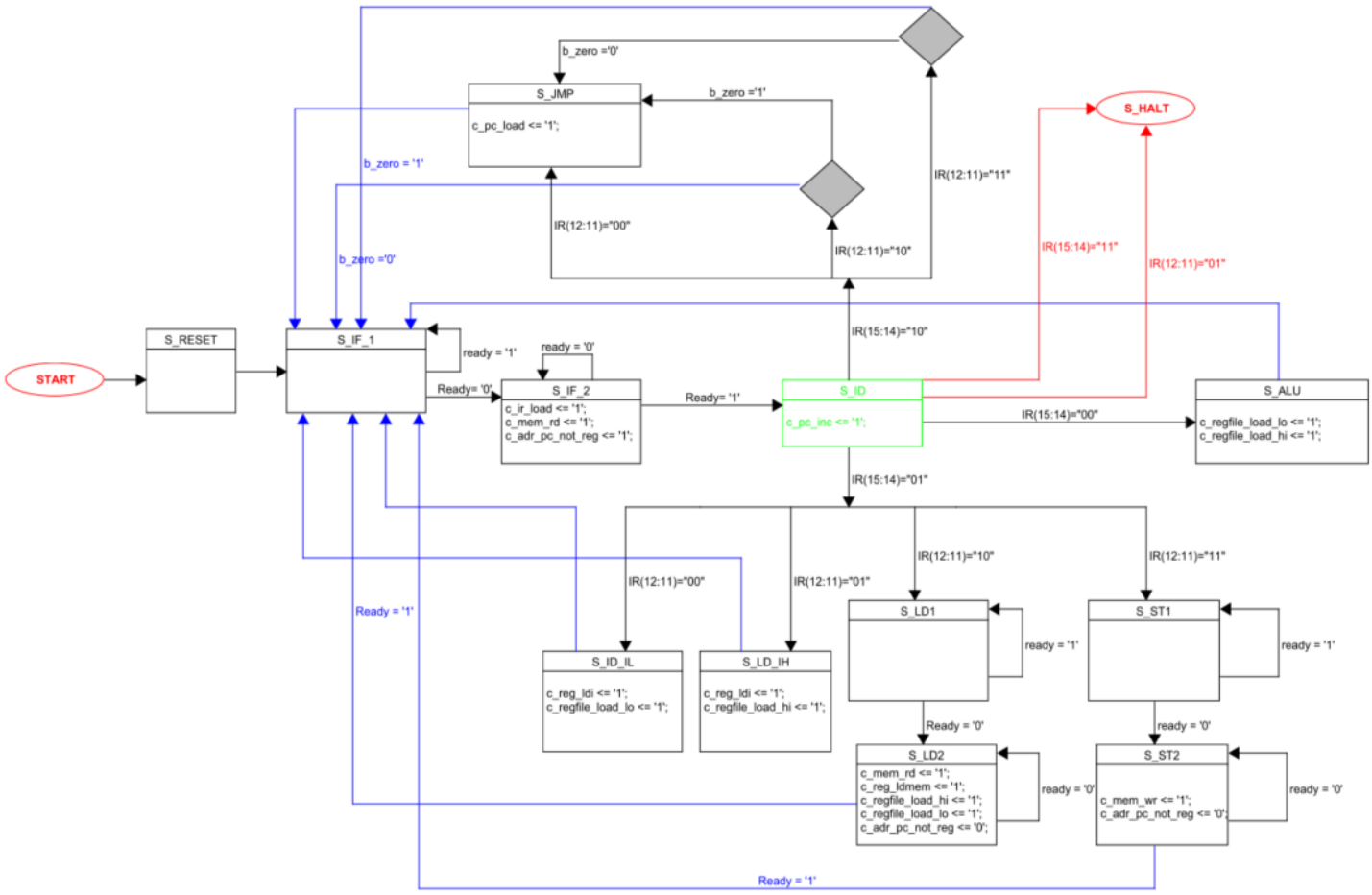

Abbildung 1: Zustandsdiagramm

1.2 VHDL Code des Steuerwerks.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity CONTROLLER is
        port (   clk, reset: in std_logic;
                        ir: in std_logic_vector(15 downto 0);  -- Befehlswort
                        ready, zero: in std_logic;          -- weitere Statussignale
                        c_reg_ldmem, c_reg_ldi,             -- Auswahl beim Register-Laden
                        c_regfile_load_lo, c_regfile_load_hi,  -- Steuersignale Reg.-File
                        c_pc_load, c_pc_inc,                -- Steuereingaenge PC
                        c_ir_load,                          -- Steuereingang IR
                        c_mem_rd, c_mem_wr,                 -- Signale zum Speicher
                        c_adr_pc_not_reg : out std_logic    -- Auswahl Adress-Quelle
);end CONTROLLER;

architecture RTL of CONTROLLER is
        --Aufzaehlung fuer den Zustand
        type t_state is (s_reset, s_if1, s_if2, s_id, s_alu, s_ldil, s_ldih,
        s_halt, s_ld1, s_ld2, s_st1, s_st2, s_jmp);
        signal state, next_state : t_state;
begin
        --Zustandsregister
        process (clk)   --(nur) Taktsignal in Sensitivitaetsliste
```

