Zum eigenen Prozessor

UNTEN TEXT

Timo Grundheber und Yannik Stamm

Bitte helfen Sie mir, ich bin in Gefahr!

Byter

byter64

Über mich Keine Freunde

Games Engineering Bachelor
7. Semester
Tief gefallen (technische Informatik)
Gatze streicheln
Erfinder des Hans

Hallo

Message

Poseidon

t_i_mo

About Me Friends

Informatik Bachelor
7. Semester
3. Zeile (braucht 5 Zeilen im Profil)
Hat kein Problem mit League of Legends
Erfinder des Hans

Was haben wir gemacht?

- Konsole
- Befehlssatz
- Prozessor
- Assembler
- Linker & Loader
- Spiel

Ein Paar Kennzahlen

Prozessor

- 32-Bit Risc
- 20 MHz
- Multicycle
- 32 GP Register
- 32 Float Register
- Keine Interrupts

Grafik

- Auflösung: 160 x 120
- 256 Farben
- 8 LEDs

ULX3S - Top

USB 1
connected to
FTDI FT231XS

3.5 mm audio jack with 4
contacts (analog stereo + digital
audio or composite video)

GPDI
(digital video, audio,
ethernet ready)

USB 2
connected
directly to FPGA

25 MHz onboard,
external differential
clock input

DIP switches

Power button

8 user LEDs

32MB SDRAM

2 fire buttons

Lattice ECP5
LFE5U-85F-6BG381C
(85K LUT)

ADC: 8 channels,
12 bit, 1 MSa/s MAX11125

Micro-SD slot

2 USB LEDs
1 WiFi LED

Placeholder for 0.96”
SPI color OLED display

4 directional
buttons

Quelle: https://github.com/ulx3s/fpga-odysseus/tree/master/presentation

ULX3S - Top

USB 1
connected to
FTDI FT231XS

3.5 mm audio jack with 4
contacts (analog stereo + digital
audio or composite video)

GPDI
(digital video, audio,
ethernet ready)

USB 2
connected
directly to FPGA

25 MHz onboard,
external differential
clock input

DIP switches

Power button

8 user LEDs

2 fire buttons

32MB SDRAM

Lattice ECP5
LFE5U-85F-6BG381C
(85K LUT)

ADC: 8 channels,
12 bit, 1 MSa/s MAX11125

Micro-SD slot

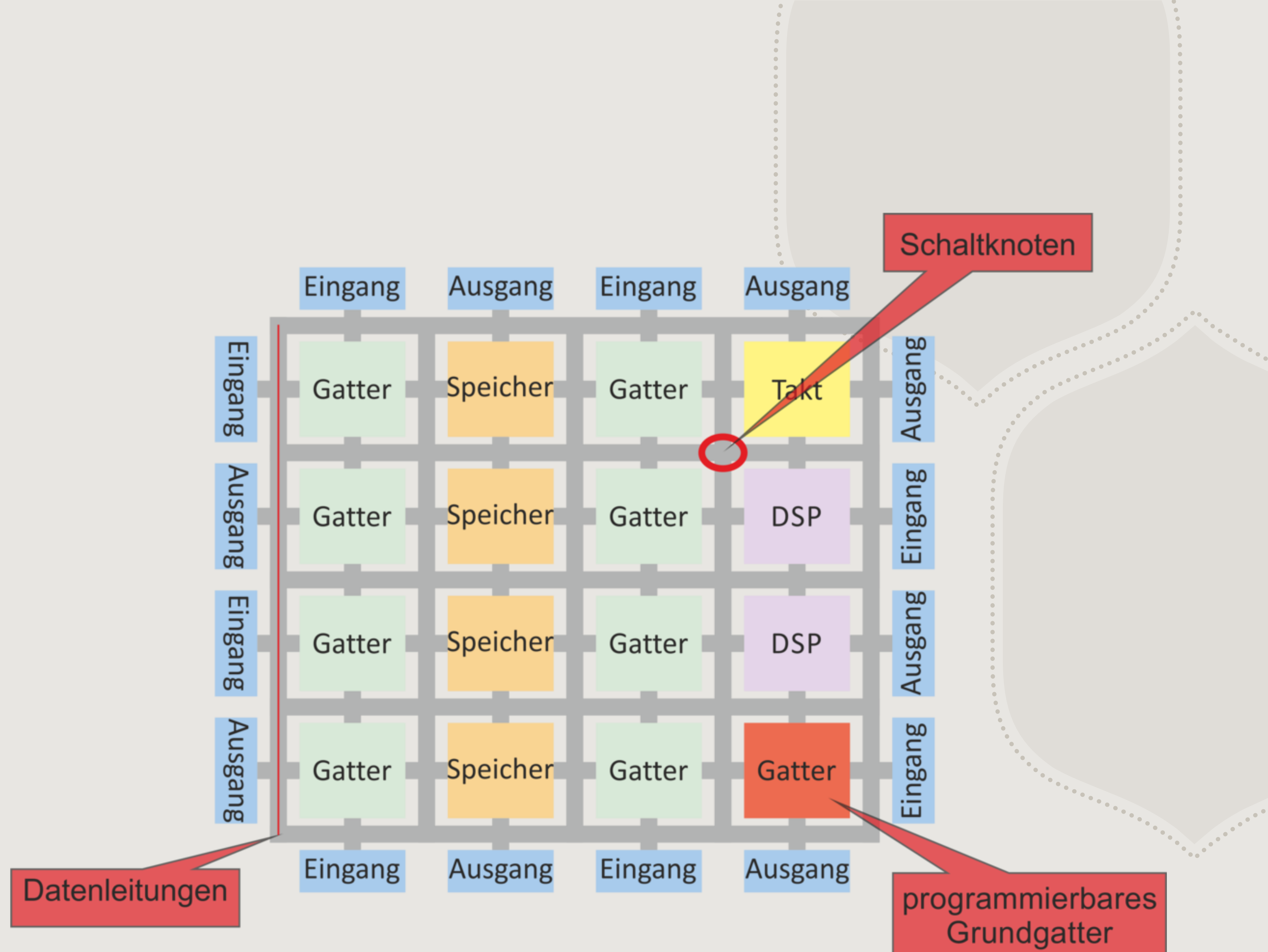
2 USB LEDs
1 WiFi LED

Placeholder for 0.96”
SPI color OLED display

4 directional
buttons

Quelle: https://github.com/ulx3s/fpga-odysseus/tree/master/presentation

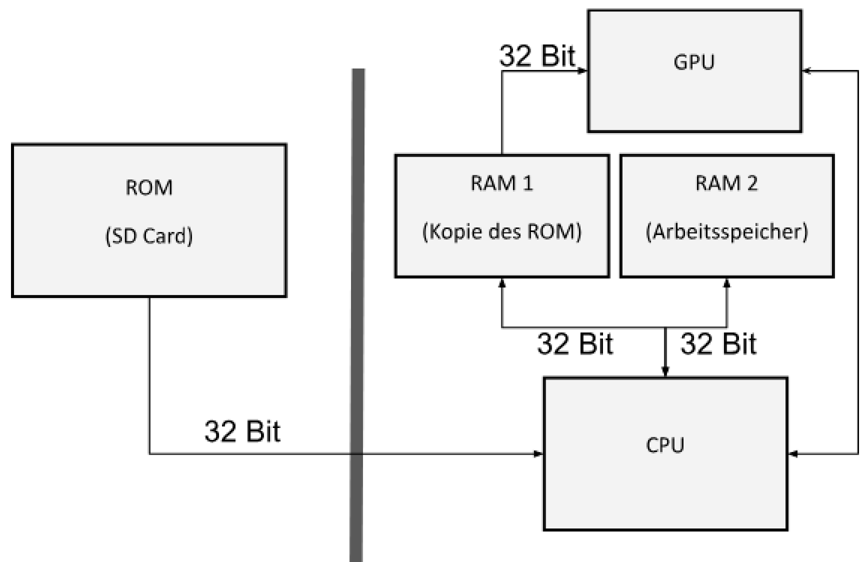
Was ist ein FPGA?



Quelle: https://www.fh-muenster.de/eti/labore_forschung/fd/fpga-design.php

32

Unsere Konsole: Plan 1



To-do-Liste

GPDI zu HDMI anschauen (irgendwer)
Liste Beschreibung einzelner Opcode befehle (Timo)

Nächste Schritte

1. Dokumentation aufräumen
2. Entwurfsartefakte erstellen (Blockdiagramme und co.) ← Do simma
3. Module aus dem Internet holen (für alles, wofür wir was finden)
4. evtl. Entwurfsartefakte überarbeiten
5. Testbenches schreiben (Autounfallkurs für Verilog und Git davor?)
6. Eigene Module schreiben
7. …
8. Profit

Einen Monat später:

Nächste Schritte

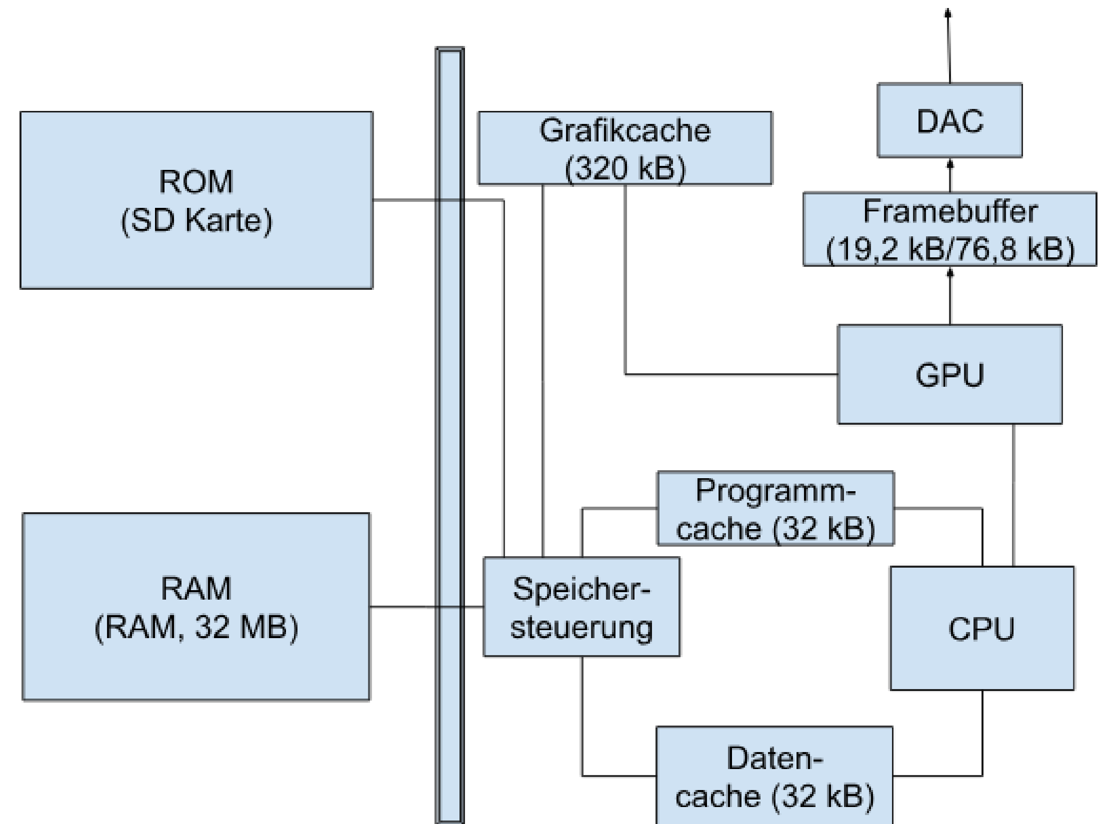
1. Testbenches schreiben ← Do Simma
2. Module aus dem Internet holen (für alles, wofür wir was finden)
3. Eigene Module schreiben
4. Mini-Assembler schreiben
5. …
6. Profit

Und noch einen Monat später:

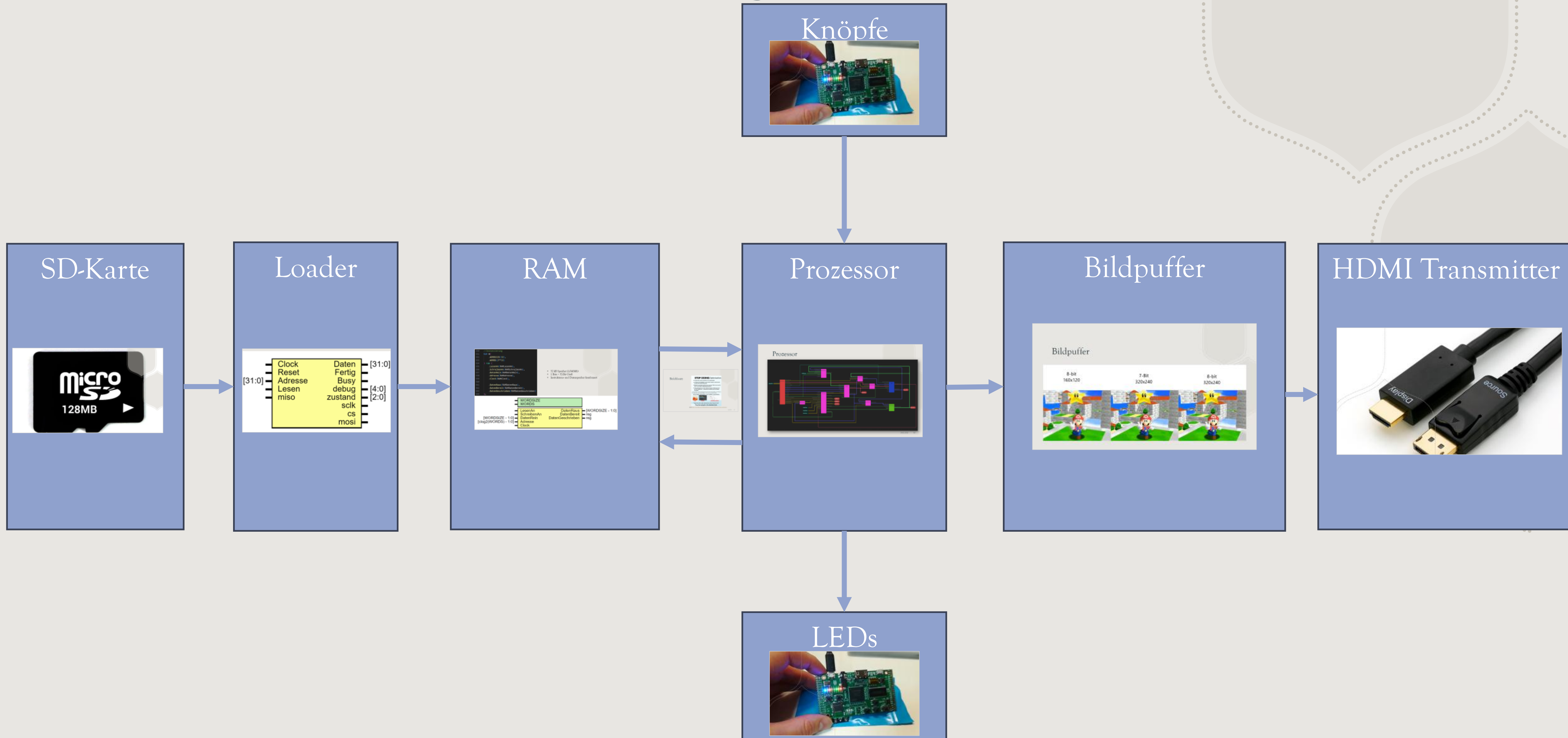
Nächste Schritte

1. Prozessor
   a. Prozessormodul testen ← Do is der Yannik
   b. Zeichnung für Prozessoraufbau aktualisieren (= neu machen)
2. HDMI-Wandler schreiben ← Do is der Dimo
3. CPU-Cache ← Do is der Moritz
4. Ram-Steuerung schreiben
5. …
6. Profit

Unsere Konsole: Plan 2.0

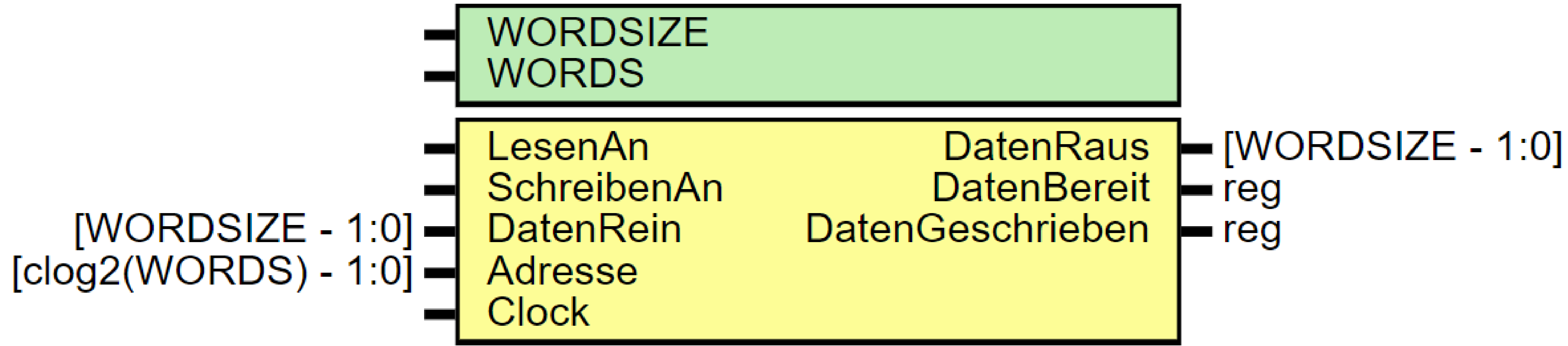


Unsere Konsole: Umsetzung



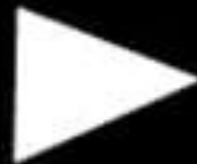
```
330    //Instanzierung
331    RAM #(
332        .WORDSIZE(32),
333        .WORDS(2**15)
334    ) ram (
335        .LesenAn(RAMLesenAn),
336        .SchreibenAn(RAMSchreibenAn),
337        .DatenRein(RAMDatenRein),
338        .Adresse(RAMAdresse),
339        .Clock(RAMClock),
340
341        .DatenRaus(RAMDatenRaus),
342        .DatenBereit(RAMDatenBereit),
343        .DatenGeschrieben(RAMDatenGeschrieben)
344    );
```

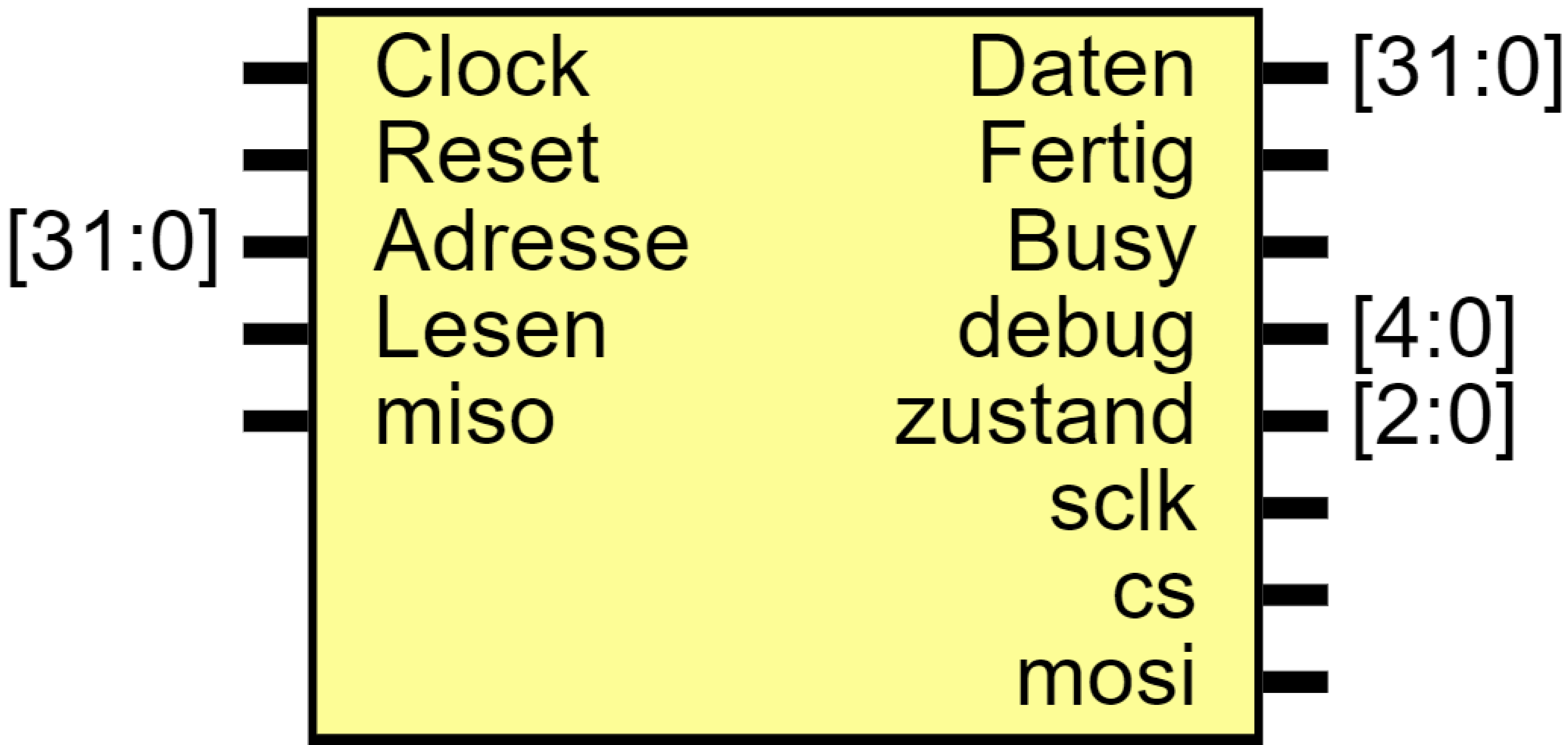
- 32 kB Speicher (1,048Mb)
- 1 Byte = 32-Bit Groß
- Instruktions- und Datenspeicher kombiniert