```
begin
        if rising_edge (clk) then
                if reset = '1' then
                        state <= s_reset;
                else
                        state <= next_state;
                end if;
        end if;
end process;

--prozess fuer die uebergangs- und Ausgabefunktion
process (state, ready, zero, ir)
begin
        --Dfault-werte fuer alle Ausgngssignale
        c_regfile_load_lo <= '0';
        c_regfile_load_hi <= '0';
        c_adr_pc_not_reg <= '0';
        c_mem_rd <= '0';
        c_mem_wr <= '0';
        c_ir_load <= '0';
        c_pc_load <= '0';
        c_pc_inc <= '0';
        c_reg_ldmem <= '0';
        c_reg_ldi <= '0';

--Zustandsabhaengige Belegung
        -- eingentliche Automaten-Logik
        next_state <= state;
        case state is
                when s_reset =>
                        next_state <= s_if1;
                when s_if1 =>
                        if ready = '0' then
                                next_state <= s_if2;
                        end if;
                when s_if2 =>
                        c_mem_rd <= '1';
                        c_ir_load <= '1';
                        c_adr_pc_not_reg <= '1';
                        if ready = '1' then
                                next_state <= s_id;
                        end if;

                when s_id =>

                        c_pc_inc <= '1';
                        if (ir(15 downto 14) = "00") then
                                next_state <= s_alu;
                        elsif (ir(15 downto 14) = "01") then
                                if   (ir(12 downto 11) = "00") then
                                        next_state <= s_ldil;
                                elsif (ir(12 downto 11) = "01") then
                                        next_state <= s_ldih;
                                elsif (ir(12 downto 11) = "10") then

                                                next_state <= s_ld1;

                                elsif (ir(12 downto 11) = "11") then
                                        next_state <= s_st1;
                                else
                                        next_state <= s_halt;
                                end if;
                        elsif (ir(15 downto 14) = "10") then
                                if (ir(12 downto 11) = "00") then
                                        next_state <= s_jmp;
                                elsif (ir(12 downto 11) = "10") then
                                        if(zero = '1') then
                                                next_state <= s_jmp;
                                        else
                                                next_state <= s_if1;
                                        end if;
                                elsif (ir(12 downto 11) = "11") then
                                        if(zero = '0') then
                                                        next_state <= s_jmp;
                                        else
                                                        next_state <= s_if1;
```

```
                                        end if;

                                else
                                        next_state <= s_halt;
                                end if;
                        else
                                next_state <= s_halt;
                        end if;

                when s_alu =>
                        next_state <= s_if1;
                        c_regfile_load_hi <= '1';
                        c_regfile_load_lo <= '1';
                when s_ldih =>
                        c_reg_ldi <= '1';
                        next_state <= s_if1;
                        c_regfile_load_hi <= '1';
                when s_ldil =>
                        c_reg_ldi <= '1';
                        next_state <= s_if1;
                        c_regfile_load_lo <= '1';

                when s_ld1 =>
                        if ready = '0'  then
                                next_state <= s_ld2;
                        end if;
                when s_ld2 =>
                        c_mem_rd <= '1';
                        c_reg_ldmem <= '1';
                        c_adr_pc_not_reg <= '0';
                        c_regfile_load_hi <= '1';
                        c_regfile_load_lo <= '1';
                        if ready = '1' then
                                next_state <= s_if1;
                        end if;

                when s_st1 =>
                        if ready = '0'  then
                                next_state <= s_st2;
                                end if;
                when s_st2 =>
                        c_mem_wr <= '1';
                        c_adr_pc_not_reg <= '0';
                        if ready = '1' then
                                next_state <= s_if1;
                        end if;
                when s_jmp =>
                        c_pc_load <= '1';
                        next_state <= s_if1;
                when s_halt =>
                        next_state <= s_halt;
                when others  =>
                        next_state <= s_reset;
        end case;
    end process;

end RTL;
```

1.3 Kommentierter Assembler Code von allen VISCY-CPU Befehlen.

```
        .org 0x0000 ; alles folgende start ab Adresse 0
        .start
        ldil            r1, 1
        ldih            r1, 0
        ; xor 0x100 =0
xor:
        ldil            r7,  0x00
        ldih            r7,  0x01 ; r7:=0x0100
        xor                     r0, r7, r7; r0:=0
        st                      [r7], r0 ; speichern r0 in adresse r7

        ;sal 0x101 = 0x10=16
sal:
        ldil            r0, 0x08; ; ro:= 0x08
        add             r7, r7, r1;  Zieladresse erhoehen
        sal             r0, r0; shift nach links
```

```
        st                      [r7], r0; speichern von ro in der Adresse r7

        ;sar 0x102= 0x08
sar:
        ld                      r2, [r7]; Laden des Wertes von der Speicheradresse r7 in r2
        sar             r0, r0; shift nach rechts
        add             r7, r7, r1; Zieladresse erhoehen
        st                      [r7], r0 ; speichern von ro in der adresse r7