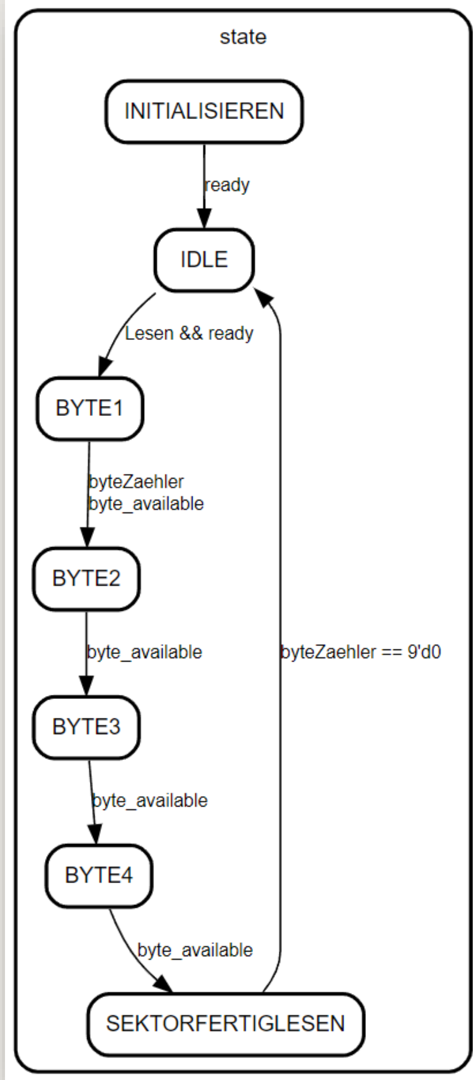


micro SD

128MB

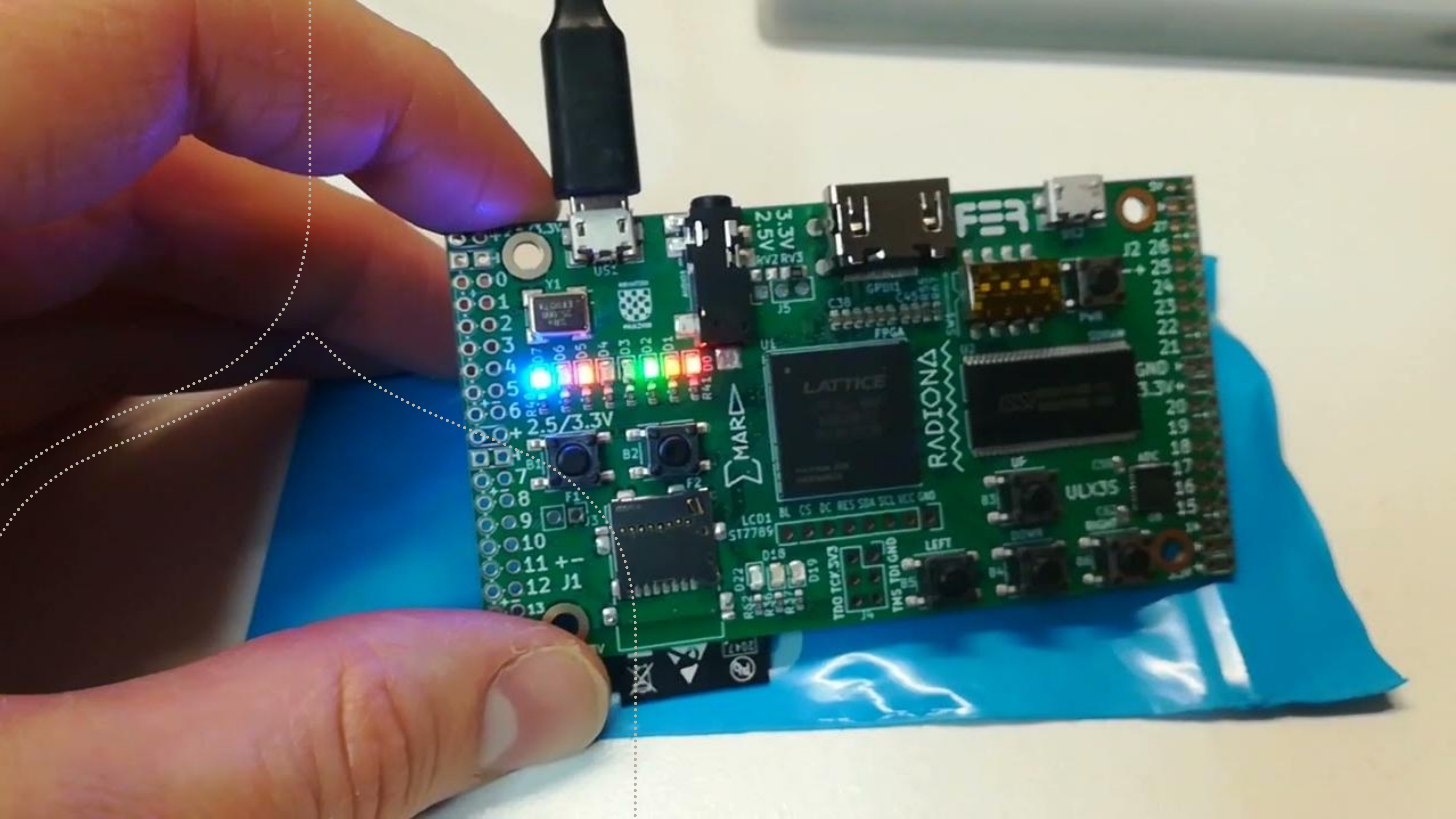



Clock
Reset
[31:0]
Adresse
Lesen
miso
Daten
[31:0]
Fertig
Busy
debug
[4:0]
zustand
[2:0]
sclk
cs
mosi



```
assign sektorAdresse = {Adresse[20:7], 9'b0};
```

```
byteZaehler <= byteZaehler + 1;
if (byteZaehler == {Adresse[6:0], 2'b00 }) begin
    Daten[31:24] <= dout;
    state <= BYTE2;
end
```



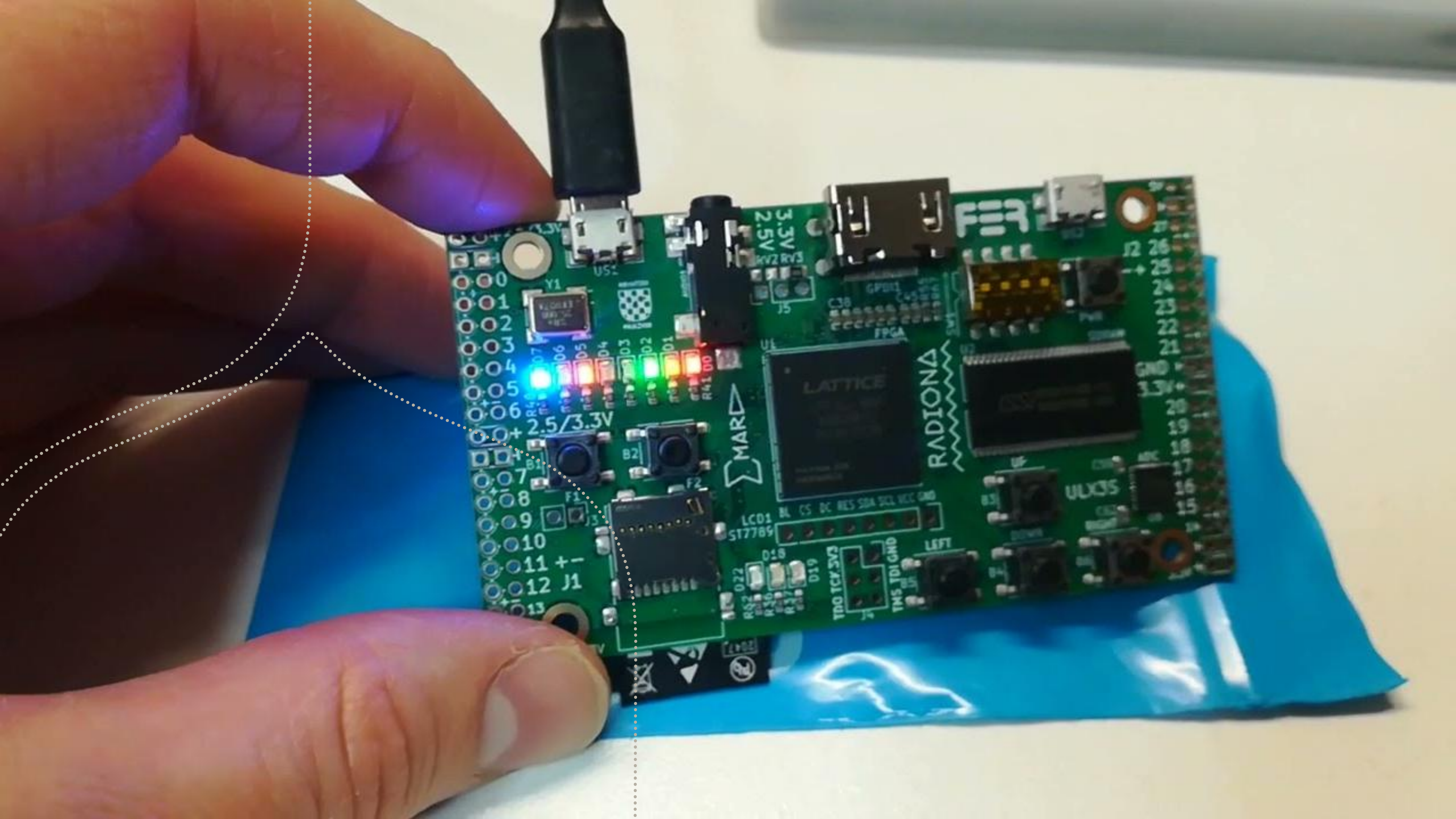
LATTICE
RADIONA
ULX3S
FPGA
2.5/3.3V
3.3V
2.5V
LCD1
ST7789
BL CS DC RES SDA SCL VCC GND
LEFT
GND
3.3V
J1
J2

LEDs

- Debuggen
  - Zustände
  - Ergebnisse
- Spielanzeige
- Sieht toll aus

```
38 //SonderBefehle
39   localparam SchreibeLEDS1        = 8'd1;
40   localparam SchreibeLEDS2        = 8'd2;
41   localparam SchreibeLEDS3        = 8'd3;
42   localparam SchreibeLEDS4        = 8'd4;
43   localparam SchreibeBP1          = 8'd5;
44   localparam SchreibeBP2          = 8'd6;
45   localparam SchreibeBPWechsel    = 8'd7;
46   localparam LadeKnoepfe          = 8'd8;
47
48   wire[7:0] SonderBefehl;
49   assign SonderBefehl = CPUDatenAdresse[31:24];
```

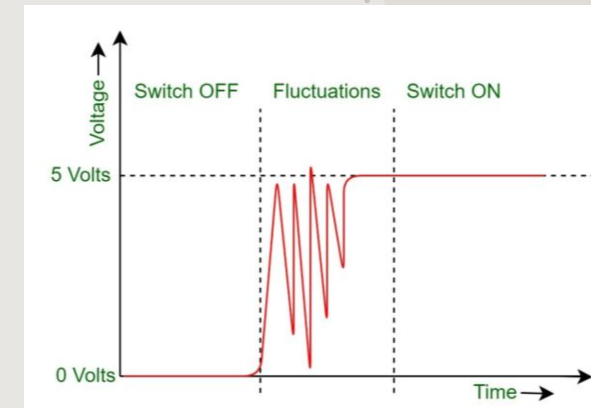
```
63 always @(posedge CPUClock) begin
64     if(CPUSchreibeDaten) begin
65         case(SonderBefehl)
66         SchreibeLEDS1: ledReg <= CPUDatenRaus[7:0];
67         SchreibeLEDS2: ledReg <= CPUDatenRaus[15:8];
68         SchreibeLEDS3: ledReg <= CPUDatenRaus[23:16];
69         SchreibeLEDS4: ledReg <= CPUDatenRaus[31:24];
70         endcase
71     end
72 end
```

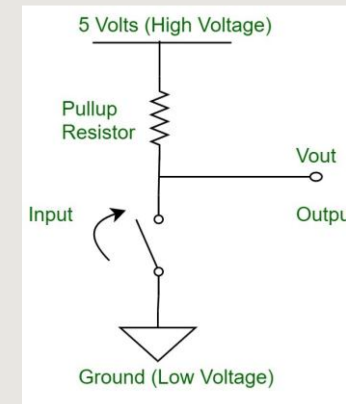


LATTICE
RADIONA
ULX3S
FPGA
3.3V
2.5V
2.5/3.3V
LCD1
ST7789
BL CS DC RES SDA SCL VCC GND
LEFT
J1
J2

Knöpfe

```
203  reg [19:0] KnopfdruckTimer[0:7];
204  integer KnopfZahl;
205  initial begin
206      for(KnopfZahl = 0; KnopfZahl < 7; KnopfZahl = KnopfZahl + 1) begin
207          KnopfdruckTimer[KnopfZahl] <= 0;
208      end
209  end
210  always @(posedge clk_25mhz) begin
211      for(KnopfZahl = 0; KnopfZahl < 7; KnopfZahl = KnopfZahl + 1) begin
212          if(KnopfdruckTimer[KnopfZahl] == 0 && (btn[KnopfZahl])) begin
213              Buttons[KnopfZahl] <= 1;
214              KnopfdruckTimer[KnopfZahl] <= 1;
215          end
216          else if(KnopfdruckTimer[KnopfZahl]!=0) begin
217              KnopfdruckTimer[KnopfZahl] <= KnopfdruckTimer[KnopfZahl] + 1;
218          end else begin
219              Buttons[KnopfZahl] <= 0;
220          end
221      end
222  end
```





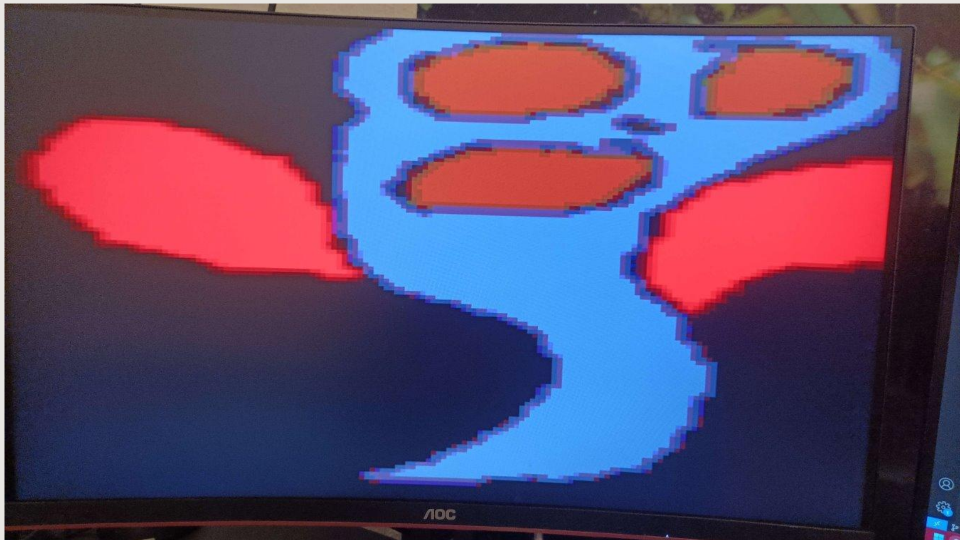

```
52   assign RAMLesenAn          = (zustand < RAMLADENBEENDEN) ? 1
53                              : (CPULeseInstruktion || CPULeseDaten);
54   assign CPUDatenGeladen     = SonderBefehl > 0 ? 1 : RAMDatenBereit;
55   assign CPUDatenRein        = (aktuelleInstruktion == 6'b111000 && SonderBefehl == LadeKnoepfe)?{26'b0,Buttons[6:1]}:RAMDatenRaus;
```

Bildpuffer

8-bit
160x120

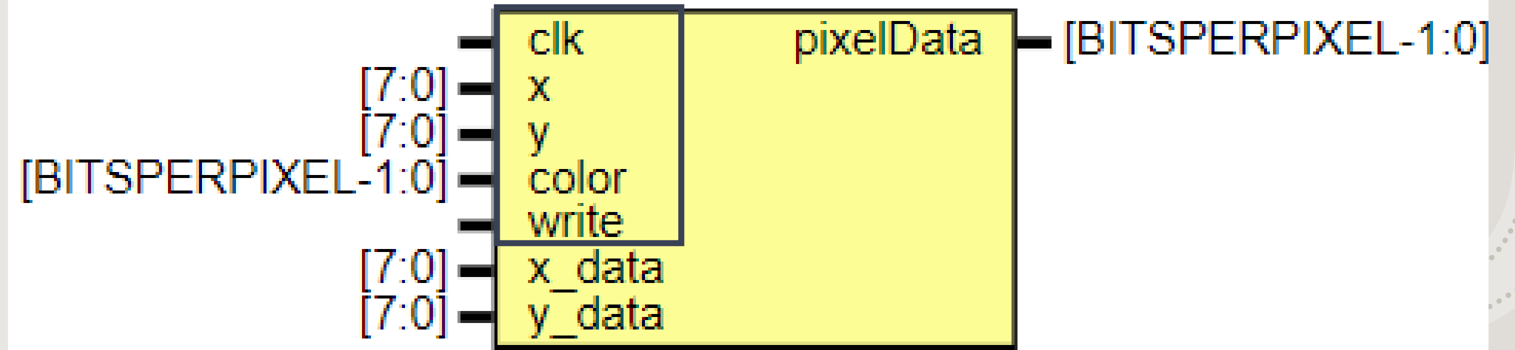
7-Bit
320x240

8-bit
320x240



AOC

Bildpuffer



- 160x120p
- 8-bit Farbtiefe
- 2 Bildpuffer
- Bildpuffer Wechsel per Memory Command (Angezeigt/Beschrieben)
- 2*19,2kB -> 38,4 kB (307,2kb) (Diesmal 1 Byte = 8 Bit)

Display
Source

GPDI

- General Purpose Differential Interface
- DVI-D Protokol

```
58    assign x = CounterX>>2;
59    assign y = CounterY>>2;
60
61    reg [7:0] red, green, blue;
62
63    always @(posedge pixclk) begin
64        red   <= {pixelData[7:5],5'b0};
65        green <= {pixelData[4:2],5'b0};
66        blue  <= {pixelData[1:0],6'b0};
67    end
```

```
44   /******** Video generation *****************************************************/
45   // This part is just like a VGA generator
46   reg [9:0] CounterX, CounterY;
47   reg hSync, vSync, DrawArea;
48   always @(posedge pixclk) DrawArea <= (CounterX<640) && (CounterY<480);
49
50   always @(posedge pixclk) CounterX <= (CounterX==799) ? 0 : CounterX+1;
51   always @(posedge pixclk) if(CounterX==799) CounterY <= (CounterY==524) ? 0 : CounterY+1;
52
53   always @(posedge pixclk) hSync <= (CounterX>=656) && (CounterX<752);
54   always @(posedge pixclk) vSync <= (CounterY>=490) && (CounterY<492);
```

2.3.1. Minimum Frequency Supported

The minimum frequency supported is specified to allow the link to differentiate between an active low-pixel format link and a power managed state (inactive link). The lowest pixel format required by the DVI specification is 640x480@60 Hz (clock timing of 25.175 MHz). The DVI link can be considered inactive if the T. M. D.S. clock transitions at less than 22.5 MHz for more than one second.

https://glenwing.github.io/docs/DVI-1.0.pdf

https://www.fpga4fun.com/HDMI.html

Befehlssatz

STOP DOING Optimization

- Code was never meant to be optimized
- YEARS OF OPTIMIZING yet NO REAL-WORLD USE FOUND for BETTER PERFORMANCE
- Wanted to get better performance anyways? We had a tool for that, it was called "Upgrading hardware"
- "Yes please give me a low memory footprint. Please give me 5% CPU utilization" - Statements dreamed up by the utterly Deranged

Look at what Low-level programmers have been demanding your Respect for all this time, with all the RAM & CPU cores we built for them

(This is REAL optimizations, done by REAL programeers):

?????

???????

```
x2 = number * 0.5F;
y  = number;
i  = * ( long * ) &y;
i  = 0x5f3759df - ( i >> 1 );
y  = * ( float * ) &i;
y  = y * ( threehalfs - ( x2 * y * y ) );
```

?????????????????