        ;sub 0x103= 0xf00f -0x0ff0 = 0xE01F
sub:
        ldil            r0, 0x0f ;
        ldih            r0, 0xf0; r0 :=0xf00f
        ldil            r2, 0xf0
        ldih            r2, 0x0f; r2:=0x0ff0
        sub                     r0, r0, r2; ro:= r0-r2
        add             r7, r7, r1; Zieladresse erhoehen
        st                      [r7], r0; speichern von ro in der Adresse r7

        ; not 0x104= 0xf00f
not:
        not                     r0, r2 ; ro:= not r2
        add             r7, r7, r1; Zieladresse erhoehen
        st                      [r7], r0;speichern von ro in der Adresse r7

        ;and 0x105= 7
and:
        ldih            r2, 0 ;
        ldil            r2, 7 ; r2 := 7
        ldih            r3, 0xFF
        ldil            r3, 0xFF; r3 := 0xFFFF;
        and                     r0, r2, r3 ; r0:= r2 and r3
        add             r7, r7, r1; Zieladresse erhoehen
        st                      [r7], r0 ;speichern von ro in der Adresse r7

        ; or 0x106 = oxffff
or:
        ldil            r0, 0xAA
        ldih            r0, 0xAA ; r0:= 0xAAAA
        ldil            r2, 0x55
        ldih            r2, 0x55 ; r2:= 0x5555
        or                      r0, r0, r2; r0 := r0 or r2
        add             r7, r7, r1 ; Zieladresse erhoehen
        st                      [r7], r0 ; speichern von ro in der Adresse r7

        ;jmp|jz 0x107 = 5+4+3+2+1=15=0x0000f
        ldil            r2, 0x05
        ldih            r2, 0x00 ; r2:= ox0005
        xor                     r0, r0, r0 ; r0 := r0 xor r0
        ldil            r4, end & 255 ; r4:= end(Adresse)
        ldih            r4, end >> 8
        ldil            r3, loop & 255 ;r3:= loop (Sprungadresse)
        ldih            r3, loop >> 8
loop:

        jz                      r2, r4; spring auf end when r2 :=0
        add             r0, r0, r2; Zieladresse erhoehen
        sub             r2, r2, r1;  r2:=r2-1 ; Schleifenzaehler erniedrigen

        jmp             r3 ; spring nach loop
end:
        add             r7, r7, r1 ;Zieladresse erhoehen
        st                      [r7], r0 ; speichern von ro in der Adresse r7

        ;jnz 0x108 =2+1=0x0003
        ldil            r2, 0x02
        ldih            r2, 0x00 ; r2:= ox0002
        xor                     r0, r0, r0 ; r0:=0
loop1:
        add             r0, r0, r2 ;r0 := r0+r2
        sub             r2, r2, r1; r2:= r2+r1
        ldil            r3, loop1 & 255 ; r3:= loop (Sprungadresse)
        ldih            r3, loop1 >> 8
        jnz             r2, r3 ;  spring nach loop falls r2!=0

        add             r7, r7, r1 ;Zieladresse erhoehen
```

```
st                      [r7], r0 ; speichern von ro in der Adresse r7

halt        ;prozessor anhalten

.org 0x0100
.res 16
.end
```

1.4 Screenshot von Gtkwave.

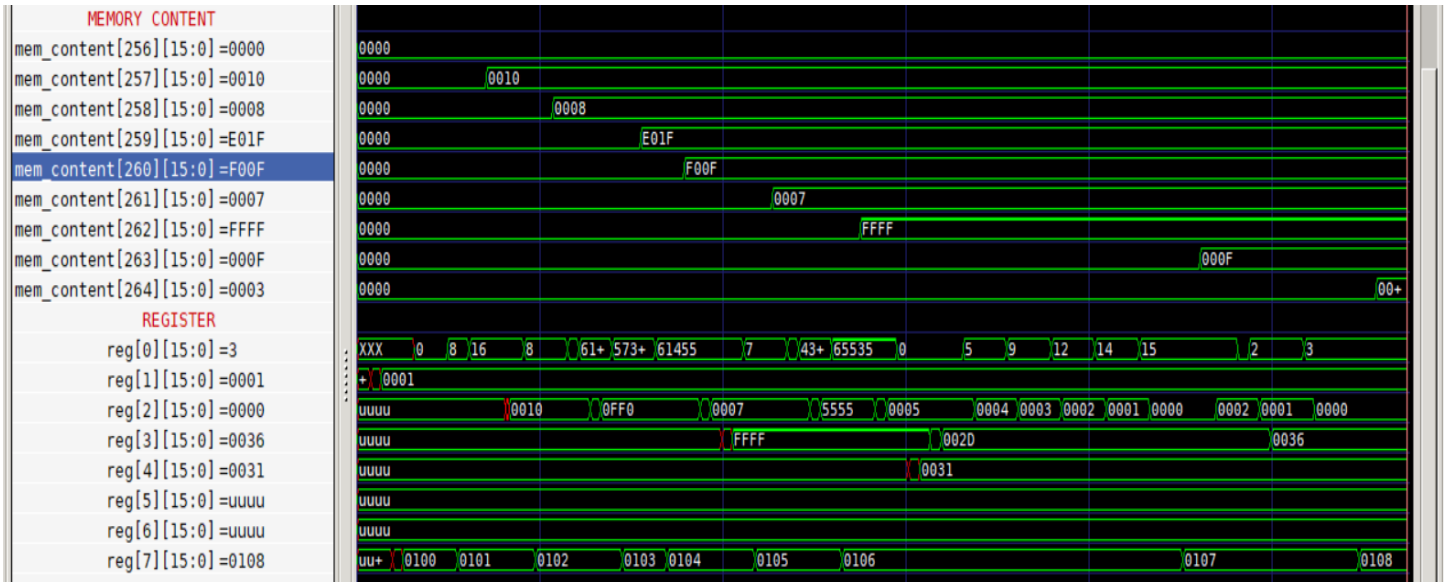

Abbildung 2: Gtkwave vom Befehlssatz

Da die Gtkwave-Form nicht übersichtlich ist, haben wir für jede Durchführung von Befehlssatz die Ergebnis in den Speicher angelegt.Für jeder Befehlssatz haben wir eine Label in Assembler-Code gemacht.