"I spent the entire week reducing the system latency by 2ms"

They have played us for absolute fools

Quelle: https://www.reddit.com/r/StopDoingScience/comments/10fv0ws/stop_doing_optimization/

Designziele

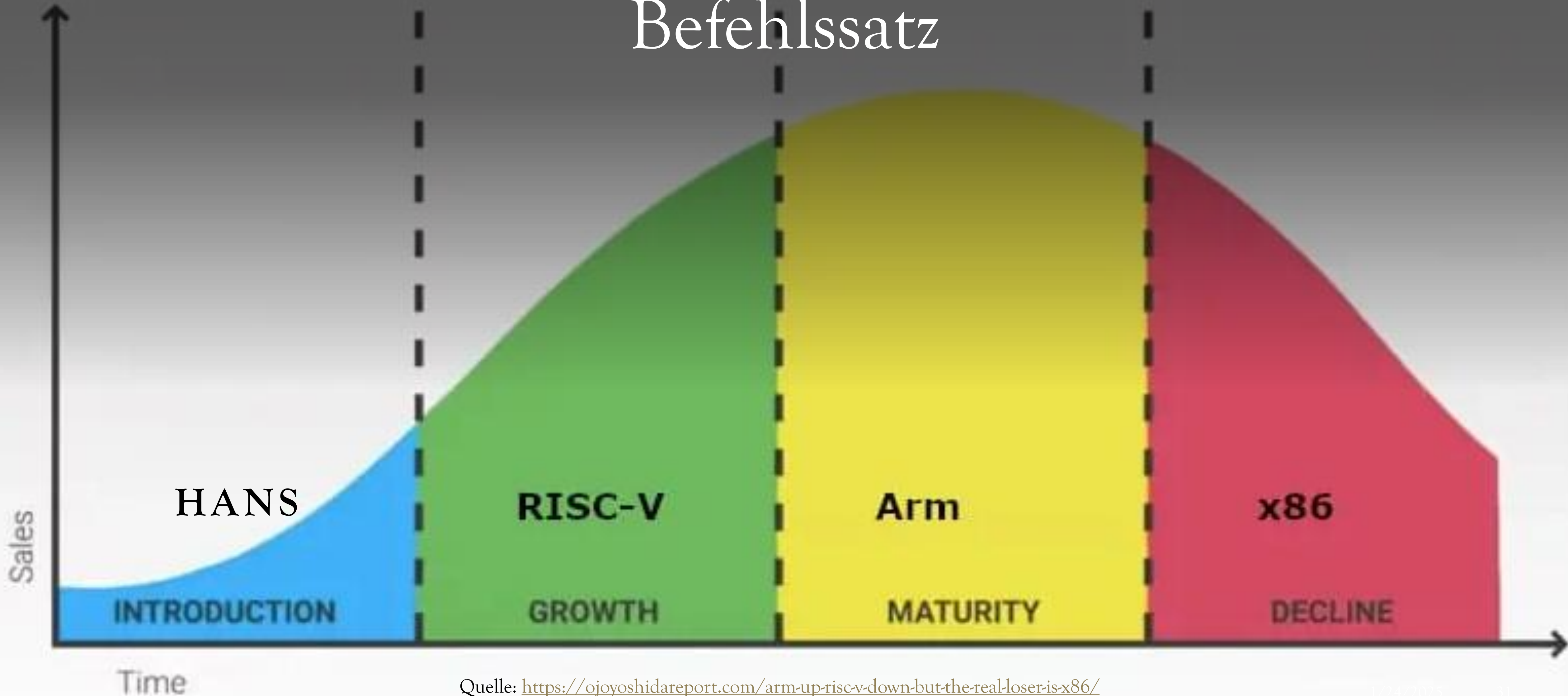
- RISC
- 32-Bit (Yannik findet das ganz wichtig)
- Spiele Mechaniken durch Hardwarebeschleuniger optimieren
- "Einfach"

RISC vs. CISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

Quelle: https://microcontrollerslab.com/difference-between-risc-and-cisc/

Befehlssatz



Quelle: https://ojoyoshidareport.com/arm-up-risc-v-down-but-the-real-loser-is-x86/

Planung

Befehle des Prozessors

Arithmetik

Skalaroperationen

- Addieren (int, float)
- Subtrahieren (int, float)
- Multiplizieren (int, float)
- Dividieren (int, float)
- Quadratwurzel (int, float)
- Sinus (float)
- Tangens (float)

Vektoroperationen

- Addieren (int, float)
- Subtrahieren (int, float)
- Multiplizieren mit Skalar (int, float)
- Dividieren mit Skalar (int, float)
- Vektorlänge berechnen (int, float)
- Vektor rotieren (float)

Vergleiche

- Gleich (int, float)
- Ungleich (int, float)
- Kleiner (int, float)
- Kleinergleich (int, float)
- Größer (int, float)
- Größergleich (int, float)

Logik

- Or (int)
- And (int)
- Linksschift (int)
- Rechtsschift (int)
- Zyklischer Linksschift (int)
- Zyklischer Rechtsschift (int)

Speicherzugriff

- Load
- Store
- LoadEx

Springen

- Branch equal zero (relativ)
- Branch not equal zero (relativ)
- Branch overflow (relativ)
- Jump (relativ)
- Jump and Link (relativ)
- Jump register (absolut)

Interrupt

- Set overflow
- Get overflow
- Enable interrupts
- Disable interrupts
- Set relative interrupt service routine Adresse
- Return from interrupt service routine
- Set software interrupt
- Get Software interrupt

Anderes

- Random Number Generator
- Leere Operationsbefehl
- Rechteckskollision (int, float)
- Kreiskollision (int, float)
- Rechteckkreiskollision(int, float)
- GPUlauf starten

	Operation	Int	Float
Skalaroperationen			
	Addieren	X	X
	Subtrahieren	X	X
	Multiplizieren	X	X
	Dividieren	X	X
	Quadratwurzel	X	X
	Sinus		X
	Tangens		X
Vektoroperationen			
	Addieren	X	X
	Subtrahieren	X	X
	Multiplizieren (Skalar)	X	X
	Dividieren (Skalar)	X	X
	Vektorlänge	X	X
	Vektor rotieren		X
Vergleiche			
	Gleich	X	X
	Ungleich	X	X
	Kleiner	X	X
	Kleiner gleich	X	X
	Größer	X	X
	Größer gleich	X	X
Logik	Or	X	
	And	X	
	Linksschift	X	
	Rechtsschift	X	
	Zyklischer Linksschift	X	
	Zyklischer Rechtsschift	X	
Speicherzugriff			
	Load		
	Store		
	LoadEx		
Springen		Relativ	Absolut
	Branch equal zero	X	
	Branch not equal zero	X	
	Branch overflow	X	
	Jump	X	
	Jump and Link	X	
	Jump register		X
Interrupt			
	Set overflow		
	Get overflow		
	Enable interrupts		
	Disable interrupts		
	Set relative interrupt service routine Adresse		
	Return from interrupt service routine		
Anderes		Int	Float
	Random Number Generator		
	Leerer Operationsbefehl		
	Rechteckskollision	X	X
	Kreiskollision	X	X
	Rechteck-Kreiskollision	X	X
	GPUlauf starten		

Anzahl Operationen: 66

Umsetzung

OPCODE

Wichtigkeit	Bit	1	2	3	4	5	6
1							Interrupts
2					J	J	
3					JAL	BOV	
4				Immediate			
5			Register				
6		Float					

Opcode Zuweisung

Bit		
1	Immediate	
2	JAL	J
3	BOV	
4	Null-Parameter	
5	Register	
6	Float	

Die Bits werden in aufsteigender (1 → 6) Reihenfolge ausgewertet.

Immediate-Format (I):

OPCODE	Z-Register	Q-Register	IMM (vorzeichenbehaftet)
6-Bits	5-Bits	5-Bits	16-Bits

Planung

Byter 23.12.2022 14:29

@Poseidon OP-Codebeauftragter, kannst du dir folgenden Link mal anschauen? https://en.wikipedia.org/wiki/MIPS_architecture#MIPS_I
https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html (Der Link hat denselben Inhalt, aber ein bisschen anders)
Die Mips hat ne sehr schöne Struktur für ihre Befehle und kriegt dadurch 32 int register, 16 float register und 16 Bit immediate Operand hin.

MIPS architecture

MIPS (Microprocessor without Interlocked Pipelined Stages) is a family of reduced instruction set computer (RISC) instruction set architectures (ISA):A-1: 19 developed by MIPS Computer Systems, now MIPS Technologies, based in the United States.
There are multiple versions of MIPS: including MIPS I, II, III, IV, and V; as well as five releases...

Poseidon 23.12.2022 14:31

Ok

Umsetzung

Allgemeiner Aufbau von Befehlen

Alle Befehle haben eine Länge von einem Wort und werden in 4 verschiedene Kategorien aufgeteilt:

Register-Format (R):

OPCODE	Z-Register	Q1-Register	Q2-Register	Schieben	Funktion
6-Bits	5-Bits	5-Bits	5-Bits	5-bits	6-bits

Immediate-Format (I):

OPCODE	RS	RT	IMM
6-Bits	5-Bits	5-Bits	16-Bits

Jump-Format (J):

OPCODE	Pseudo-Adresse
6-Bits	26-Bits

Null-Parameter-Format (N):

OPCODE	Befehl
6-Bits	26-Bits

Register-Format (R):

OPCODE	Z-Register	Q1-Register	Q2-Register	Platzhalter	Funktion
6-Bits	5-Bits	5-Bits	5-Bits	5-bits	6-bits

Immediate-Format (I):

OPCODE	Z-Register	Q-Register	IMM (vorzeichenbehaftet)
6-Bits	5-Bits	5-Bits	16-Bits

Jump-Format (J):

OPCODE	Relative Adresse (vorzeichenbehaftet)
6-Bits	26-Bits

Zuweisung der “OPCode”-Bits

Code	Bedeutung	Beschreibung
00 xxxx	Register-Format	Markierung für Registerbefehle
01 xxxx	Jump-Format	Markierung für Jumpbefehle
1 xxxxx	Immediate-Format	Markierung für Immediatebefehle

Finale Version

Opcode

Bits [31:30]	Format
00	Registerformat
01	Jumpformat
1x	Immediateformat

Register Format

Bits	-	31:30	29:26	25:21	20:16	15:11	10:6	5:0
Zweck	-	Format	frei	Z-Register	Q1-Reg	Q2-Reg	frei	Funktion
Größe	-	2-Bits	4-Bits	5-Bits	5-Bits	5-Bits	5-Bits	6-Bits

Immediate Format

Bits	-	31:31	30:26	25:21	20:16	15:0
Zweck	-	Format	Funktion	Z-Register	Q-Reg	Immediate
Größe	-	1-Bit	5-Bits	5-Bits	5-Bits	16-Bits

Jump Format

Bits	-	31:30	29:26	25:0
Zweck	-	Format	OP-Code	Immediate
Größe	-	2-Bits	4-Bits	26-Bits

Kurzname	Typ	Bitcode	Kurzbeschreibung
Jmp	J	010000 xxxxxxxxxxxxxxxxxxxxxxxxxx	PC := PC + IMO

Befehle
Arithmetik

Kurzname	Typ	Bitcode	Kurzbeschreibung
Add	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0000	Z := Q1 + Q2
Sub	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0001	Z := Q1 - Q2
Mul	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0010	Z := Q1 · Q2
Sqrt	R	000000 xxxxx xxxxx zzzzz zzzzz 00 0011	Z := √Q1
Div	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0100	Z := Q1 : Q2
Mod	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0101	Z := Q1 mod Q2
Sla	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0110	Z := Q1 · 2^(Q2)
Sra	R	000000 xxxxx xxxxx xxxxx zzzzz 00 0111	Z := Q1 : 2^(Q2)
Ce	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1000	Z := Q1 == Q2
Cne	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1001	Z := Q1 != Q2
Cg	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1010	Z := Q1 > Q2
Cl	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1011	Z := Q1 < Q2
Cgu	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1100	Z := Q1 > Q2 (Hinweis: Q1 und Q2 werden als vorzeichenlose Ganzzahlen interpretiert)
Clu	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1101	Z := Q1 < Q2 (Hinweis: Q1 und Q2 werden als vorzeichenlose Ganzzahlen interpretiert)

Befehle

Bitmanipulation

Kurzname	Typ	Bitcode	Kurzbeschreibung
Not	R	000000 xxxxx xxxxx zzzzz zzzzz 01 0000	Z := ~Q1
And	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0001	Z := Q1 & Q2
Or	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0010	Z := Q1 | Q2
Xor	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0011	Z := Q1 ^ Q2
Xnor	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0100	Z := !(Q1 ^ Q2)
Sll	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0110	Z := Q1 << Q2
Srl	R	000000 xxxxx xxxxx xxxxx zzzzz 01 0111	Z := Q1 >> Q2

Befehle

Floating Point Operations

Kurzname	Typ	Bitcode	Kurzbeschreibung
Add.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 0000	Zf := Qf1 + Qf2
Sub.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 0001	Zf := Qf1 - Qf2
Mul.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 0010	Zf := Qf1 · Qf2
Squrt.s	R	000000 xxxxx xxxxx zzzzz zzzzz 10 0011	Zf := √Qf1
Div.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 0100	Zf := Qf1 : Qf2
Ce.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1000	Z := Qf1 == Qf2
Cne.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1001	Z := Qf1 != Qf2
Cg.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1010	Z := Qf1 > Qf2
Cl.s	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1011	Z := Qf1 < Qf2

Befehle Immediate

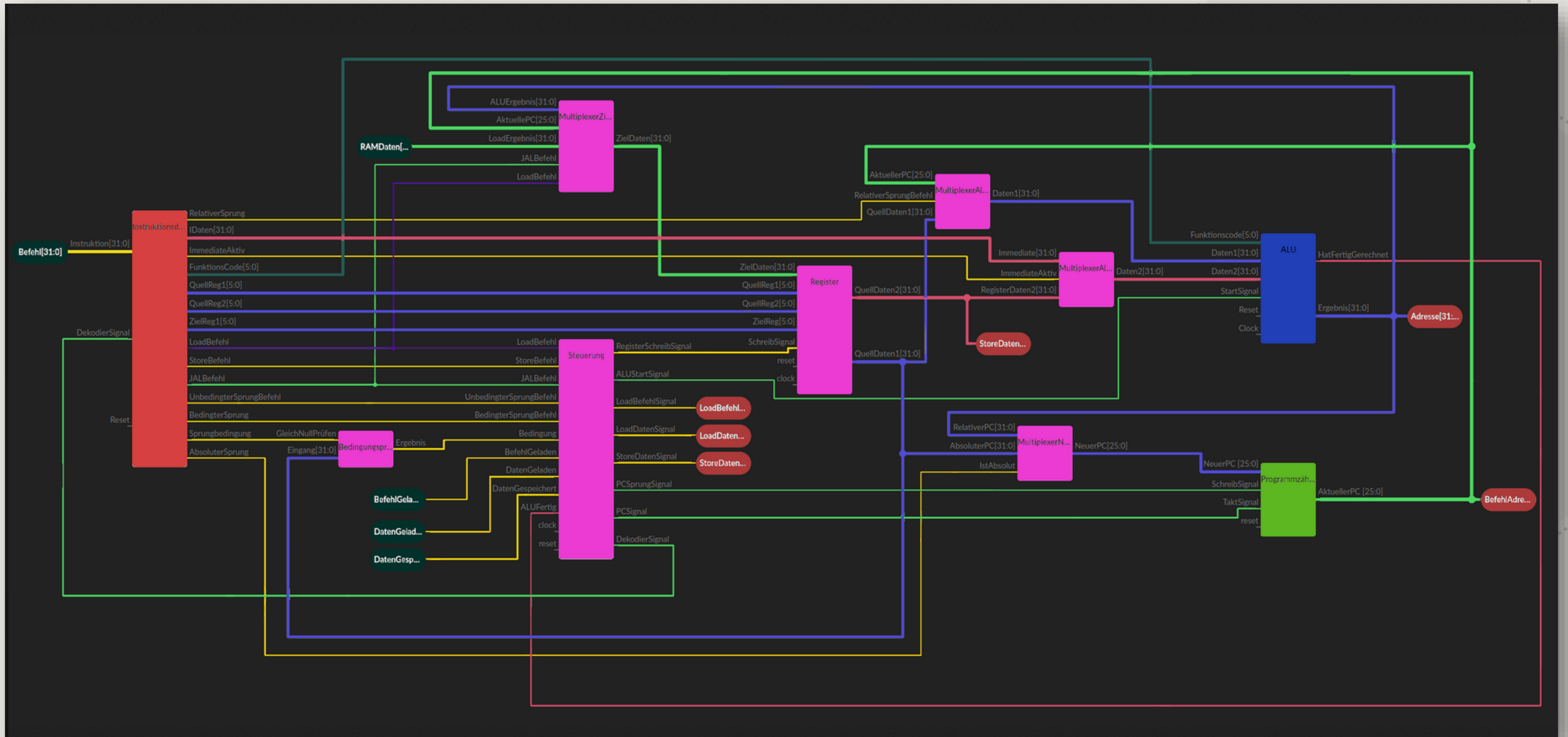
Kurzname	Typ	Bitcode	Kurzbeschreibung
Addi	I	100000 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 + IMO
Subi	I	100001 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 - IMO
Muli	I	100010 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 · IMO
Divi	I	100100 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 : IMO
Modi	I	100101 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 mod IMO
Slai	I	100110 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 · 2^(IMO)
Srai	I	100111 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 : 2^(IMO)
Cei	I	101000 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 == IMO
Cnei	I	101001 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 != IMO
Cgi	I	101010 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 > IMO
Cli	I	101011 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 < IMO
Cgui	I	101100 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 > IMO (Hinweis: Q1 wird als vorzeichenlose Ganzzahl interpretiert)
Clui	I	101101 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 < IMO (Hinweis: Q1 wird als vorzeichenlose Ganzzahl interpretiert)
Addis	I	110000 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 + (IMO << 16)
Andi	I	110001 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 & IMO
Ori	I	110010 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 | IMO
Xori	I	110011 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 ^ IMO
Xnori	I	110100 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := !(Q1 ^ IMO)
Slli	I	110110 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 << IMO
Srli	I	110111 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := Q1 >> IMO

Befehle

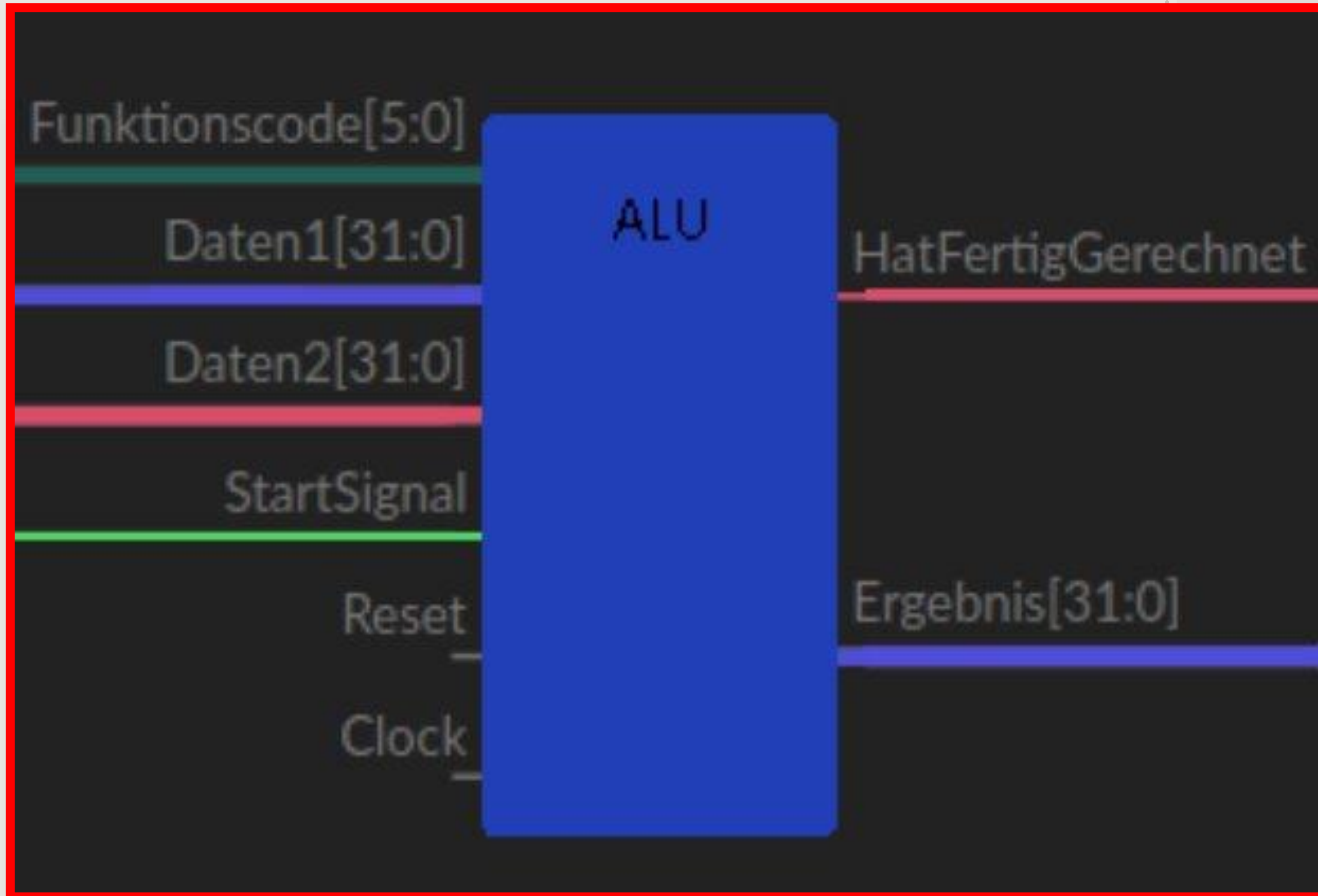
Rest

Kurzname	Typ	Bitcode	Kurzbeschreibung
IToF	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1110	Zf := (float)Q1
UIToF	R	000000 xxxxx xxxxx xxxxx zzzzz 00 1111	Zf := (float)Q1 (Hinweis: Q1 wird als vorzeichenlose Ganzzahl interpretiert)
FToI	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1110	Z := (int)Qf1
FToUI	R	000000 xxxxx xxxxx xxxxx zzzzz 10 1111	Z := (unsigned int)Qf1
Load	I	111000 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := RAM[Q1 + IMO]
Load.s	I	111001 xxxxx xxxxx xxxxxxxxxxxxxxxx	Zf := RAM[Q1 + IMO]
Store	I	111010 xxxxx xxxxx xxxxxxxxxxxxxxxx	RAM[Q1 + IMO] := Z
Store.s	I	111011 xxxxx xxxxx xxxxxxxxxxxxxxxx	RAM[Q1 + IMO] := Zf
Jreg	I	111100 zzzzz xxxxx zzzzzzzzzzzzzzzz	PC := Q1
Bez	I	111101 zzzzz xxxxx xxxxxxxxxxxxxxxx	Falls Q1 == 0, dann PC := PC + IMO
Bnez	I	111110 zzzzz xxxxx xxxxxxxxxxxxxxxx	Falls Q1 != 0, dann PC := PC + IMO
Jal	I	111111 xxxxx zzzzz xxxxxxxxxxxxxxxx	Z := PC, PC := PC + IMO
Jmp	J	010000 xxxxxxxxxxxxxxxxxxxxxxxxxx	PC := PC + IMO