1.5 Screenshot des Schaltbildes ('xsch')

Anzahl Flipflops:	4
maximale Verzögerungszeit:	Konsole:2802 ps XSCH: 2235 ps

2 IC-Layout

2.1 Komplettes Chip

2.2 Standardzell-Bereich

3 Inbetriebnahme der PCU

3.1 Statistik

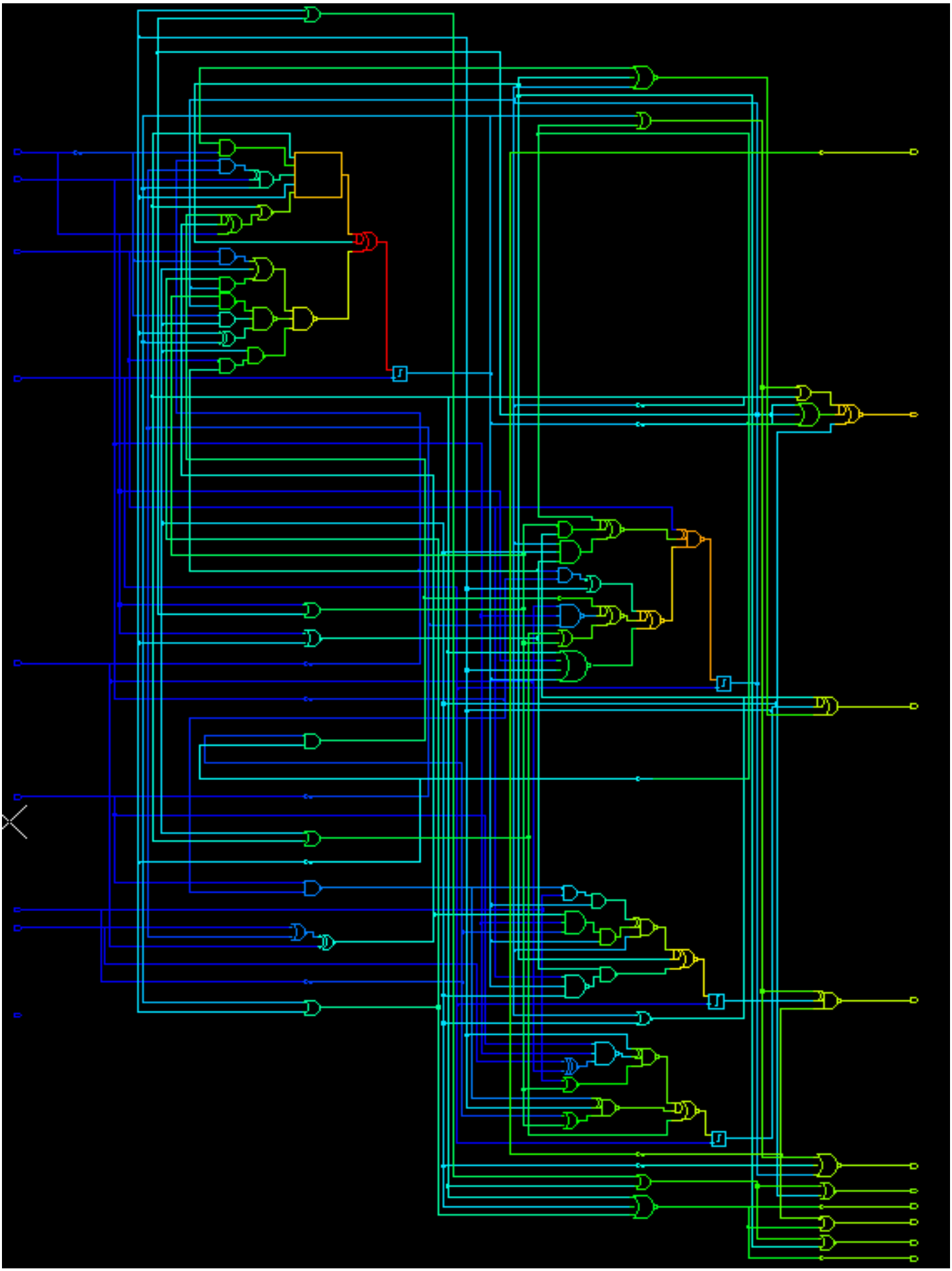

Abbildung 3: Steuerwerk Synthese

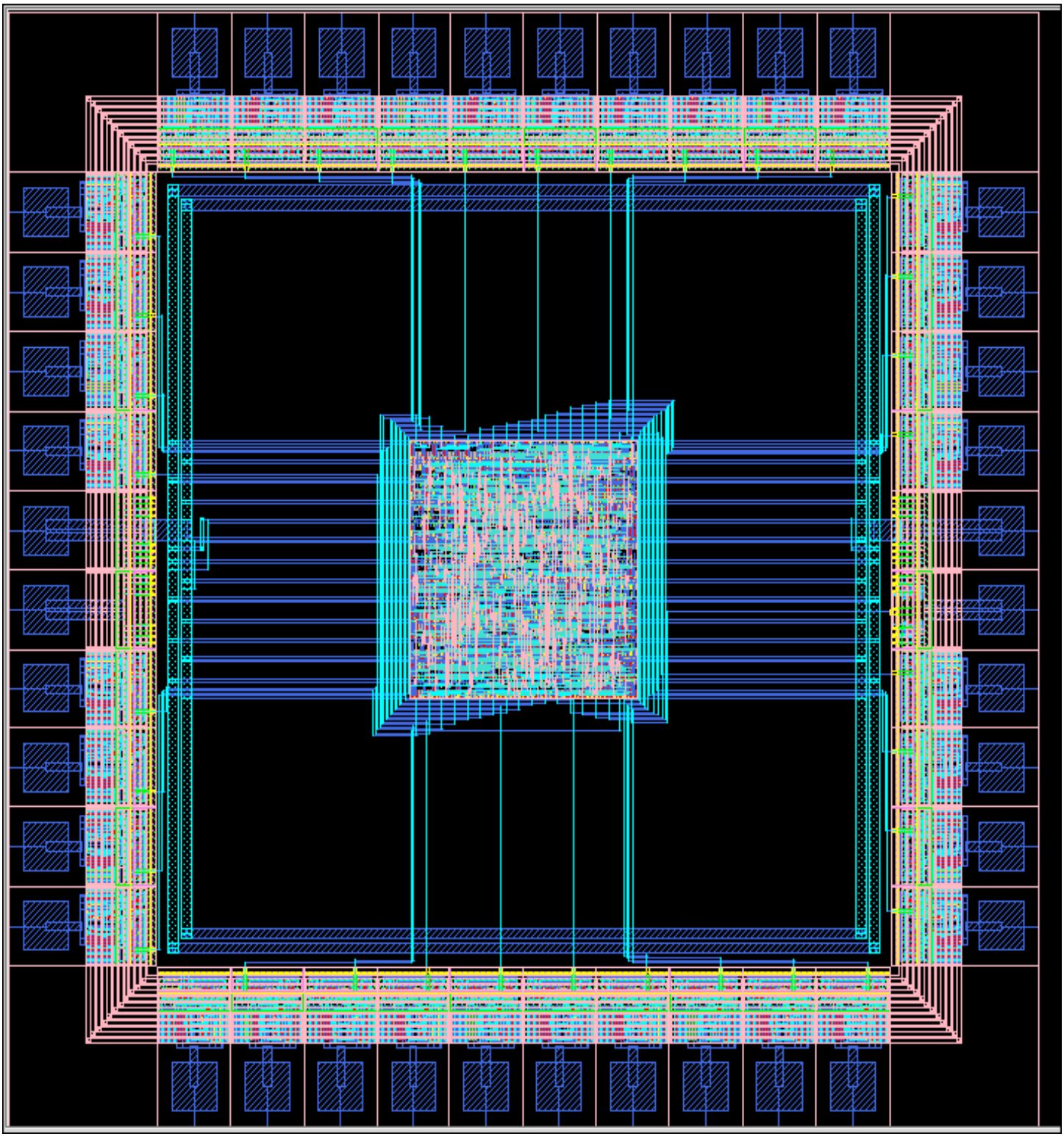

Abbildung 4: Chip Layout

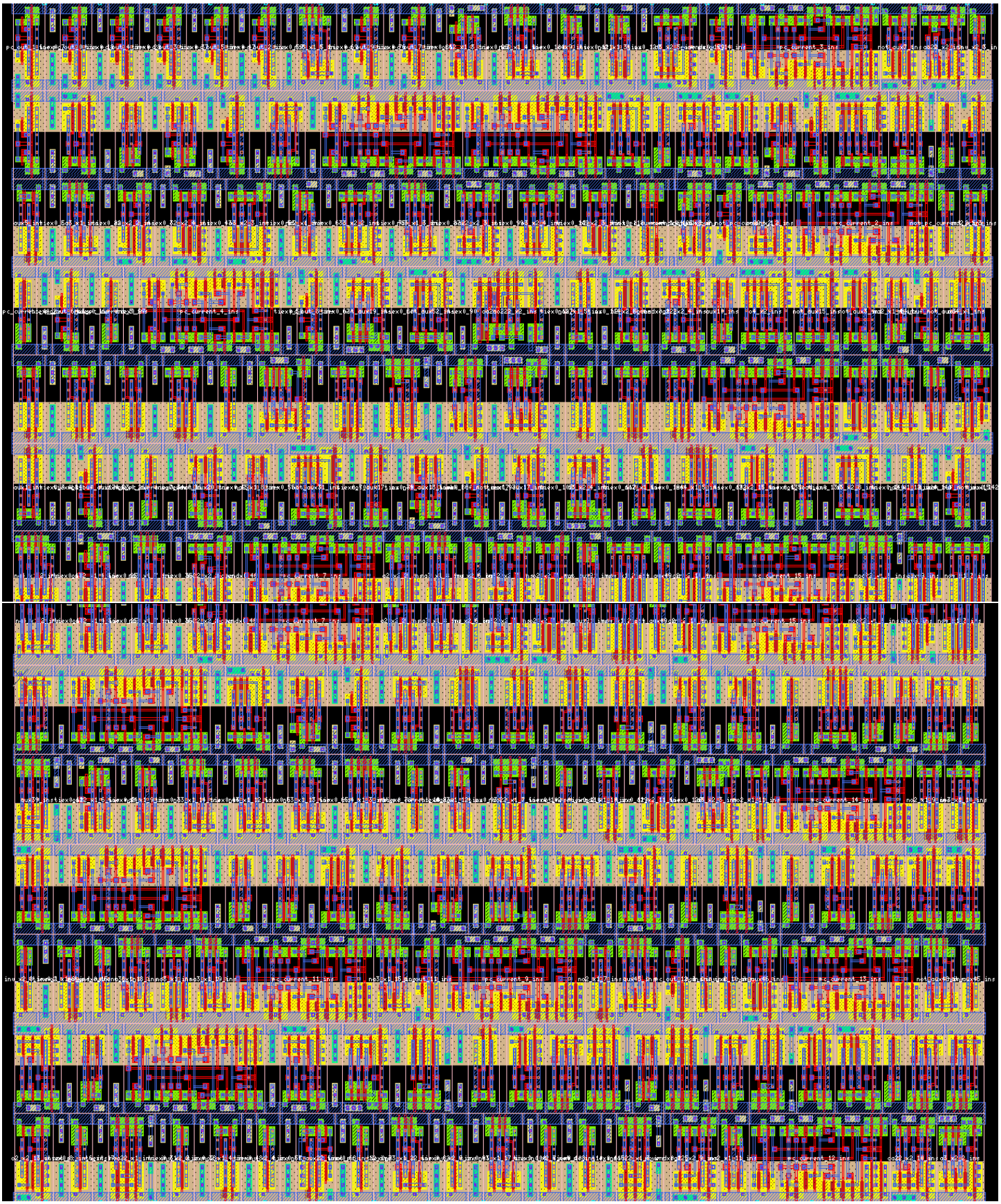

Abbildung 5: Standardzell-Bereich

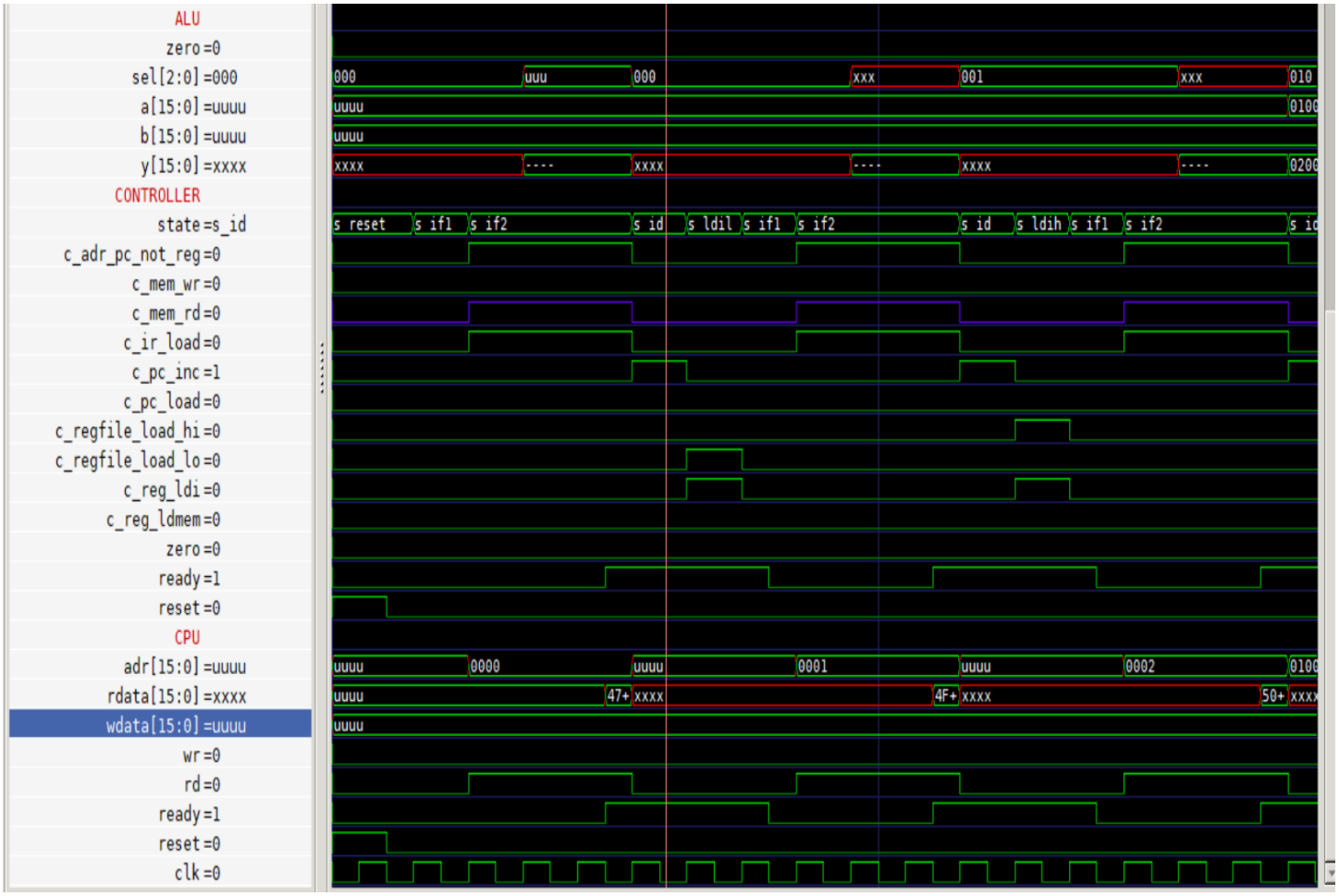