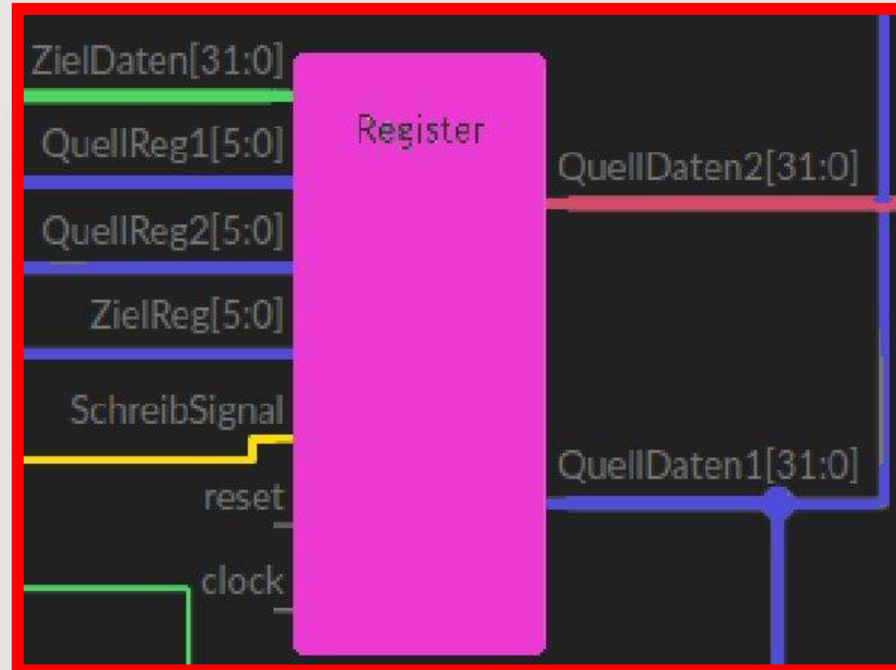
Prozessor

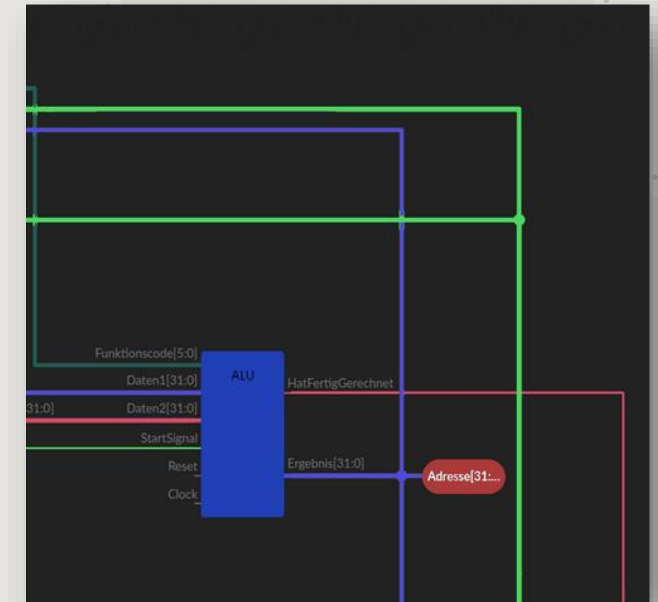


ALU

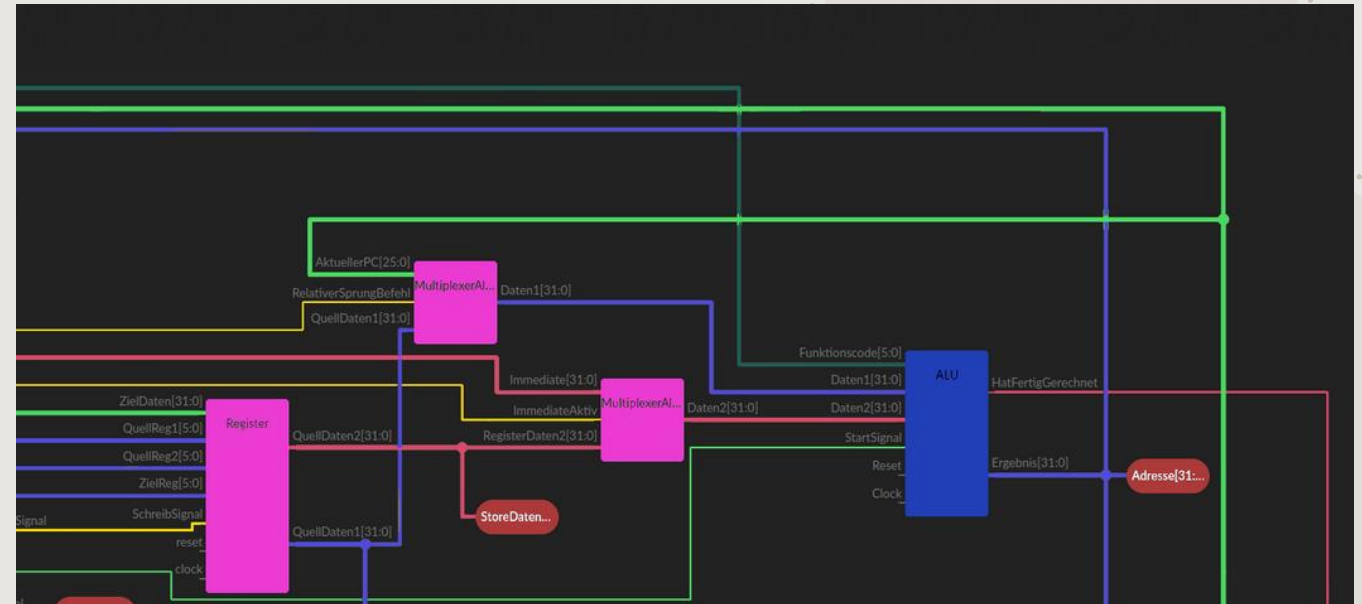


Register



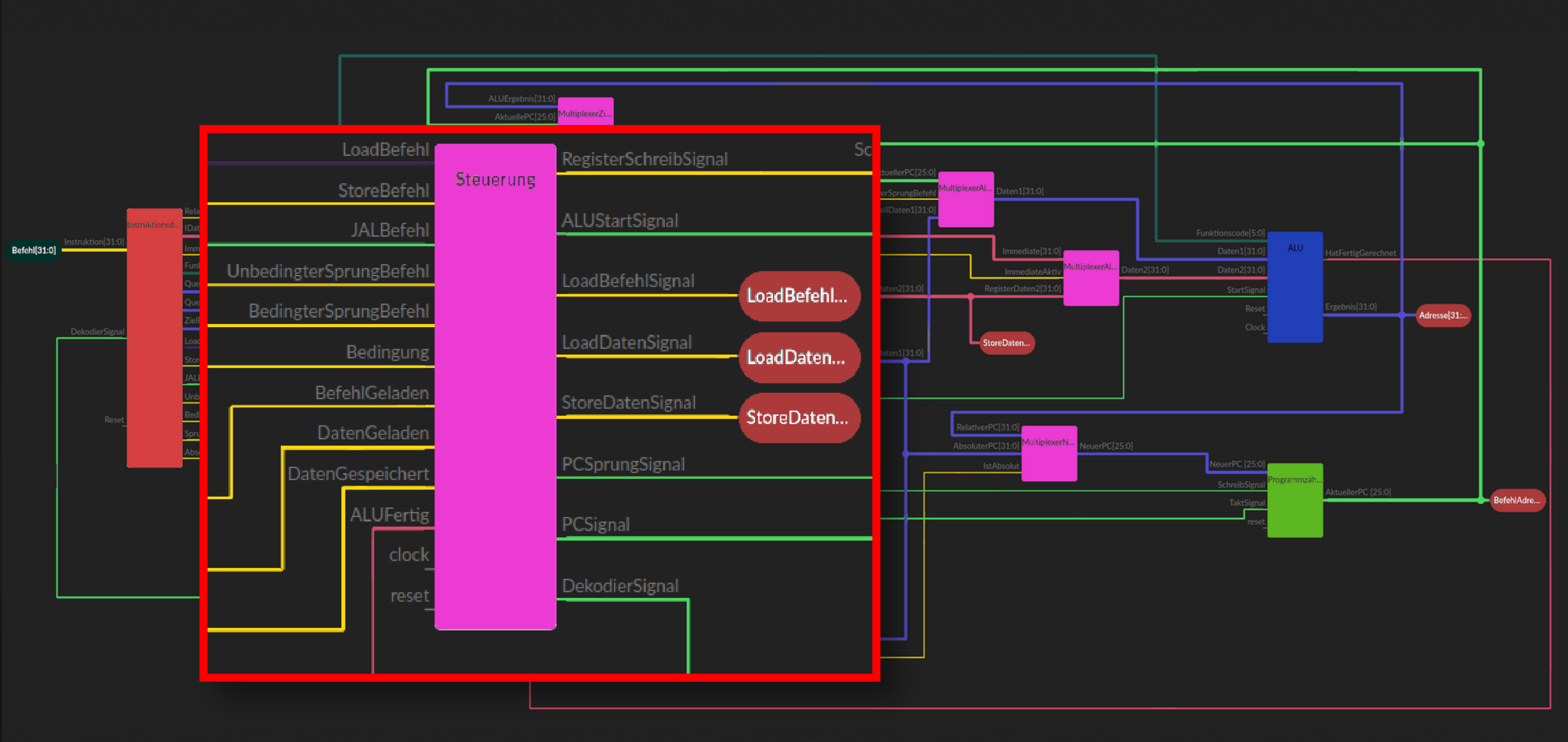


Instruktionsdekodierer

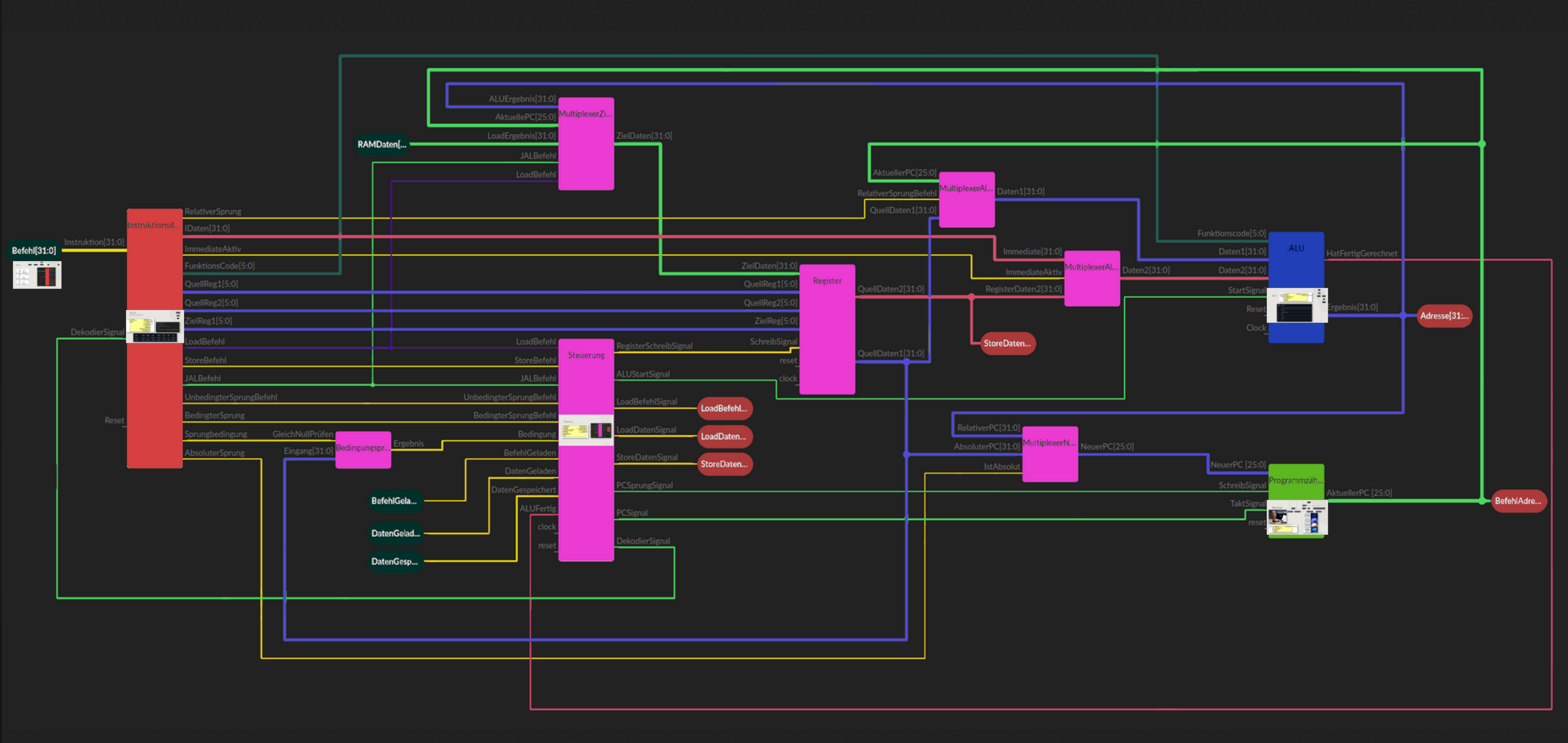


Programmzähler

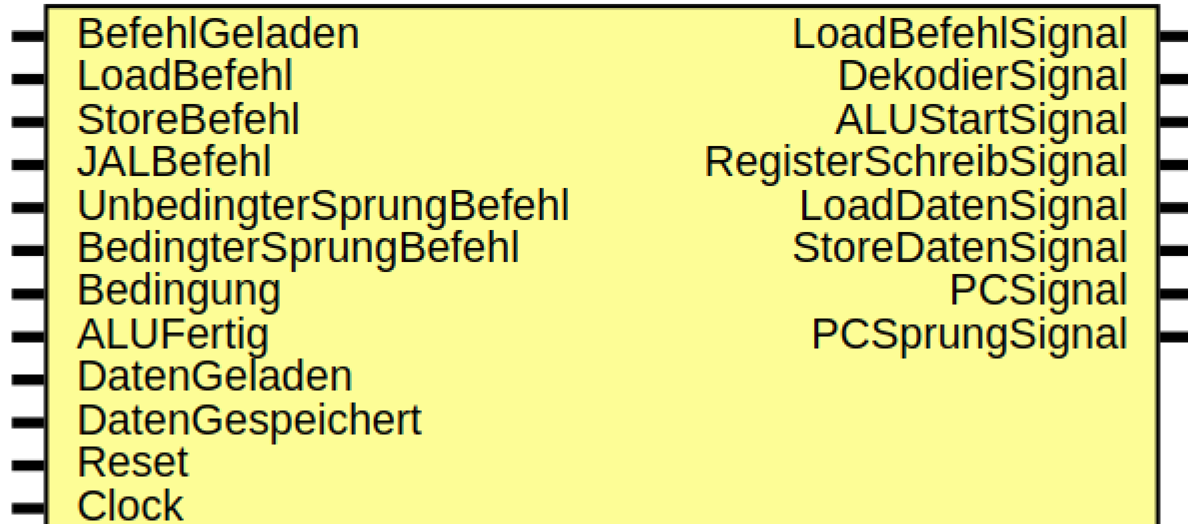
Steuerung

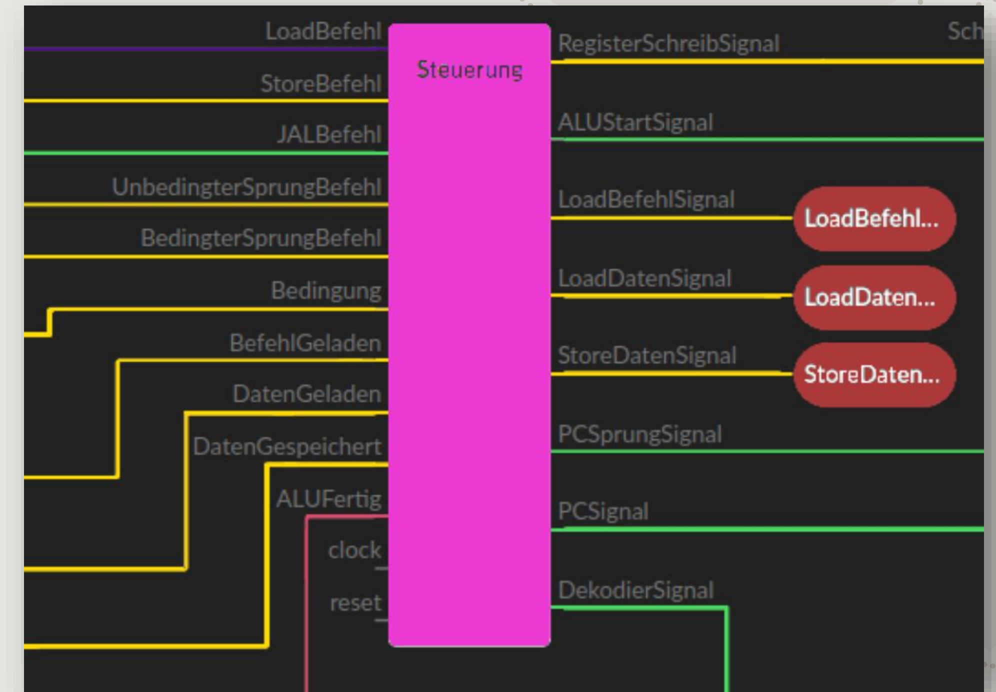


Prozessor

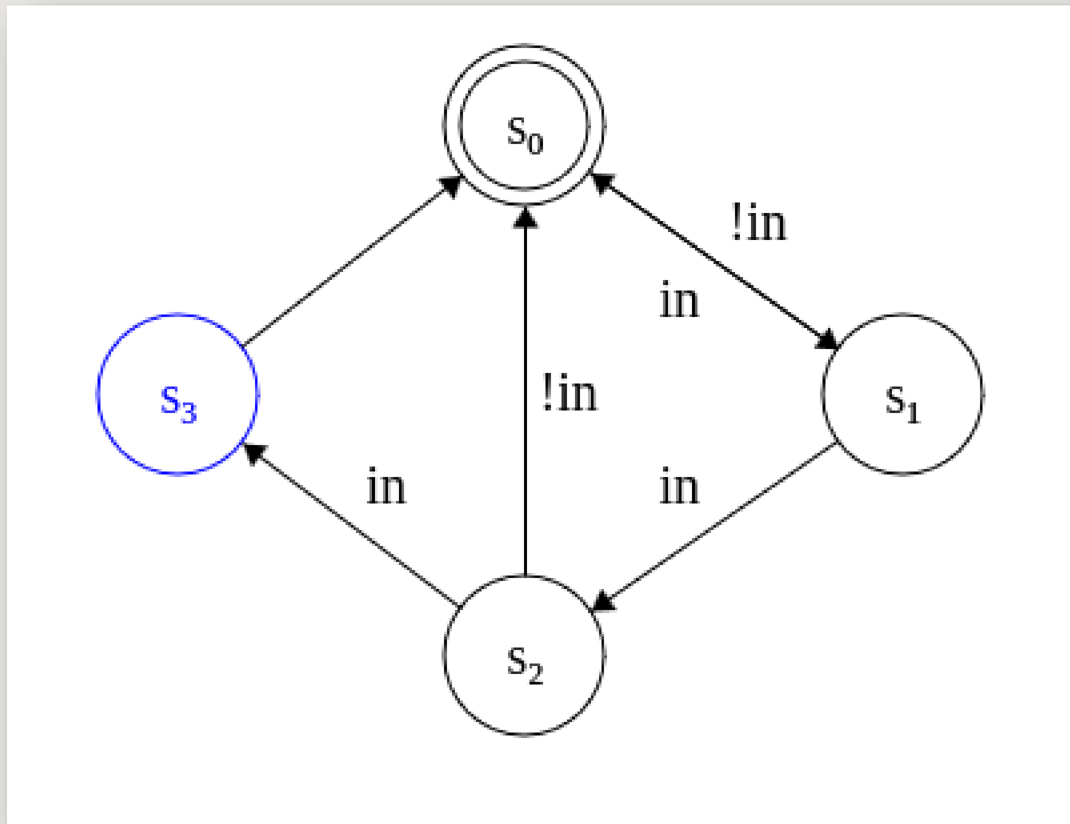


Steuerung





Zustandsmaschinen in Verilog




```
module StateMachine(
    input clk,
    input rst,
    input inp,
    output out
  );
  assign out = (state == State2 | state == State3);

  reg[1:0] state;

  localparam State0 = 2'b00;
  localparam State1 = 2'b10;
  localparam State2 = 2'b01;
  localparam State3 = 2'b11;

  always @(posedge clk)
  begin

    case(state)
      State0:
        state <= inp ? State1 : State0;
      State1:
        state <= inp ? State2 : State0;
      State2:
        state <= inp ? State3 : State0;
      State3:
        state <= State0;
    endcase
    if(rst)
      state <= State0;
  end
endmodule
```

Steuerung

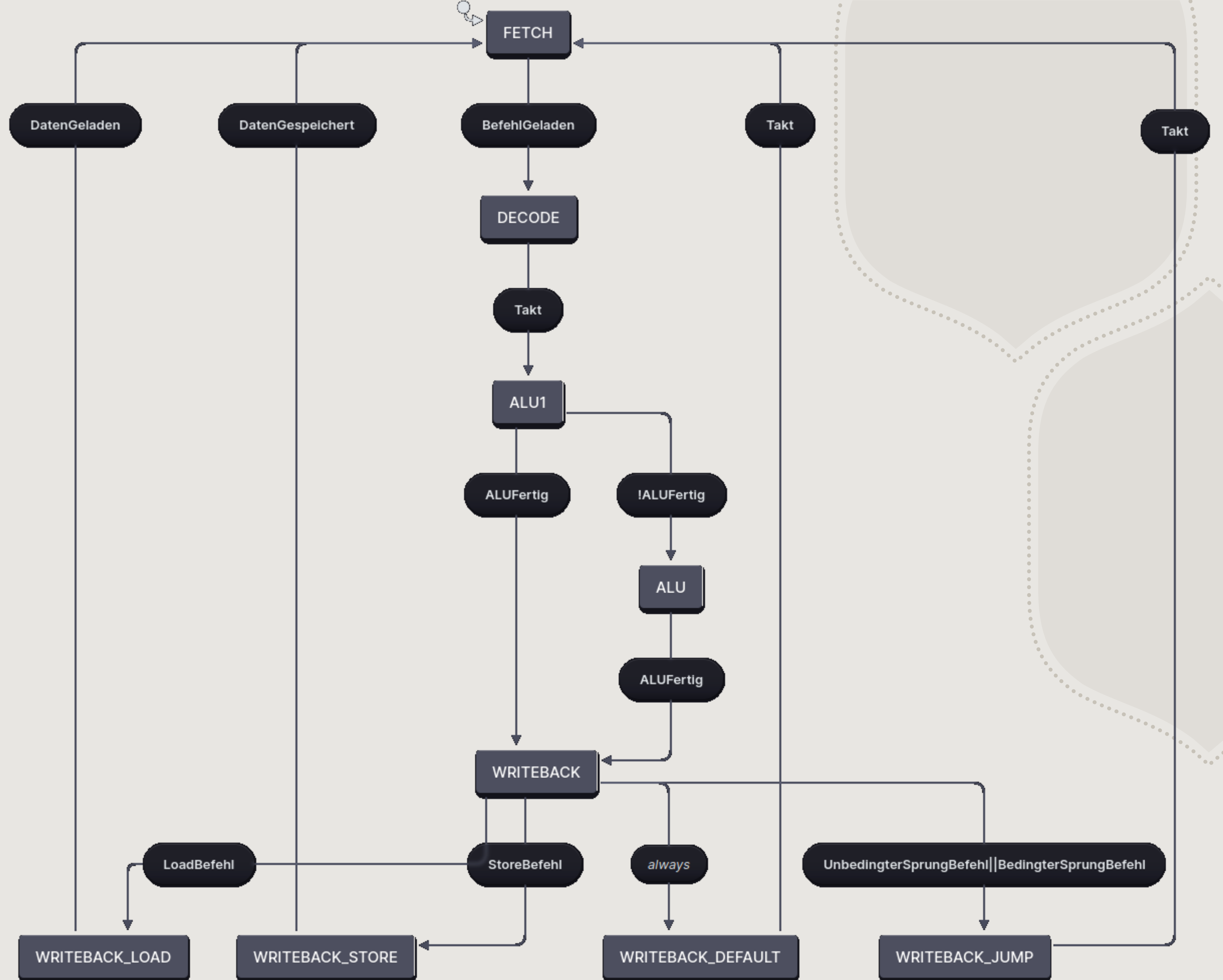
```
localparam FETCH = 4'b000;

localparam DECODE = 4'b001;

localparam ALU1 = 4'b010;
localparam ALU = 4'b011;

localparam WRITEBACK_JUMP = 4'b100;
localparam WRITEBACK_STORE = 4'b101;
localparam WRITEBACK_LOAD = 4'b110;
localparam WRITEBACK_DEFAULT = 4'b111;

localparam WRITEBACK_STORE2 = 4'b1000;
localparam WRITEBACK_LOAD2 = 4'b1001;
localparam WRITEBACK_WRITELOAD = 4'b1011;

reg [3:0] current_state;
reg [3:0] next_state;
```

```
44      //combinational portion
45
46      always @(*) begin
47          case(current_state)
48              FETCH: begin
49                  if (BefehlGeladen)
50                      next_state <= DECODE;
51                  else
52                      next_state <= FETCH;
53              end
54              DECODE:
55                  next_state <= ALU1;
56              ALU1: begin
57                  if (ALUFertig) begin
58                      if (UnbedingterSprungBefehl || BedingterSprungBefehl)
59                          next_state <= WRITEBACK_JUMP;
60                      else if (StoreBefehl)
61                          next_state <= WRITEBACK_STORE;
62                      else if (LoadBefehl)
63                          next_state <= WRITEBACK_LOAD;
64                      else
65                          next_state <= WRITEBACK_DEFAULT;
66                  end
67                  else
68                      next_state <= ALU;
69              end
```

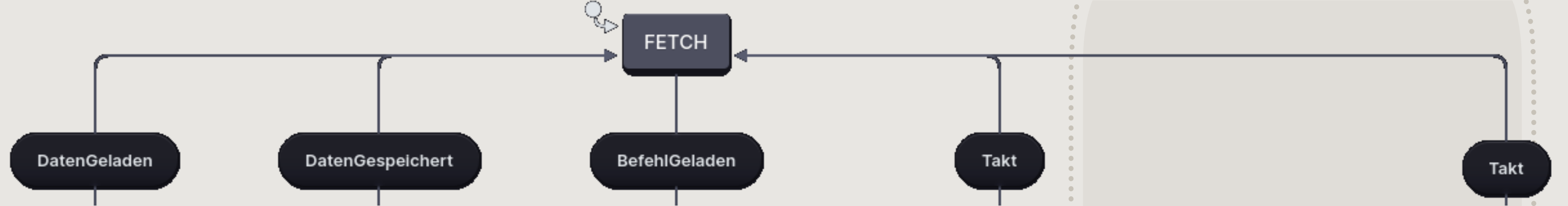
Steuerung

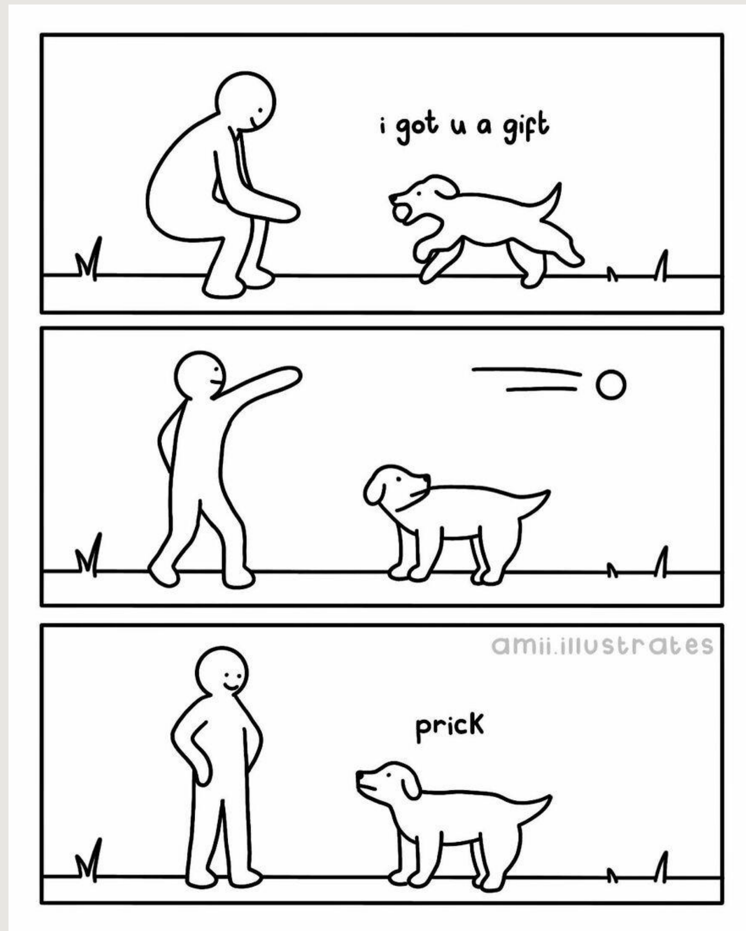
```
120     assign LoadBefehlSignal         = current_state == FETCH;
121     assign DekodierSignal           = current_state == DECODE;
122     assign ALUStartSignal           = current_state == ALU1;
123     assign RegisterSchreibSignal    = (current_state == ALU1 && JALBefehl) || current_state == WRITEBACK_DEFAULT
124                                     || current_state == WRITEBACK_LOAD || current_state == WRITEBACK_WRITELOAD;
125     assign PCSignal                 = current_state > ALU && current_state < WRITEBACK_STORE2; //In einen Writeback Zuständen //Darf nur 1 Takt oben sein
126     assign StoreDatenSignal         = current_state == WRITEBACK_STORE || current_state == WRITEBACK_STORE2;
127     assign LoadDatenSignal          = current_state == WRITEBACK_LOAD  || current_state == WRITEBACK_LOAD2;
128     assign PCSprungSignal           = UnbedingterSprungBefehl || (BedingterSprungBefehl && Bedingung);
129
130     //sequential portion
131
132     always @(posedge Clock) begin
133         current_state <= next_state;
134         if (Reset) begin
135             current_state <= FETCH;
136         end
137     end
```

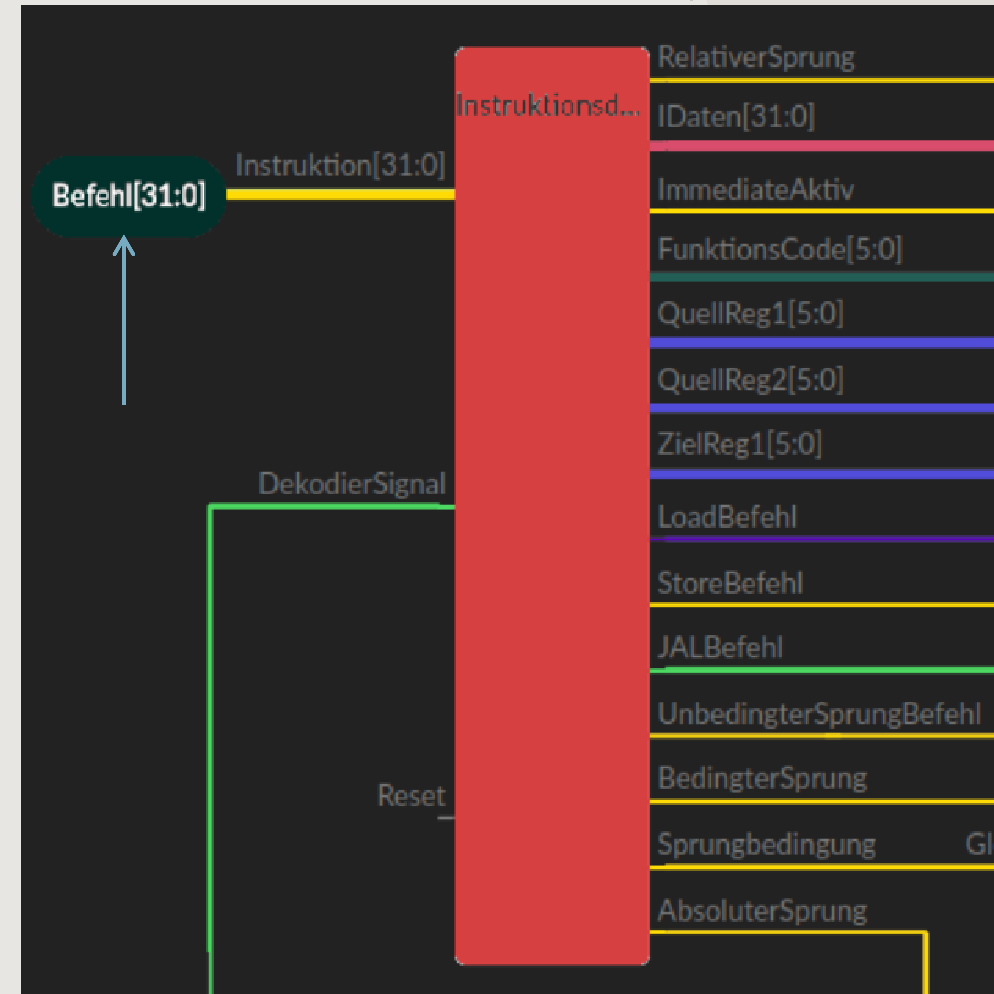
Steuerung



Fetch

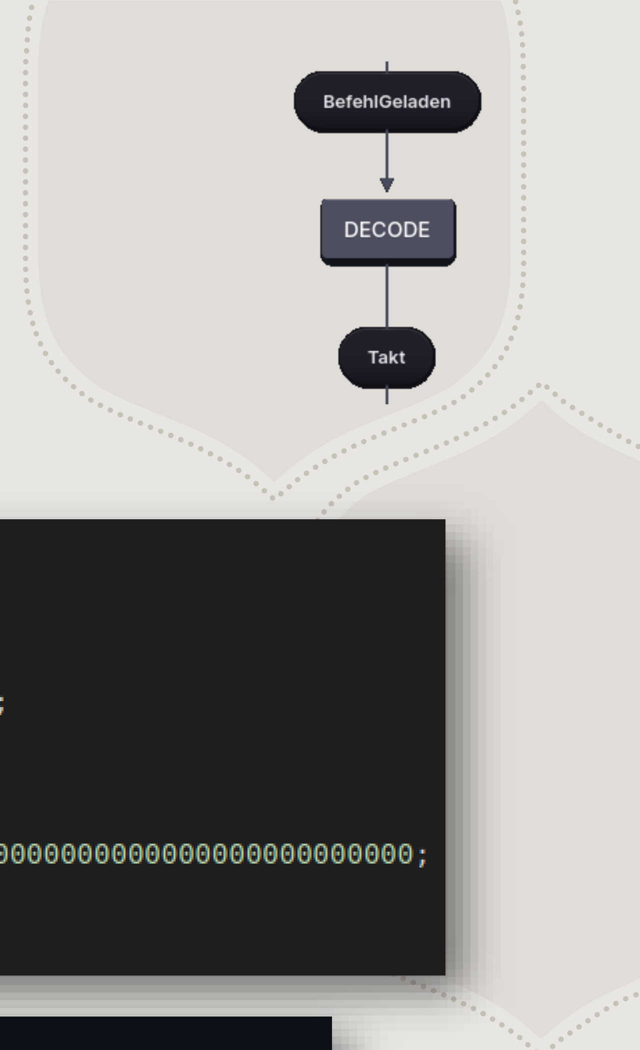


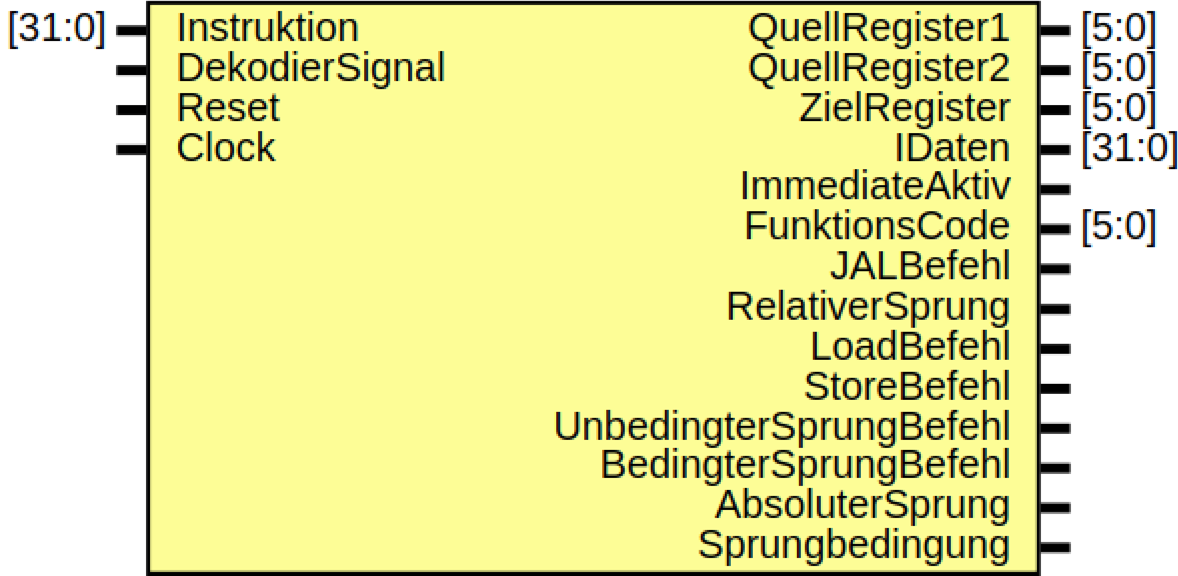




Decode

Instruktionsdekodierer

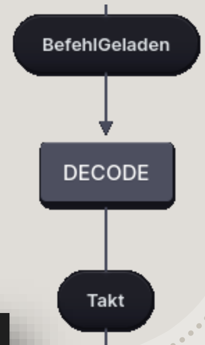





```
always @(posedge Clock)
begin
  if (DekodierSignal)
  begin
    AktuellerBefehl <= Instruktion;
  end
  if(Reset)
  begin
    AktuellerBefehl <= 32'b00000000000000000000000000000000;
  end
end
```

Bits	-	31:30	29:26	25:21	20:16	15:11	10:6	5:0
Zweck	-	Format	frei	Z-Register	Q1-Reg	Q2-Reg	frei	Funktion
Größe	-	2-Bits	4-Bits	5-Bits	5-Bits	5-Bits	5-Bits	6-Bits

Decode
Instruktionsdekodierer




```
assign IDaten                    =   (Format == JumpFormat) ? {{6{GrosserImmediate[25]}}, GrosserImmediate} :
                                      (Opcode == AddisCode) ? {KleinerImmediate, 16'b0} :
                                       (Format[1] == ImmediateFormat) ? {{16{KleinerImmediate[15]}}, KleinerImmediate} :
                                        32'b0;
```

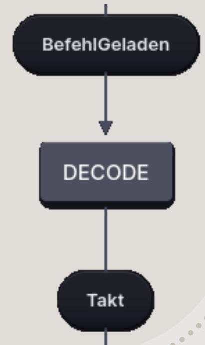
Immediate Format

Bits	-	31:31	30:26	25:21	20:16	15:0
Zweck	-	Format	Funktion	Z-Register	Q-Reg	Immediate
Größe	-	1-Bit	5-Bits	5-Bits	5-Bits	16-Bits

Jump Format

Bits	-	31:30	29:26	25:0
Zweck	-	Format	OP-Code	Immediate
Größe	-	2-Bits	4-Bits	26-Bits

Decode
Instruktionsdekodierer

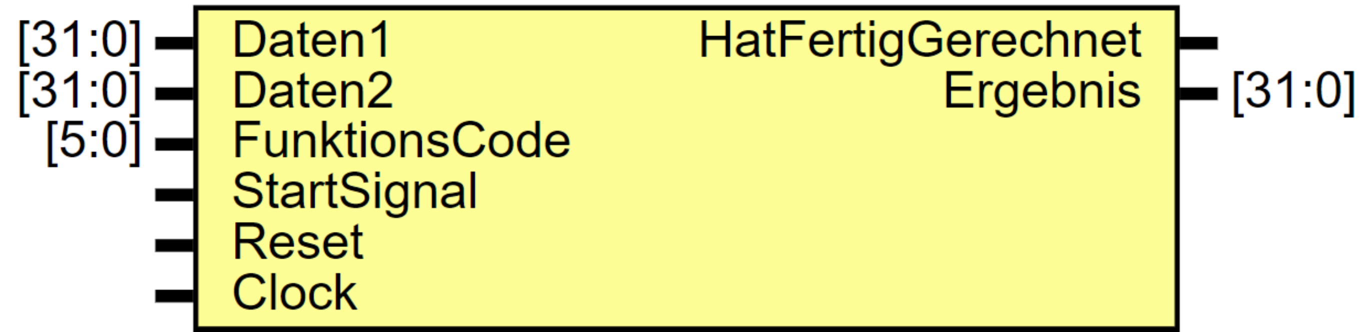


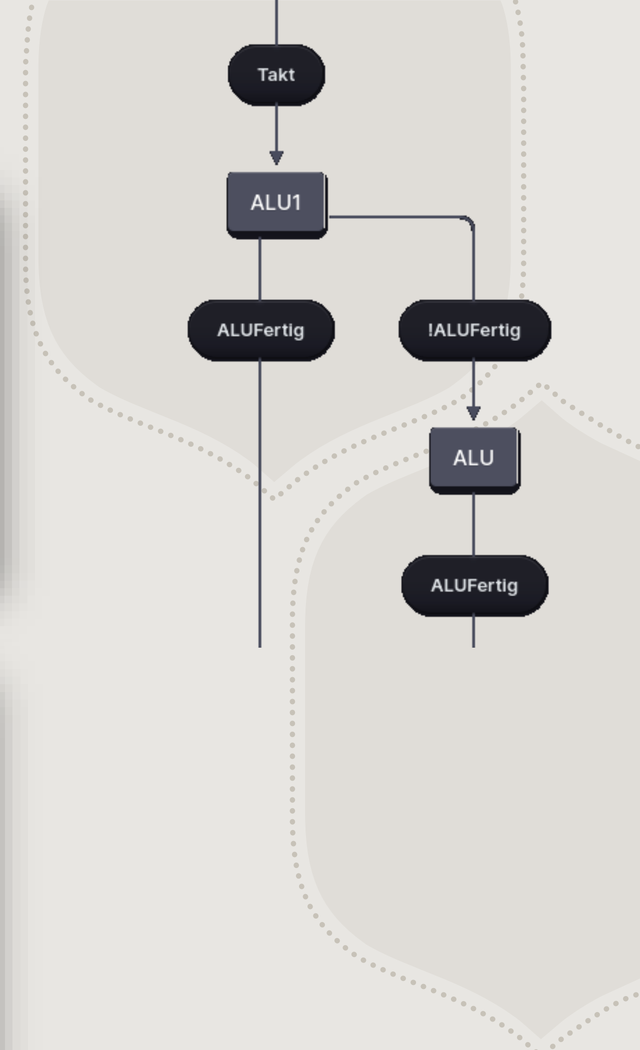
Name	Richtung	Ursprung/Ziel	Beschreibung
FunktionsCode[5:0]	Eingang	Instruktionsdekodierer	Der Funktionscode. Bestimmt die Operation

```
assign FunktionsCode          =   (Format == RegisterFormat) ? Funktion :
                                   ((Opcode == AddisCode || Format == JumpFormat||(Opcode >= LoadCode && Opcode <= JALCode)) ? 6'b0:
                                    {1'b0,FunktionAnfang});
```