Abbildung 6: Die ersten drei Befehle

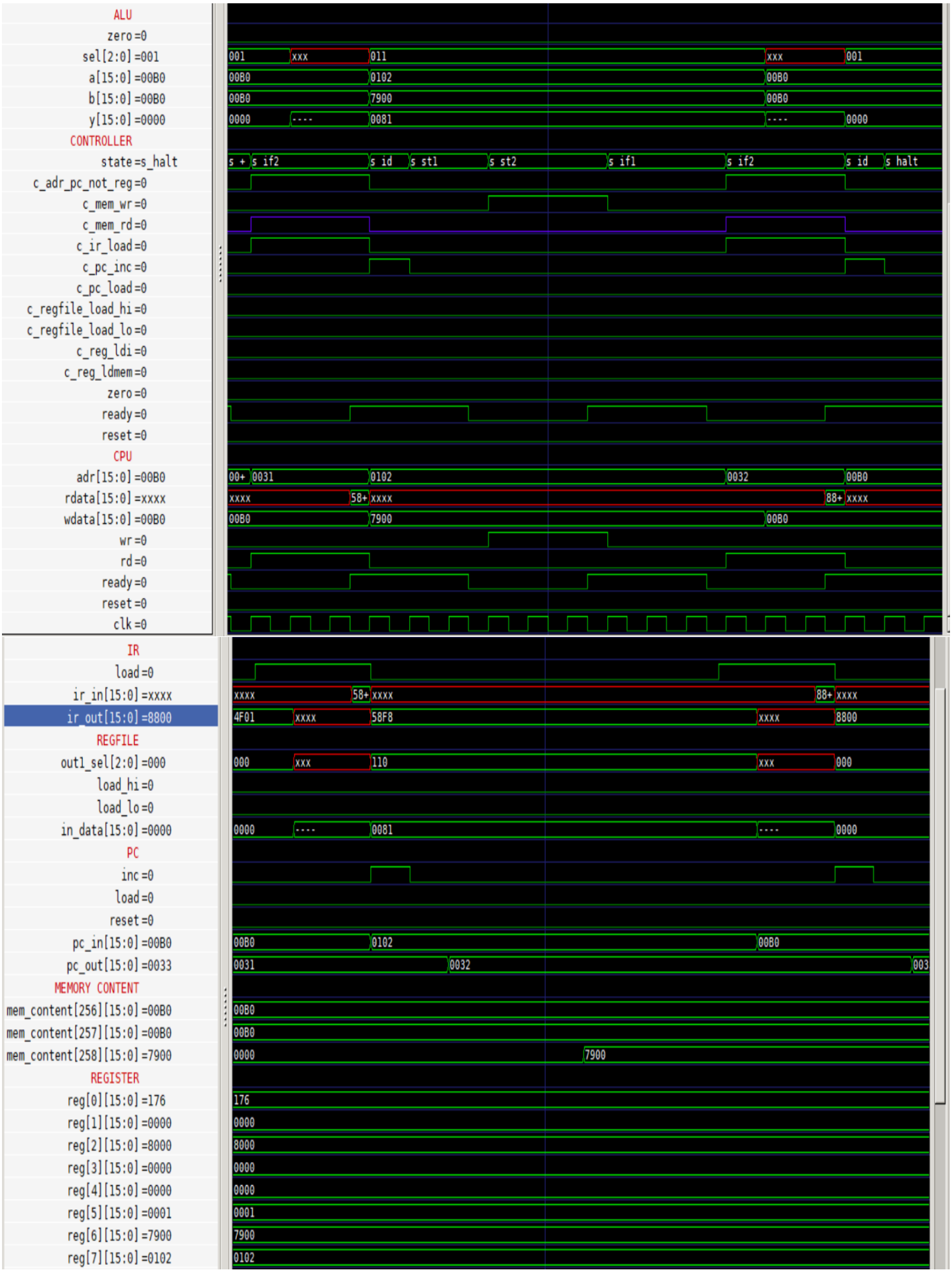

Abbildung 7: Der letzten drei Befehle (inklusive "halt")

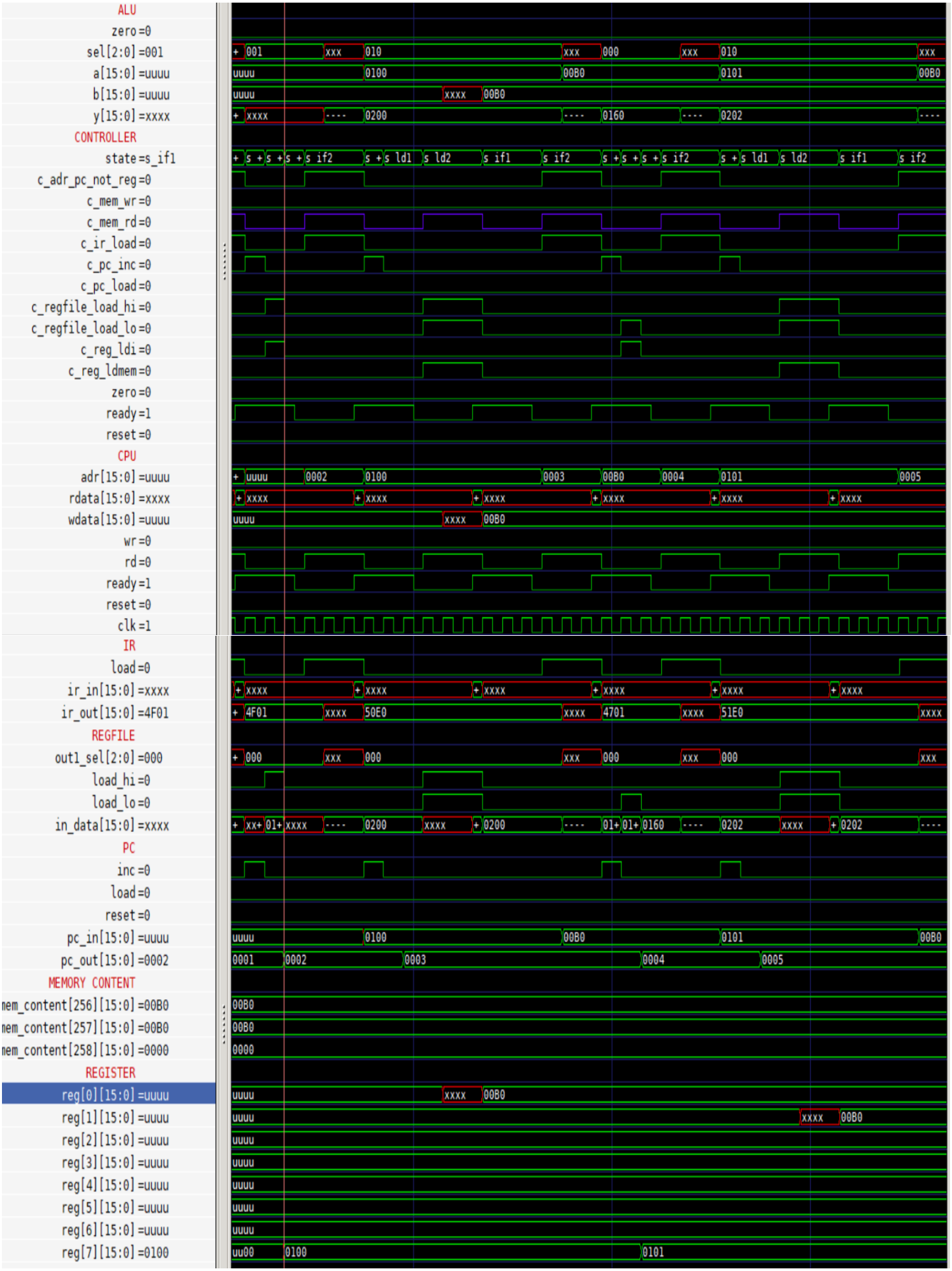

Abbildung 8: Befehle, die die Quelloperanden aus dem Speicher lesen