Load	I	111000 xxxxx xxxxx xxxxxxxxxxxxxxxx	Z := RAM[Q1 + IMO]
Load.s	I	111001 xxxxx xxxxx xxxxxxxxxxxxxxxx	Zf := RAM[Q1 + IMO]
Store	I	111010 xxxxx xxxxx xxxxxxxxxxxxxxxx	RAM[Q1 + IMO] := Z
Store.s	I	111011 xxxxx xxxxx xxxxxxxxxxxxxxxx	RAM[Q1 + IMO] := Zf
Jreg	I	111100 zzzzz xxxxx zzzzzzzzzzzzzzzz	PC := Q1
Bez	I	111101 zzzzz xxxxx xxxxxxxxxxxxxxxx	Falls Q1 == 0, dann PC := PC + IMO
Bnez	I	111110 zzzzz xxxxx xxxxxxxxxxxxxxxx	Falls Q1 != 0, dann PC := PC + IMO
Jal	I	111111 xxxxx zzzzz xxxxxxxxxxxxxxxx	Z := PC, PC := PC + IMO

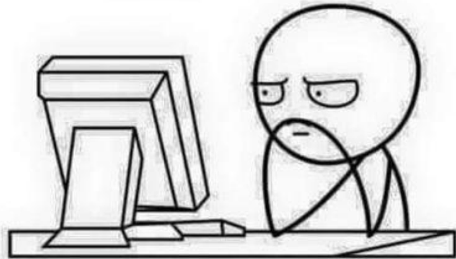
ALU





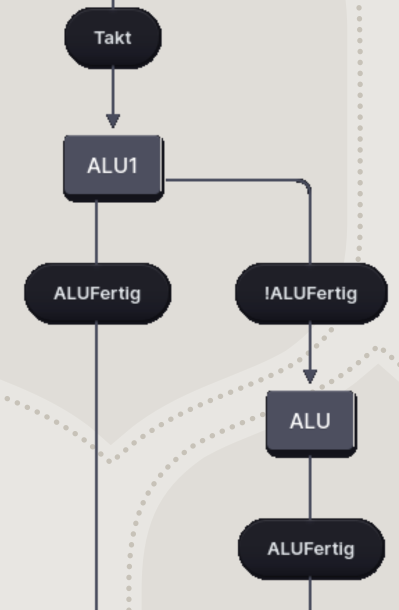
Dauer in Takten	Befehle
1	Add, Sub, Mul, Sla, Sra, Ce, Cne, Cg, Cl, Cgu, Clu, Not, And, Or, Xor, Xnor, Sll, Srl, Ce.s, Cne.s, FToUI
2	FToI
5	IToF, UIToF
7	Add.s, Sub.s, Cg.s, Cl.s
9	Mul.s
16	Sqrt*
32	Mod*, Div*
36	Div.s
52	Sqrt.s

ALU



https://de.pinterest.com/pamelajsmith/computer-memes/

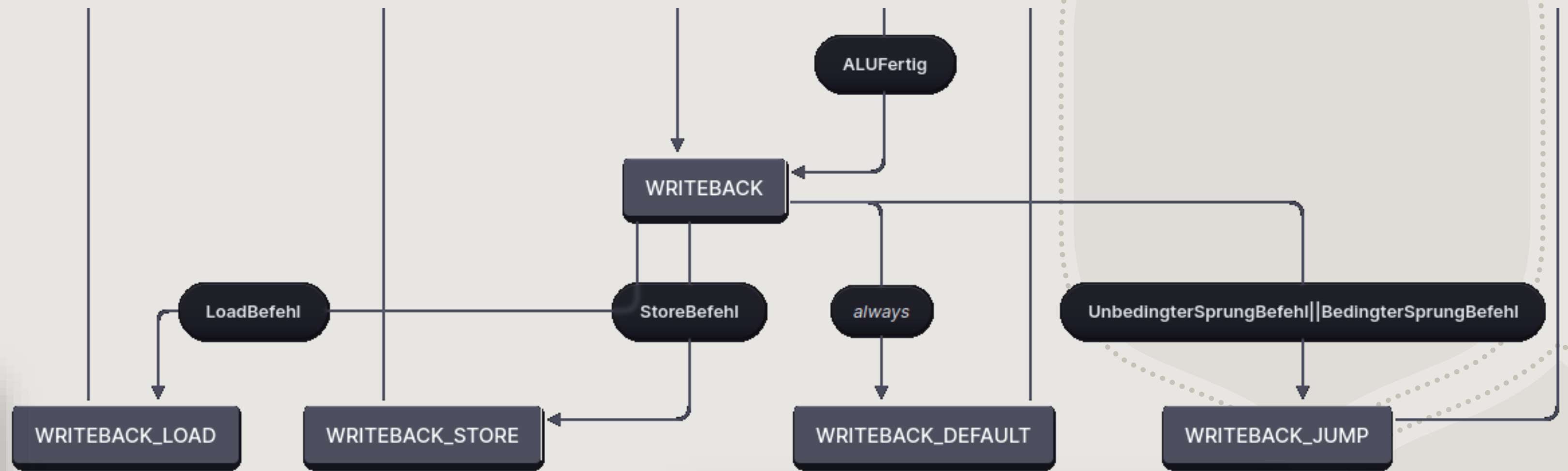
```
always @(posedge Clock) begin
    if(TakteBisFertig != 0) begin
        TakteBisFertig <= TakteBisFertig - 1;
    end
    else if (StartSignal) begin
        Radikand <= Daten1;
        case (FunktionsCode[5:0])
            IntZuFloat : begin
                TakteBisFertig <= 5;
            end
            UnsignedIntZuFloat : begin
                TakteBisFertig <= 5;
            end
            //Float Arithmetik
            //Add.s
            FloatAddition: begin
                TakteBisFertig <= 7;
            end
            //Sub.s
            FloatSubtraktion: begin
                TakteBisFertig <= 7;
            end
            //Mul.s
            FloatMultiplikation    : begin
                TakteBisFertig <= 9;
            end
            //Div.s
            FloatDivision    : begin
                TakteBisFertig <= 36;
            end
            //Cg.s
            FloatGroesser   : begin
                TakteBisFertig <= 7;
            end
            //Cl.s
            FloatKleiner    : begin
                TakteBisFertig <= 7;
            end
            FloatZuInt : begin
                TakteBisFertig <= 2;
            end
            FloatZuUnsignedInt : begin
                TakteBisFertig <= 1;
            end
            default : begin
                TakteBisFertig <= 0;
            end
        endcase
    end
    if(Reset) begin
        TakteBisFertig <= 0;
    end
```

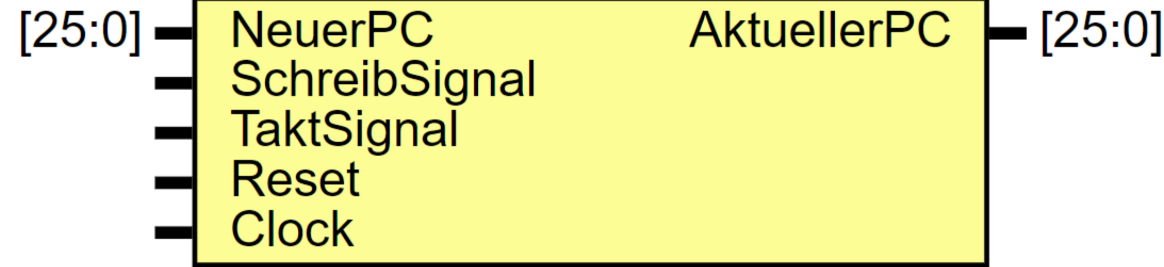



```
assign HatFertigGerechnet = (FunktionsCode == IntDivision || FunktionsCode == IntModulo)?(DivModFertig):
                            (FunktionsCode == IntQuadratwurzel)?(WurzelFertig):
                            (FunktionsCode == FloatQuadratwurzel)?FloatWurzelerFertig:
                            (TakteBisFertig == 0);
```

Writeback

https://www.reddit.com/r/okbuddyphd/comments/z7t5cp/okbuddymipszy/





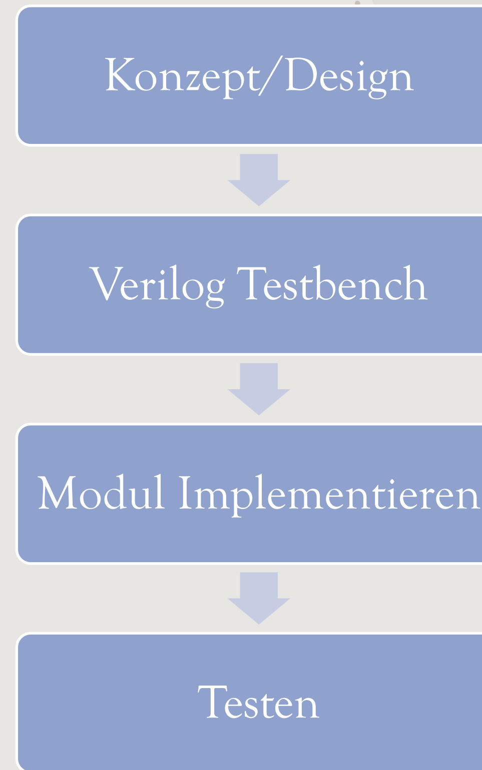
https://benswift.me/blog/2021/02/19/respectful-learning-memes/

Resourcen Ver*sch*wendung auf dem ECP5-85F

```
Info: Device utilisation:
Info:            TRELLIS_IO:      26/    365     7%
Info:                  DCCA:       4/     56     7%
Info:                DP16KD:      64/    208    30%
Info:            MULT18X18D:       9/    156     5%
Info:                ALU54B:       0/     78     0%
Info:               EHXPLLL:       2/      4    50%
Info:               EXTREFB:       0/      2     0%
Info:                  DCUA:       0/      2     0%
Info:             PCSCLKDIV:       0/      2     0%
Info:               IOLOGIC:       0/    224     0%
Info:              SIOLOGIC:       3/    141     2%
Info:                   GSR:       0/      1     0%
Info:                 JTAGG:       0/      1     0%
Info:                  OSCG:       0/      1     0%
Info:                 SEDGA:       0/      1     0%
Info:                   DTR:       0/      1     0%
Info:               USRMCLK:       0/      1     0%
Info:               CLKDIVF:       0/      4     0%
Info:             ECLKSYNCB:       0/     10     0%
Info:               DLLDELD:       0/      8     0%
Info:                DDRDLL:       0/      4     0%
Info:               DQSBUFM:       0/     14     0%
Info:        TRELLIS_ECLKBUF:      0/      8     0%
Info:          ECLKBRIDGECS:       0/      2     0%
Info:                  DCSC:       0/      2     0%
Info:            TRELLIS_FF:    9938/  83640    11%
Info:          TRELLIS_COMB:   64495/  83640    77%
Info:          TRELLIS_RAMW:    4800/  10455    45%
```

```
=== Top ===

   Number of wires:                  38662
   Number of wire bits:             165669
   Number of public wires:           38662
   Number of public wire bits:      165669
   Number of ports:                      7
   Number of port bits:                 26
   Number of memories:                   0
   Number of memory bits:                0
   Number of processes:                  0
   Number of cells:                  53881
     $scopeinfo                        344
     CCU2C                            1566
     DP16KD                             64
     EHXPLLL                             2
     L6MUX21                           443
     LUT4                            31809
     MULT18X18D                          9
     ODDRX1F                             3
     PFUMX                            4903
     TRELLIS_DPR16X4                  4800
     TRELLIS_FF                       9938
```

Programmstrukturierung



Programmstrukturierung

Unser alter Instruktionsdekodierer

```
assign QuellRegister1 =         (AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]!=2'b10) ? {1'b0, AktuellerBefehl[20:16]}:
       ((AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]==2'b10) ? {1'b1, AktuellerBefehl[20:16]}:
        ((AktuellerBefehl[31:31]==1'b1) ? {1'b0, AktuellerBefehl[20:16]}:
         6'b0));

assign QuellRegister2 =         (AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]!=2'b10) ? {1'b0, AktuellerBefehl[15:11]}:
       ((AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]==2'b10) ? {1'b1, AktuellerBefehl[15:11]}:
        ((AktuellerBefehl[31:26]==6'b101100) ? {1'b0, AktuellerBefehl[25:21]}:
         6'b0));

assign ZielRegister =           (AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]!=2'b10) ? {1'b0, AktuellerBefehl[25:21]}:
       (((AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]==2'b10)||AktuellerBefehl[31:26]==6'b101011) ? {1'b1, AktuellerBefehl[25:21]}:
        ((AktuellerBefehl[31:31]==1'b1) ? {1'b0, AktuellerBefehl[25:21]}:
         6'b0));

assign IDaten =                 (AktuellerBefehl[31:30]==2'b01) ? AktuellerBefehl[25:0]:
       ((AktuellerBefehl[31:31]==1'b1) ? {10'b0, AktuellerBefehl[15:0]}:
        26'b0);

assign KleinerImmediateAktiv =  (AktuellerBefehl[31:31]==1'b1);

assign GrosserImmediateAktiv =  (AktuellerBefehl[31:30]==2'b01);

assign FunktionsCode =          (AktuellerBefehl[31:30]==2'b00) ? AktuellerBefehl[5:0]:
       ((AktuellerBefehl[31:30]==2'b01||(AktuellerBefehl[31:26]>=6'b101010&&AktuellerBefehl[31:26]<=6'b101111)) ? 6'b0:
        {1'b0,AktuellerBefehl[30:26]});

assign JALBefehl =              (AktuellerBefehl[31:26]==6'b101111);

assign RelativerSprung =        (AktuellerBefehl[31:26]==6'b101111 || AktuellerBefehl[31:26]==6'b010000 || AktuellerBefehl[31:26]==6'b101110);

assign AbsoluterSprung =        (AktuellerBefehl[31:26]==6'b101101);

assign FloatBefehl =            ((AktuellerBefehl[31:30]==2'b00 && AktuellerBefehl[5:4]==2'b10)||AktuellerBefehl[31:26]==6'b101011);

assign LoadBefehl =             (AktuellerBefehl[31:27]==5'b10101);

assign StoreBefehl =            (AktuellerBefehl[31:26]==6'b101100);

assign UnbedingterSprungBefehl =(AktuellerBefehl[31:26]==6'b101101 || AktuellerBefehl[31:26]==6'b101111 || AktuellerBefehl[31:26]==6'b010000);

assign BedingterSprungBefehl =  (AktuellerBefehl[31:26]==6'b101110);
```

Programmstrukturierungen

25:21	20:16	15:11	10:6	5:0
Z-Register	Q1-Reg	Q2-Reg	frei	Funktion
5-Bits	5-Bits	5-Bits	5-Bits	6-Bits

```
//Der aktuell gespeicherte Befehl
reg [31:0] AktuellerBefehl;

//Variablen
wire[5:0] Opcode;
// FORMATE
wire[1:0] Format;
wire[1:0] Kategorie;
//Register
wire[4:0] ZRegister;
wire[4:0] Q1Register;
wire[4:0] Q2Register;
wire[5:0] Funktion;
//Funktion
wire[5:0] FunktionAnfang;
//Immediate
wire[15:0] KleinerImmediate;
wire[25:0] GrosserImmediate;
//GleitkommaBefehl
wire[3:0] GleitkommaBefehl;

//Zuweisungen
assign Opcode = AktuellerBefehl[31:26];
assign Format = AktuellerBefehl[31:30];
assign Kategorie = AktuellerBefehl[5:4];
assign ZRegister = AktuellerBefehl[25:21];
assign Q1Register = AktuellerBefehl[20:16];
assign Q2Register = AktuellerBefehl[15:11];
assign Funktion = AktuellerBefehl[5:0];
assign FunktionAnfang = AktuellerBefehl[30:26];
assign KleinerImmediate = AktuellerBefehl[15:0];
assign GrosserImmediate = AktuellerBefehl[25:0];
assign GleitkommaBefehl = AktuellerBefehl[3:0];
```

Programmstrukturierungen

ELEKTROTECHNIK
IST EINE LÜGE

- Niemand hat je Strom gesehen!
- Elektrizität haben die DIE DA OBEN erfunden, um ZAUBEREI zu verbergen
- ""Strom"" ist nutzlos. Du willst Licht? Mach ein Feuer!!!

"Islolation"? SIE VESTECKEN ETWAS VOR DIR!

???

DAS IST VIEL ZU KLEIN

Speichert angeblich 'Elektronen', hat aber gar keine Öffnung zum be- und entladen

Das hier ist, was sich wirklich hinter den LÜGEN der Elektotechnik verbirgt:

Buntes Gekritzel

???

Das mit den Kreisen

ECHTE MENSCHEN glauben ernsthaft, dass man mit BUCHSTABEN RECHNEN KANN

ALLE Bilder von Strom sind ZEICHNUNGEN

SIE HALTEN UNS FÜR DUMM!!!

```
//Formate
localparam[1:0] RegisterFormat = 2'b00;
localparam[1:0] JumpFormat = 2'b01;
localparam ImmediateFormat = 1'b1;
//Befehlstypen
localparam FloatVergleich = 1'b1;

localparam[1:0] Arithmetik = 2'b00;
localparam[1:0] Vergleich = 2'b01;
localparam[1:0] Gleitkomma = 2'b10;
localparam[1:0] Vektor = 2'b11;

//Wichtige Befehle die einzeln behandelt werden müssen
localparam[5:0] LoadCode    = 6'b111000;
localparam[5:0] LoadSCode   = 6'b111001;
localparam[5:0] StoreCode   = 6'b111010;
localparam[5:0] StoreSCode  = 6'b111011;
localparam[5:0] JregCode    = 6'b111100;
localparam[5:0] BezCode     = 6'b111101;
localparam[5:0] BNezCode    = 6'b111110;
localparam[5:0] JALCode     = 6'b111111;
localparam[5:0] JmpCode     = 6'b010000;
localparam[5:0] AddisCode   = 6'b110000;
localparam[5:0] IToF        = 6'b001110;
localparam[5:0] UIToF       = 6'b001111;
```

Programmstrukturierungen

```
assign QuellRegister1          =   {((Format == RegisterFormat && Kategorie == Gleitkomma) ?   1'b1    :   1'b0),Q1Register};

assign QuellRegister2          =   (Opcode == StoreCode) ? {1'b0, ZRegister}:
                                    (Opcode == StoreSCode) ? {1'b1, ZRegister}:
                                     {((Format == RegisterFormat && Kategorie == Gleitkomma) ? 1'b1 : 1'b0), Q2Register};

assign ZielRegister            =   (Opcode==LoadSCode || Opcode == StoreSCode || (Format == RegisterFormat && ((Kategorie == Gleitkomma
                                    (((Format == RegisterFormat) || (Format[1] == ImmediateFormat)) ? {1'b0, ZRegister}:
                                     6'b0);

assign IDaten                  =   (Format == JumpFormat) ? {{6{GrosserImmediate[25]}}, GrosserImmediate} :
                                    (Opcode == AddisCode) ? {KleinerImmediate, 16'b0} :
                                     (Format[1] == ImmediateFormat) ? {{16{KleinerImmediate[15]}}, KleinerImmediate} :
                                      32'b0;

assign ImmediateAktiv          =   (Format == JumpFormat || Format[1] == ImmediateFormat);

assign FunktionsCode           =   (Format == RegisterFormat) ? Funktion :
                                    ((Opcode == AddisCode || Format == JumpFormat||(Opcode >= LoadCode && Opcode <= JALCode)) ? 6'b0:
                                     {1'b0,FunktionAnfang});

assign JALBefehl               =   (Opcode == JALCode);

assign RelativerSprung         =   (Opcode == JALCode || Opcode == JmpCode || Opcode == BezCode || Opcode == BNezCode);

assign AbsoluterSprung         =   (Opcode == JregCode);

assign LoadBefehl              =   (Opcode == LoadCode || Opcode == LoadSCode);

assign StoreBefehl             =   (Opcode == StoreCode || Opcode == StoreSCode);

assign UnbedingterSprungBefehl =   (Opcode == JregCode || Opcode == JALCode || Opcode == JmpCode);

assign BedingterSprungBefehl   =   (Opcode == BezCode || Opcode == BNezCode);

assign Sprungbedingung         =   (Opcode == BezCode);
```

Danke an JRP der unseren Code reviewt hat und uns Verbesserungsvorschläge gegeben hat :)

Programmstrukturierungen

```
//Gausssumme für 15 (= 120). Hier sollten LEDs auf DatenRaus 7:0 sein
Daten[0] = 32'b10000000010000000000000000001111; //Addi R2, R0, #15
Daten[1] = 32'b00000000001000000000000000000000; //Add R1, R0, R0
Daten[2] = 32'b10000011111000000000000000000001; //Addi R31, R0, #1
Daten[3] = 32'b10011011111111110000000000011111; //Sli R31, R31, #31
Daten[4] = 32'b00000000001000010001000000000000; //Add R1, R1, R2
Daten[5] = 32'b10000100010000100000000000000001; //Subi R2, R2, 1
Daten[6] = 32'b00000000011000100000000000001000; //Ce R3, R2, R0
Daten[7] = 32'b11101000001111110000000000000000; //Store R1, R31, #0
Daten[8] = 32'b11110100000000111111111111111011; //Bez R3, -4
Daten[9] = 32'b01000011111111111111111111111111; //jmp -1
for (idx = 10; idx < WORDS; idx = idx + 1)
    Daten[idx] = 0;
end
```

Programmstrukturierungen

```verilog
module queue #(
    parameter DATA_WIDTH = 32,
    parameter MAX_QUEUE_SIZE = 16
)(
    input clk,
    input rst,
    input enqueue,
    input dequeue,
    input [DATA_WIDTH-1:0] data_in,
    output reg [DATA_WIDTH-1:0] data_out,
    output empty,
    output full,
    output[$clog2(MAX_QUEUE_SIZE)-1:0] size
);
```

```verilog
module RAM#(
    parameter WORDSIZE = 32,
    parameter WORDS = 32
)
(
    input LesenAn,
    input SchreibenAn,
    input[WORDSIZE - 1:0] DatenRein,
    input[$clog2(WORDS) - 1:0] Adresse,
    input Clock,

    output reg[WORDSIZE - 1:0] DatenRaus,
    output reg DatenBereit = 0,
    output reg DatenGeschrieben = 0
);

reg[WORDSIZE - 1:0] Daten[WORDS - 1:0];

always @(posedge Clock) begin

    if (LesenAn) begin
        DatenRaus <= Daten[Adresse];
        DatenBereit <= 1;
    end
    else if(DatenBereit) begin
        DatenBereit <= 0;
    end
end

always @(posedge Clock) begin
    if(SchreibenAn) begin
        Daten[Adresse] <= DatenRein;
        DatenGeschrieben <= 1;
    end
    else if(DatenGeschrieben) begin
        DatenGeschrieben <= 0;
    end
end
endmodule
```

Programmstrukturierungen

```
 1 module pipeline1 #(
 2     parameter int DATA_WIDTH = 16
 3 ) (
 4     input  logic                  clk,
 5     input  logic                  rst,
 6     input  logic                  valid_in,
 7     input  logic [DATA_WIDTH-1:0] inputs   [8],
 8     output logic                  valid_out,
 9     output logic [DATA_WIDTH-1:0] sum
10 );
11
12     logic [DATA_WIDTH-1:0] add0_r, add1_r, add2_r;
13     logic [1:0] valid_r;
14
15     always_ff @(posedge clk) begin
16         if (rst) begin
17             valid_r <= '0;
18             add0_r  <= '0;  // Doesn't need to be reset
19             add1_r  <= '0;  // Doesn't need to be reset
20             add2_r  <= '0;  // Doesn't need to be reset
21         end else begin
22             valid_r[0] <= valid_in;
23             valid_r[1] <= valid_r[0];
24
25             add0_r <= inputs[0] + inputs[1];
26             add1_r <= inputs[2] + inputs[3];
27             add2_r <= add0_r + add1_r;
28         end
29     end
30
31     assign valid_out = valid_r[1];
32     assign sum = add2_r;
33
34 endmodule
```

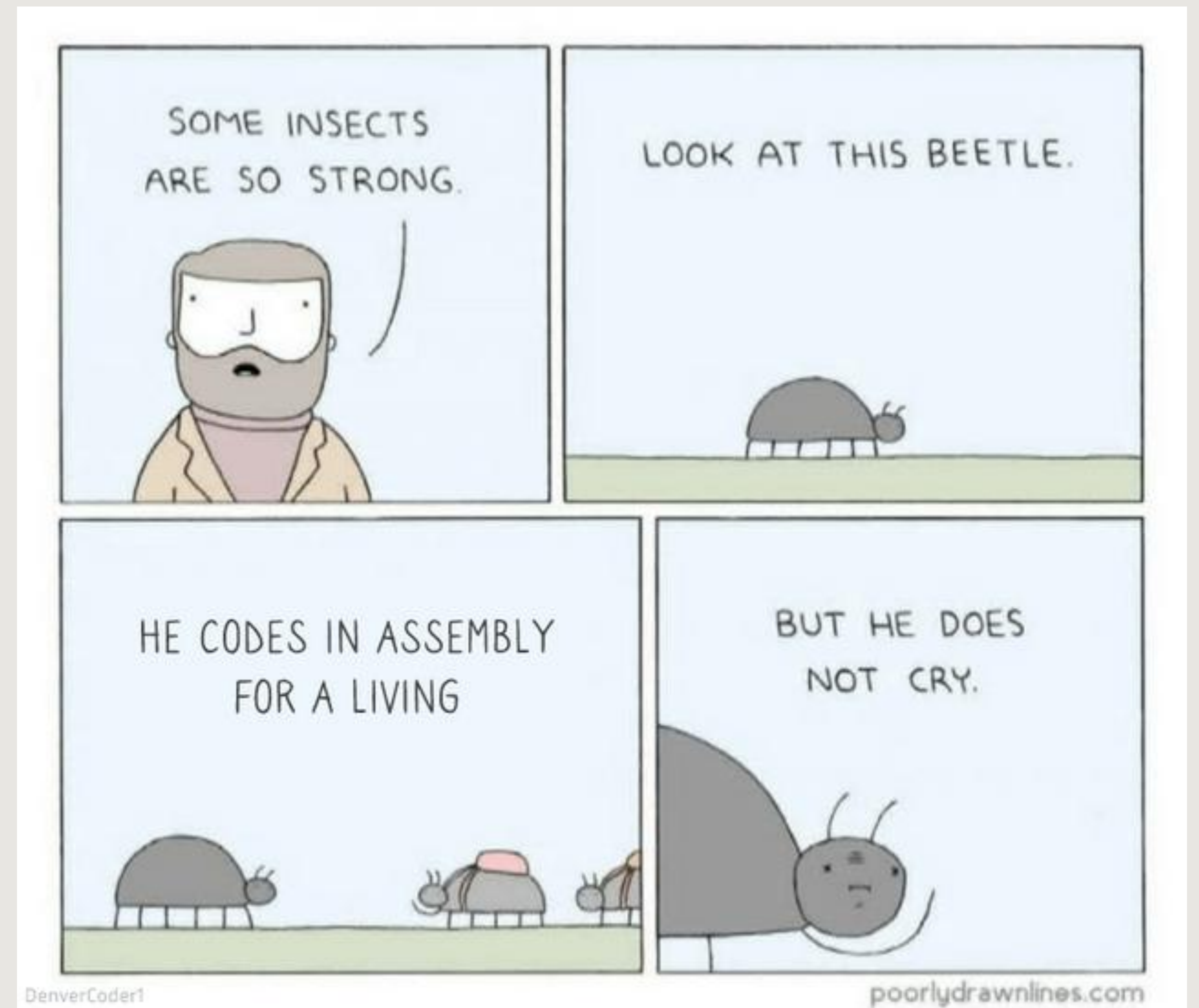
```
15     always_ff @(posedge clk) begin
16         if (rst) begin
17             valid_r <= '0; // ONLY RESET VALID LOGIC
18         end else begin
19             valid_r[0] <= valid_in;
20             valid_r[1] <= valid_r[0];
21
22             add0_r <= inputs[0] + inputs[1];
23             add1_r <= inputs[2] + inputs[3];
24             add2_r <= add0_r + add1_r;
25         end
26     end
```

```
15     always_ff @(posedge clk) begin
16         valid_r[0] <= valid_in;
17         valid_r[1] <= valid_r[0];
18
19         add0_r <= inputs[0] + inputs[1];
20         add1_r <= inputs[2] + inputs[3];
21         add2_r <= add0_r + add1_r;
22
23         if (rst) begin
24             valid_r <= '0;
25         end
26     end
```

https://stitt-hub.com/crafting-clean-reset-logic/

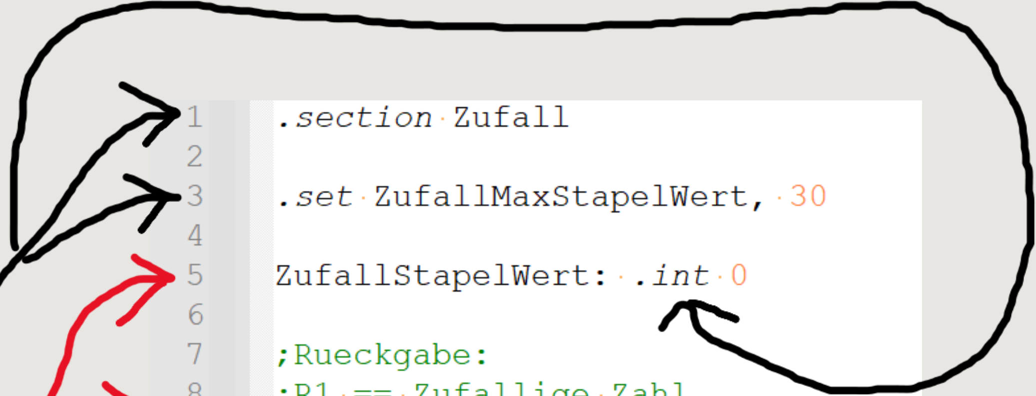
https://xkcd.com/927/

Der Assembler



Beispiel

- Befehle
- Kommentare
- Anweisungen
- Symbole



Funktionsweise

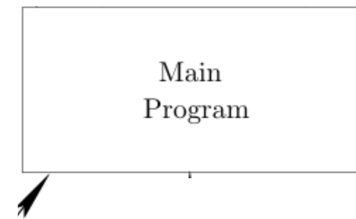
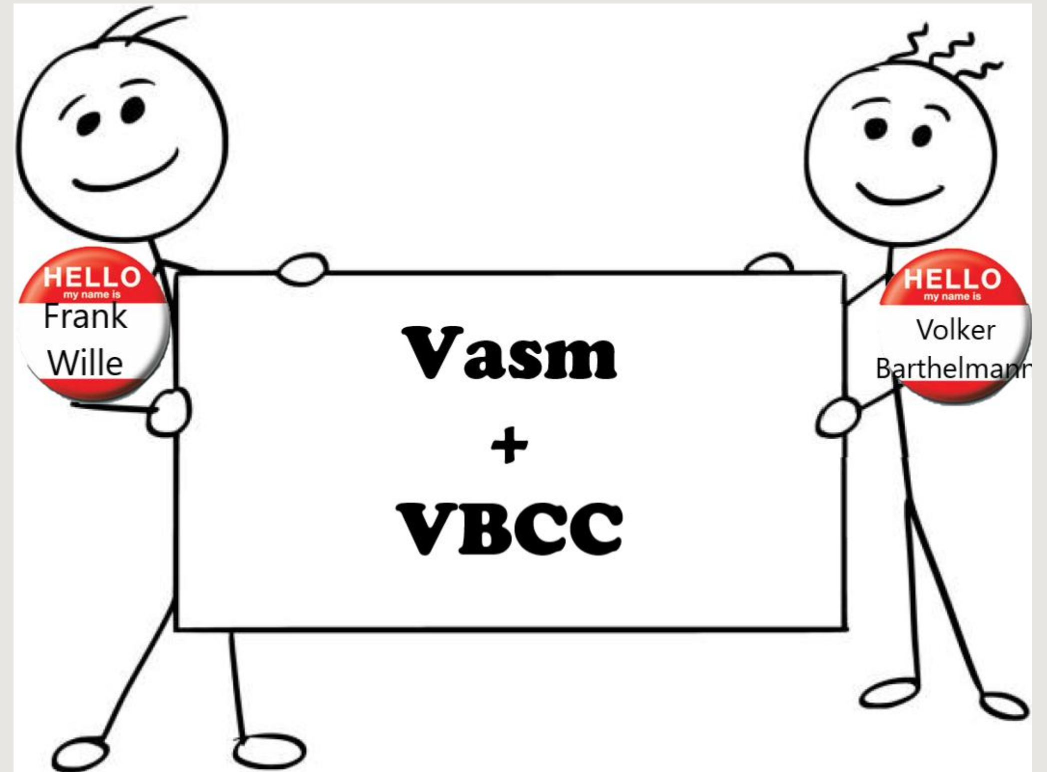
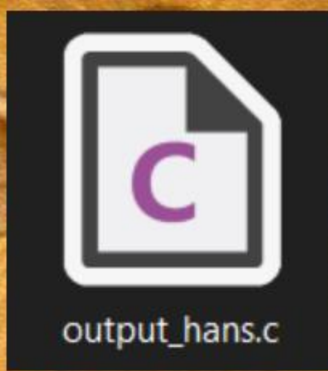


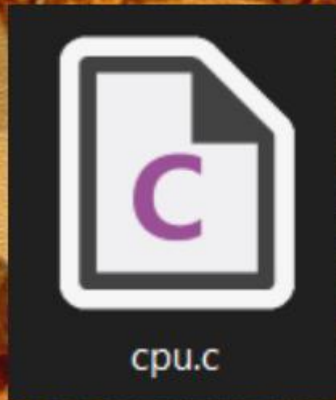
Figure 1–1. The Main Components and Operations of an Assembler.

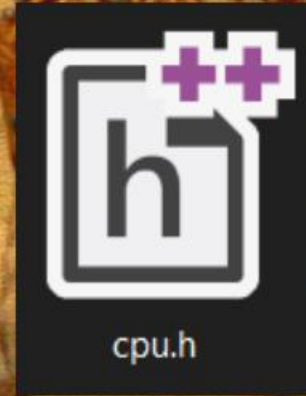
Quelle: "Assemblers and Loaders" von David Salomon, veröffentlicht 1993, das einzige Buch auf der Welt über Assemblerbau —_—

- Vasm: Assembler
- VBCC: C-Compiler









Eigener Assembler

Ich

```c
/*Returns the number of the register identifier, or a constant*/
static int parse_reg(char** start, int regtype)
{
    char* stringPos = *start;
    regsym* symbol;
    int registerIndex;
    unsigned int identifierLength;

    /*If string is an identifier...*/
    if (ISIDSTART(*stringPos))
    {
        stringPos++;
        while (ISIDCHAR(*stringPos)) stringPos++;

        /*...find that identifier and return the register number it references */
        identifierLength = stringPos - *start;

        if (symbol = find_regsym_nc(*start, identifierLength))
        {
            if (symbol->reg_type == regtype)
            {
                *start = stringPos;
                return symbol->reg_num;
            }

            return -1; /*In case register type is not the expected type, die*/
        }
    }
```

```
cgei R4, R2, 42
```

cpu.c

```
59   "cgei",    { TargetReg,SourceReg1,Immediate16Minus1 },  { FORMI(10) },
60   "clei",    { TargetReg,SourceReg1,Immediate16Plus1 },  { FORMI(11) },
61   "cgui",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(12) },
62   "clui",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(13) },
63   "cgeui",   { TargetReg,SourceReg1,Immediate16Minus1 },  { FORMI(12) },
64   "cleui",   { TargetReg,SourceReg1,Immediate16Plus1 },  { FORMI(13) },
65   "addis",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(16) },
66   "andi",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(17) },
67   "ori",     { TargetReg,SourceReg1,Immediate16 },  { FORMI(18) },
68   "xori",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(19) },
69   "xnori",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(20) },
70   "slli",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(22) },
71   "srli",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(23) },
72   "load",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(24) },
73   "load.s",  { TargetFloatReg,SourceReg1,Immediate16 },            { FORMI(25) },
```

Befehlsdeklaration

cpu.h

```c
40  /* operand types */
41  enum {
42      None=0,
43      Data,
44      SourceReg1,
45      SourceReg2,
46      TargetReg,
47      SourceFloatReg1,
48      SourceFloatReg2,
49      TargetFloatReg,
50      Immediate16,
51      Immediate16Plus1,
52      Immediate16Minus1,
53      Immediate16Label,
54      Immediate26Label
55  };
```

cpu.c

```c
59  "cgei",   { TargetReg,SourceReg1,Immediate16Minus1 },  { FORMI(10) },
60  "clei",   { TargetReg,SourceReg1,Immediate16Plus1 },  { FORMI(11) },
61  "cgui",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(12) },
62  "clui",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(13) },
63  "cgeui",  { TargetReg,SourceReg1,Immediate16Minus1 },  { FORMI(12) },
64  "cleui",  { TargetReg,SourceReg1,Immediate16Plus1 },  { FORMI(13) },
65  "addis",  { TargetReg,SourceReg1,Immediate16 },  { FORMI(16) },
66  "andi",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(17) },
67  "ori",    { TargetReg,SourceReg1,Immediate16 },  { FORMI(18) },
68  "xori",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(19) },
69  "xnori",  { TargetReg,SourceReg1,Immediate16 },  { FORMI(20) },
70  "slli",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(22) },
71  "srli",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(23) },
72  "load",   { TargetReg,SourceReg1,Immediate16 },  { FORMI(24) },
73  "load.s", { TargetFloatReg,SourceReg1,Immediate16 },          { FORMI(25) },
```

cpu.h

```c
77  /* instruction formats */
78  #define FORMR(x) ((x)&0x3f)
79  #define FORMI(x) ((0x20|((x)&0x1f))<<26)
80  #define FORMJ(x) ((0x10|((x)&0xf))<<26)
```

Was ist denn nun die Ausgabe unseres Assemblers?!?

Hier musst du jetzt Assembly Code zeigen

Linker

Was macht ein Linker?

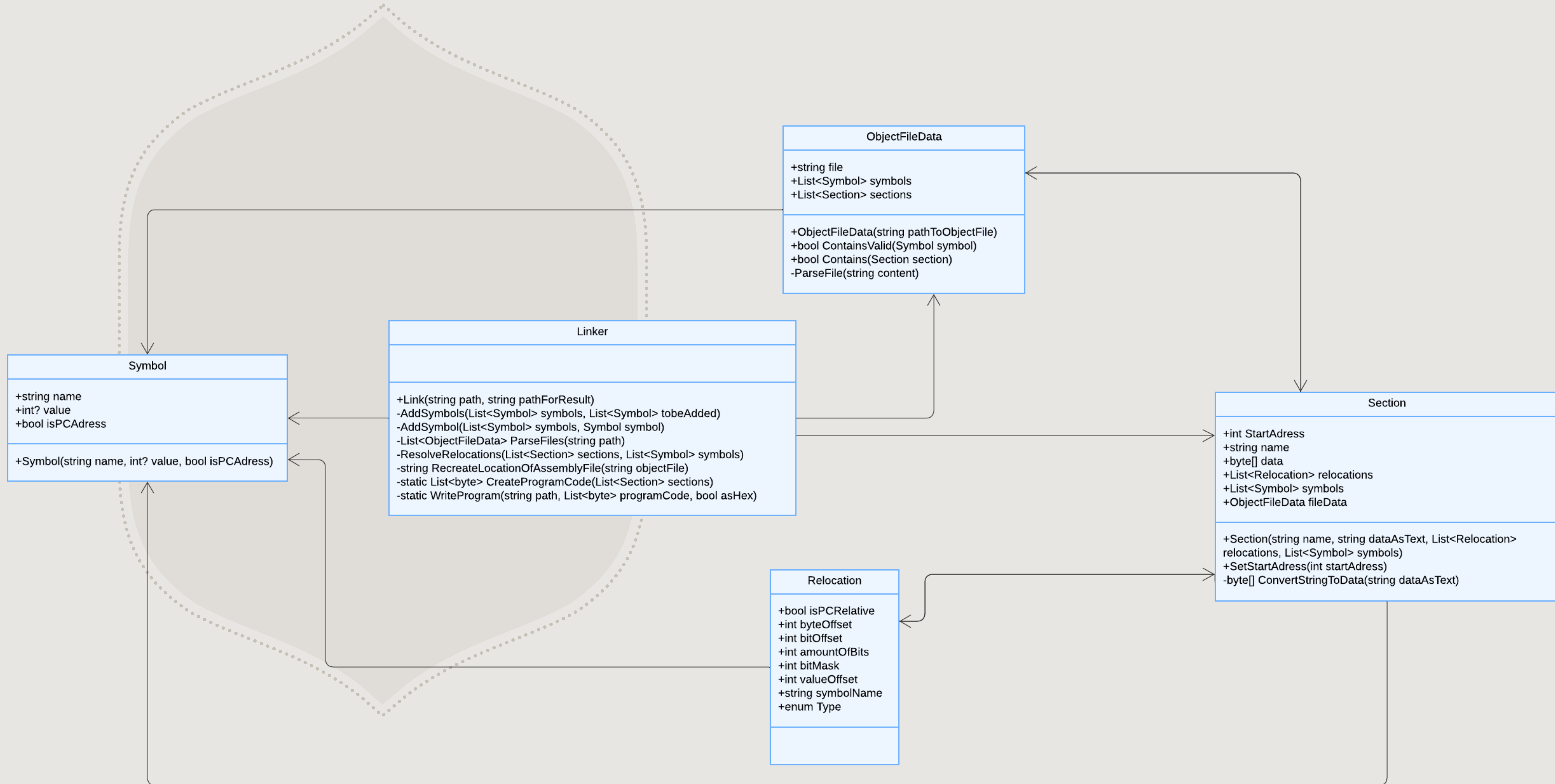
1. Statisches Linken

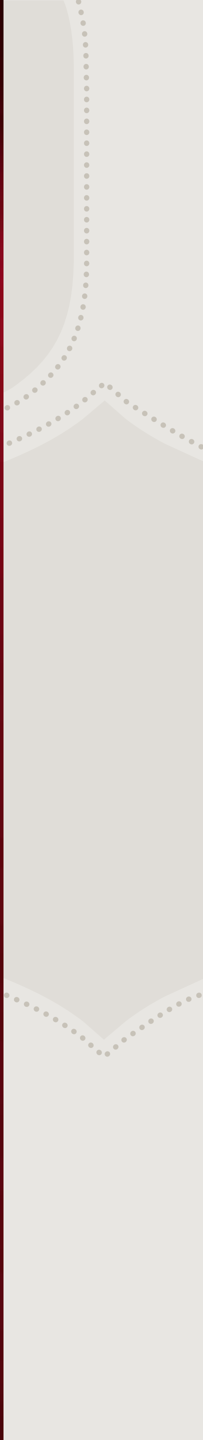
- Relokationen auflösen
- Objektdateien in eine .exe verschmelzen
- Bibliotheken einbinden

2. Dynamisches Linken

- Nochmal Relokationen auflösen
- Nochmal Bibliotheken einbinden

Unser Linker





Was ist denn nun die Ausgabe unseres ~~Assemblers~~?!?

Linkers

Hier musst du jetzt Assembly Code zeigen

Programmgröße

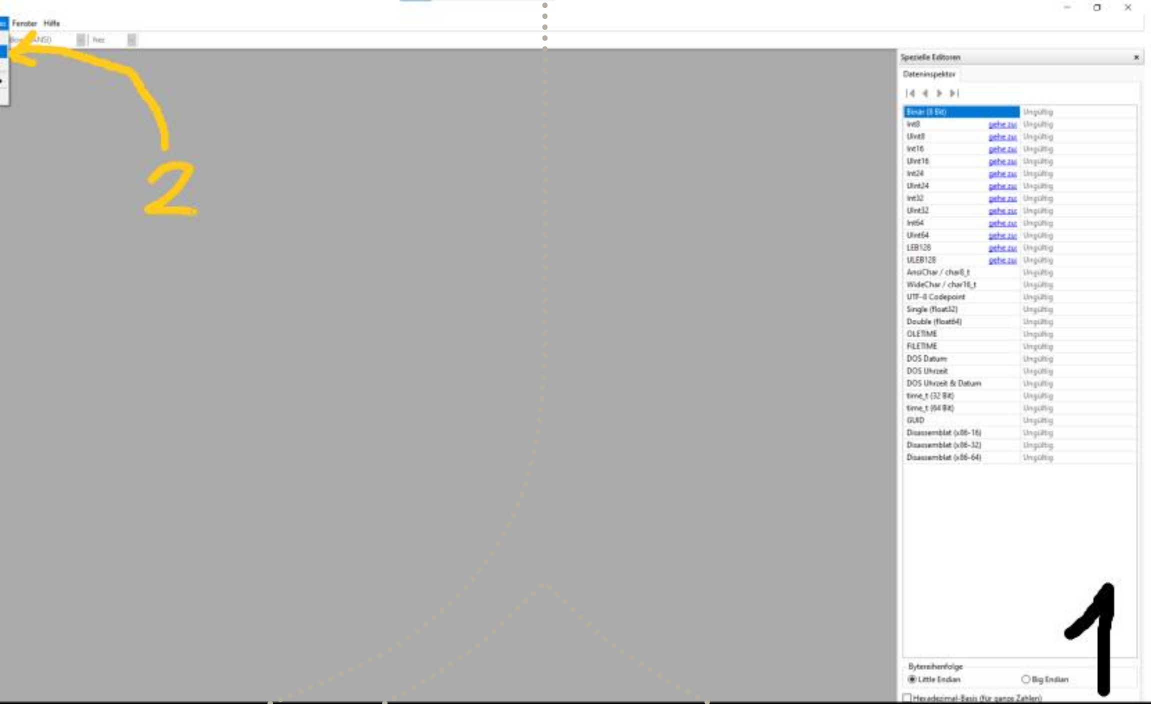
Startabschnitt

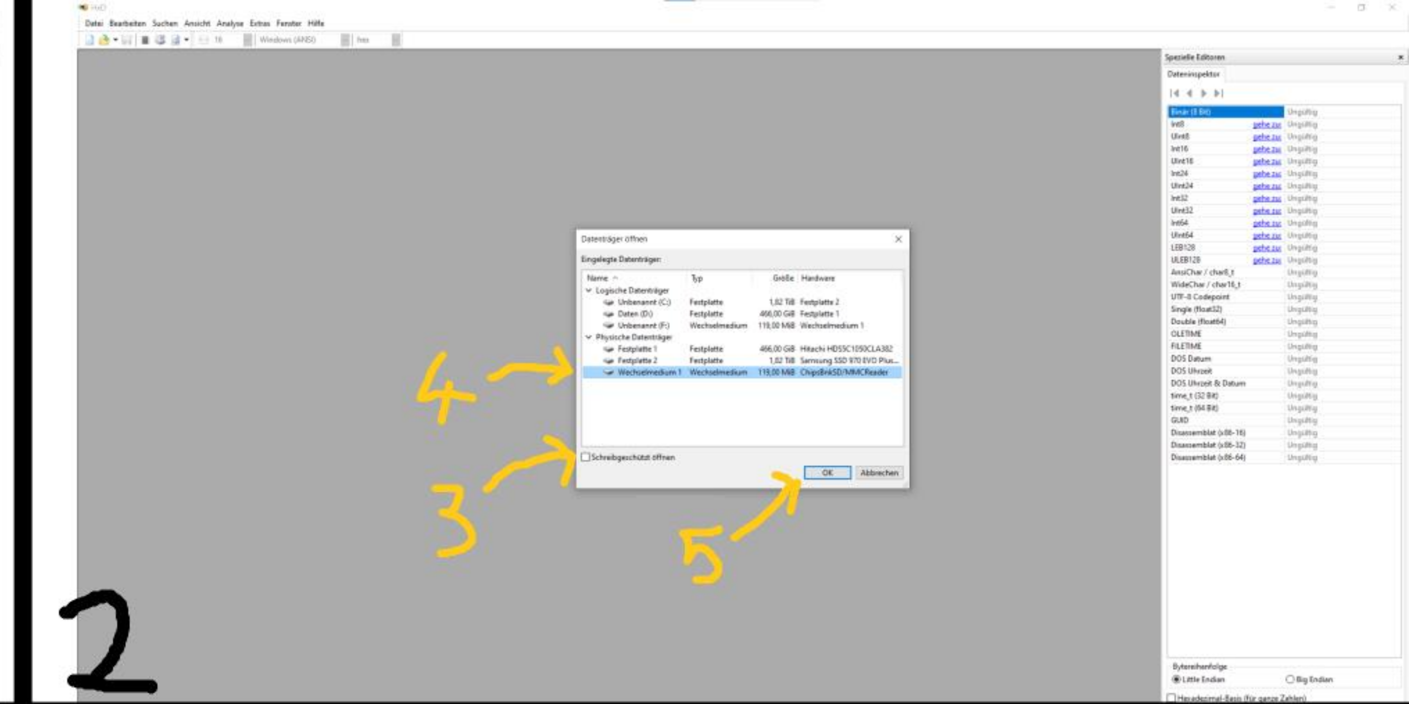

```
000002FB83E0FFFFC3FF000183C0000083A00006DBBD001883800008DB9C001883600000030000
00FFC00017885B00028042000CF0020000FFC0025A40000003FFC0026B40000001FFC00280C060
0700E80300000806000005D86300180063E808F803000383A00005DBBD001843FFFFEE83A00006DB
BD001843FFFFEB43FFFFFF00000000000000FF87FF0006EBDF0006E93F0005E91F0004E8FF0003
E8DF0002E8BF000180A0000A80C0000A80E0008C81000064E120001FFFC0002080A0000080C000
0080E000A08100000AE1200020FFC0001A80A0000080C0006E80E000A08100000AE1200020FFC0
001480A0000080C0000A80E0000A81000064E1200020FFC0000E80A0009680C0000A80E0000A81
000064E1200020FFC00008E3DF0006E13F0005E11F0004E0FF0003E0DF0002E0BF000183FF0006
F01E000087FF0005EBDF0005E89F0004E87F0003E85F0002E83F000100253800008640000D88400
08D846000800451000000062E800E92300008042000 1C46200FF00630808F403FFFA004208018042
0100006220080042280 0F403FFF5E3DF0005E09F0004E07F0003E05F0002E03F000183FF0005F0
1E000087FF0005E85F0001E8BF0002E8DF0003E8FF0004EBDF000580C00000A8E50009F4070006
94E5000A87FF0001E8FF000180C6000190A5000A43FFFFF887FF0001E8BF000180C60001A4E600
00F407001BE0FF000183FF000184C6000188E7000280E70084F0070000FFC0001B40000011FFC0
002B4000000FFFC000364000000DFFC00045400000 0BFFC0005440000009FFC000624000 0007FF
C0007140000005FFC0008140000003FFC0008E40000001FFC0009F8042000443FFFFE3E05F0001
E0BF0002E0DF0003E0FF0004E3DF000583FF0005F01E000087FF0001E8BF0001D8A3000800A510
0000A5E800E8850000E8850001E8850002E8850100E8850102E8850200E8850202E8850300E885
0301E8850302E0BF000183FF0001F01E000087FF0001E8BF0001D8A3000800A5100000A5E800E8
850000E8850001E8850101E8850201E8850301E0BF000183FF0001F01E000087FF0001E8BF0001
D8A3000800A5100000A5E800E8850000E8850001E8850002E8850102E8850200E8850201E88503
00E8850301E8850302E0BF000183FF0001F01E000087FF0001E8BF0001D8A3000800A5100000A5
E800E8850000E8850001E8850002E8850101E8850102E8850202E8850300E8850301E8850302E0
BF000183FF0001F01E000087FF0001E8BF0001D8A3000800A5100000A5E800E8850000E8850002
```

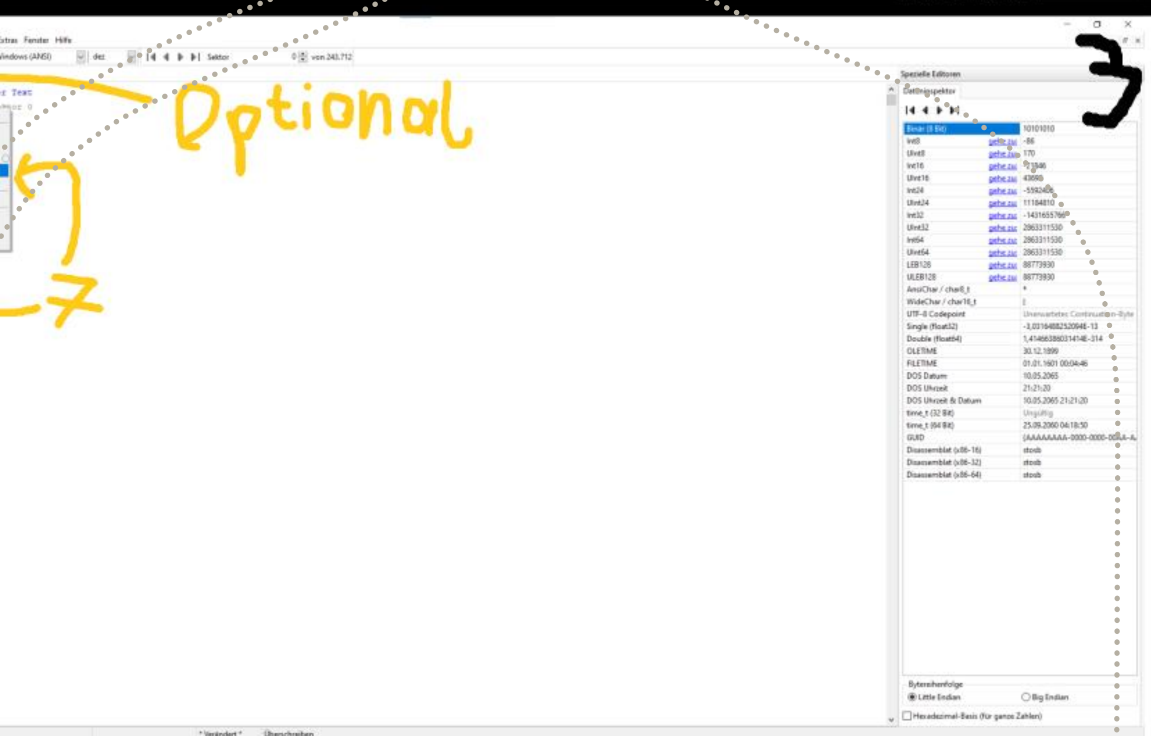
Der Loader

in zwei Akten

1. Akt – HxD

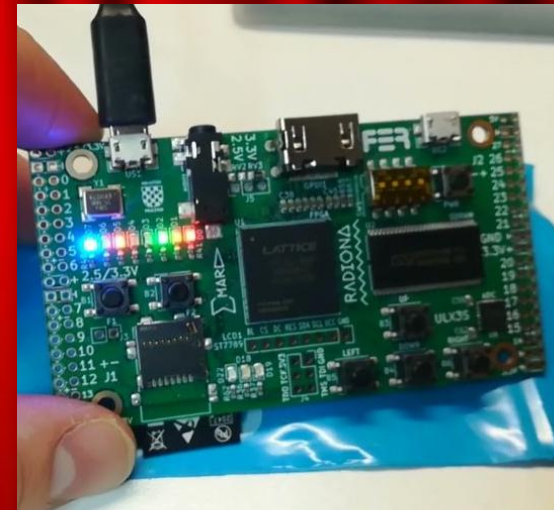

2
1

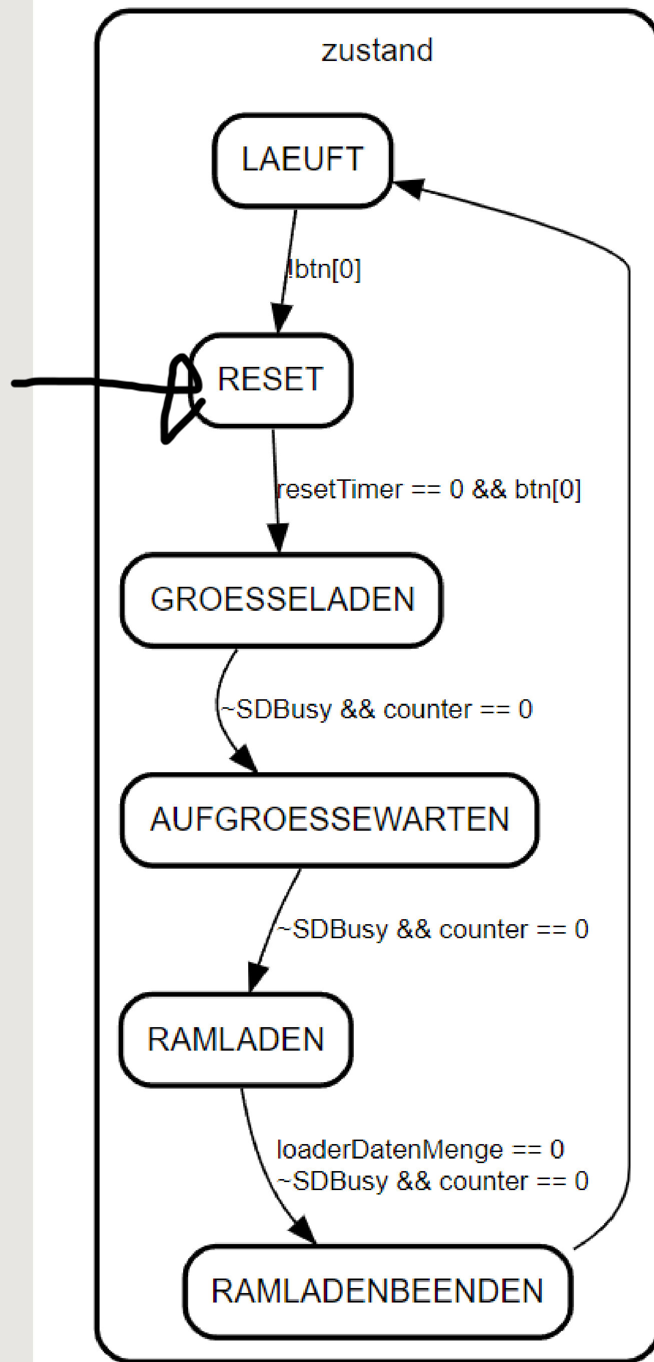

2
4
3
5


Optional
7
3

4

2. Akt – Hans




zustand
LAEUFT
!btn[0]
RESET
resetTimer == 0 && btn[0]
GROESSELADEN
~SDBusy && counter == 0
AUFGROESSEWARTEN
~SDBusy && counter == 0
RAMLADEN
loaderDatenMenge == 0
~SDBusy && counter == 0
RAMLADENBEENDEN

Entwickeln für den Hans

1. Hans Assembly schreiben
2. Mit Programmbauer assemblieren und linken
3. Über HxD auf SD-Karte laden
4. In ULX3s reinstecken und Strom anschließen

Was haben wir entwickelt?

Was haben wir entwickelt?

Gameplay

Unsere Registerkonventionen

```
Globale Register:
R31 - Stapelzeiger (wächst nach unten, zeigt auf die nächste freie Adresse)
R30 - Rücksprungzeiger
R29 - Bildpufferadrese
R28 - Knopfadresse
R27 - Spielzustand
```

Konzept: @l, @h, @ha

```
.set Zahl, 923692430 ; Bitmuster: 00110111.00001110.01101101.10001110

;                                              Zahl | Ganzzahl
;-----------------------------              -------------------------
Addi  R2, R0, Zahl@l ; Immedi               046   |  28046
Addis R2, R2, Zahl@h ; Immedi               094   |  14094
```

Kondsebt: @l, @h, @ha

```
.set Zahl, 923725198 ; Bitmuster: 00110111.00001110.11101101.10001110

;                                   Bitmuster      | Nat. Zahl | Ganzzahl
;-----------------------------------------------------------------------
Addi  R2, R0, Zahl@l ; Immediate: 11101101.10001110 |   60814   |  -4722
Addis R2, R2, Zahl@h ; Immediate:  00110111.00001110 |   14094   |  14094
```

Gonnsebt: @l, @h, @ha

```
.set Zahl, 923725198 ; Bitmuster: 00110111.00001110.11101101.10001110

;                                    Bitmuster      | Nat. Zahl | Ganzzahl
;-------------------------------------------------------------------------------
Addi  R2, R0, Zahl@l ; Immediate:  11101101.10001110 |   60814   |  -4722
Addis R2, R2 , Zahl@ha; Immediate:  00110111.00001111 |   14095   |  14095
```

Konzept: Sprungtabelle

```
;R7 enthält die Zahl, die gezeichnet werden soll

Cei R8, R7, 0
Bez R8, Zeichne1
Jal R30, NullZeichnen ; R7 == 0
Jmp NachZifferZeichnen

Zeichne1:
Cei R8, R7, 1
Bez R8, Zeichne2
Jal R30, EinsZeichnen ; R7 == 1
Jmp NachZifferZeichnen

Zeichne2:
Cei R8, R7, 2
Bez R8, Zeichne3
Jal R30, ZweiZeichnen ; R7 == 2
Jmp NachZifferZeichnen

Zeichne3:
Cei R8, R7, 3
Bez R8, Zeichne4
Jal R30, DreiZeichnen ; R7 == 3
Jmp NachZifferZeichnen

Zeichne4:
Cei R8, R7, 4
Bez R8, Zeichne5
Jal R30, VierZeichnen ; R7 == 4
Jmp NachZifferZeichnen
```

```
;R7 enthält die Zahl, die gezeichnet werden soll

Muli R7, R7, 2; Versatz berechnen
Addi R7, R7, ZiffernZeichnenSprungTabelle - 1
Jreg R7

ZiffernZeichnenSprungTabelle:
Jal R30, NullZeichnen ; R7 == 0
Jmp NachZifferZeichnen

Jal R30, EinsZeichnen ; R7 == 1
Jmp NachZifferZeichnen

Jal R30, ZweiZeichnen ; R7 == 2
Jmp NachZifferZeichnen

Jal R30, DreiZeichnen ; R7 == 3
Jmp NachZifferZeichnen

Jal R30, VierZeichnen ; R7 == 4
Jmp NachZifferZeichnen

Jal R30, FuenfZeichnen ; R7 == 5
Jmp NachZifferZeichnen

Jal R30, SechsZeichnen ; R7 == 6
Jmp NachZifferZeichnen

Jal R30, SiebenZeichnen ; R7 == 7
Jmp NachZifferZeichnen
```

Freestyle Code zeigen

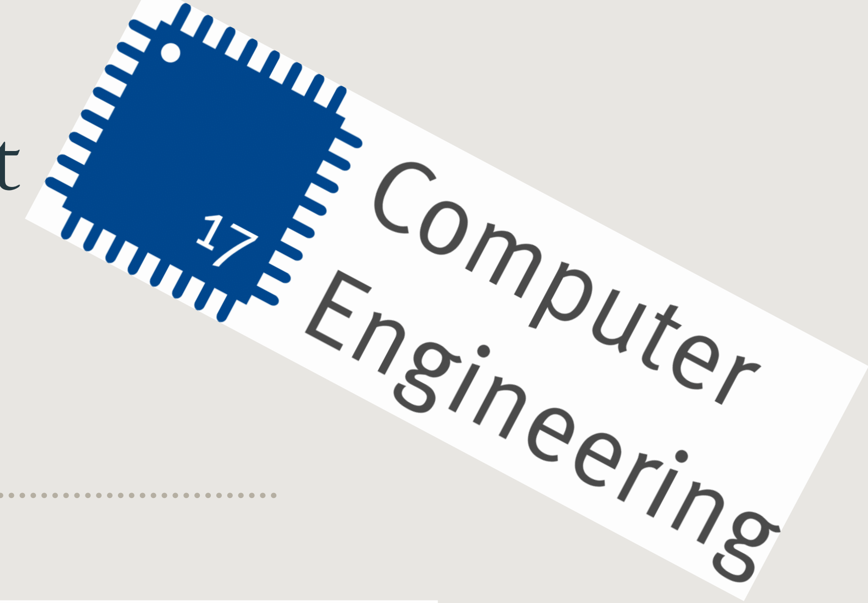
Interessante Dateien:

- Start.hasm
- Anzeige.hasm
- Spieler.hasm
- Ball.hasm

Hans 2 Starterpacket

Hans 1

Gesponsortes Segment
–
Angebotene Module



Rechenanlagen	Rechnerarchitektur	Virtual Prototyping of Embedded Systems with SystemC
Amateurfunk (Klasse E)	Embedded Processor Lab (Hardwarepraktikum)	Seminar Advanced Computer Architecture

Danke fürs Zuhören

https://github.com/Byter64/Hans/wiki

Besonderen Dank an:

- **Goran Mahovlić**
- Jan Rinze
- Frank Wille
- Moritz Lotz