

Asynchronous DRAM over SPI

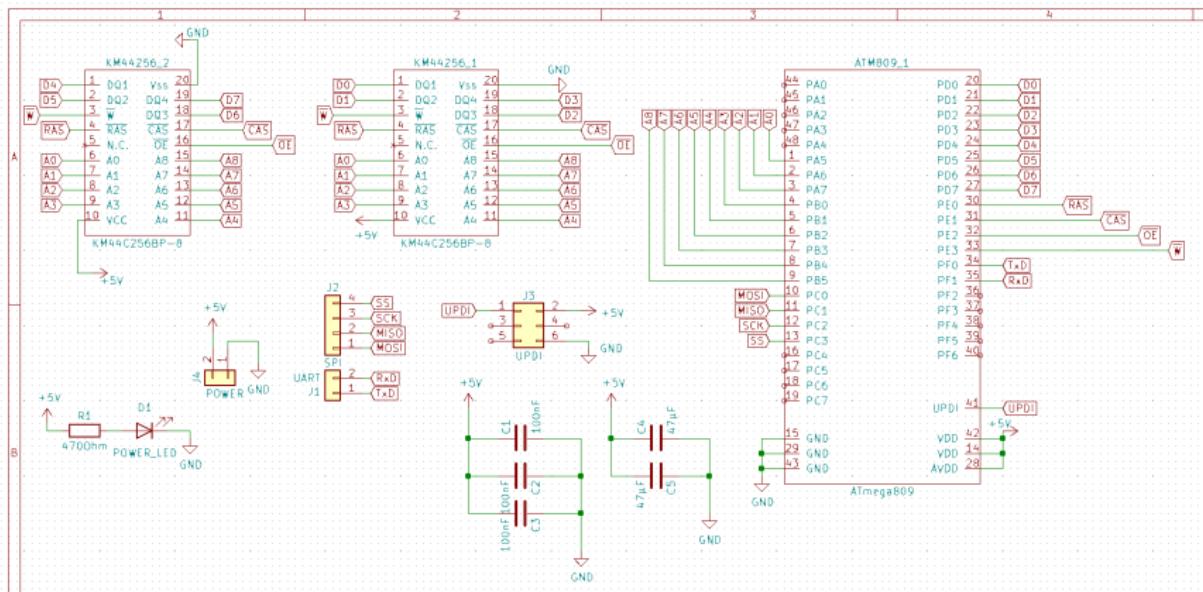
Implementierung eines DRAM-Controllers mit Web-API

Maximilian Gaul

04.02.2020

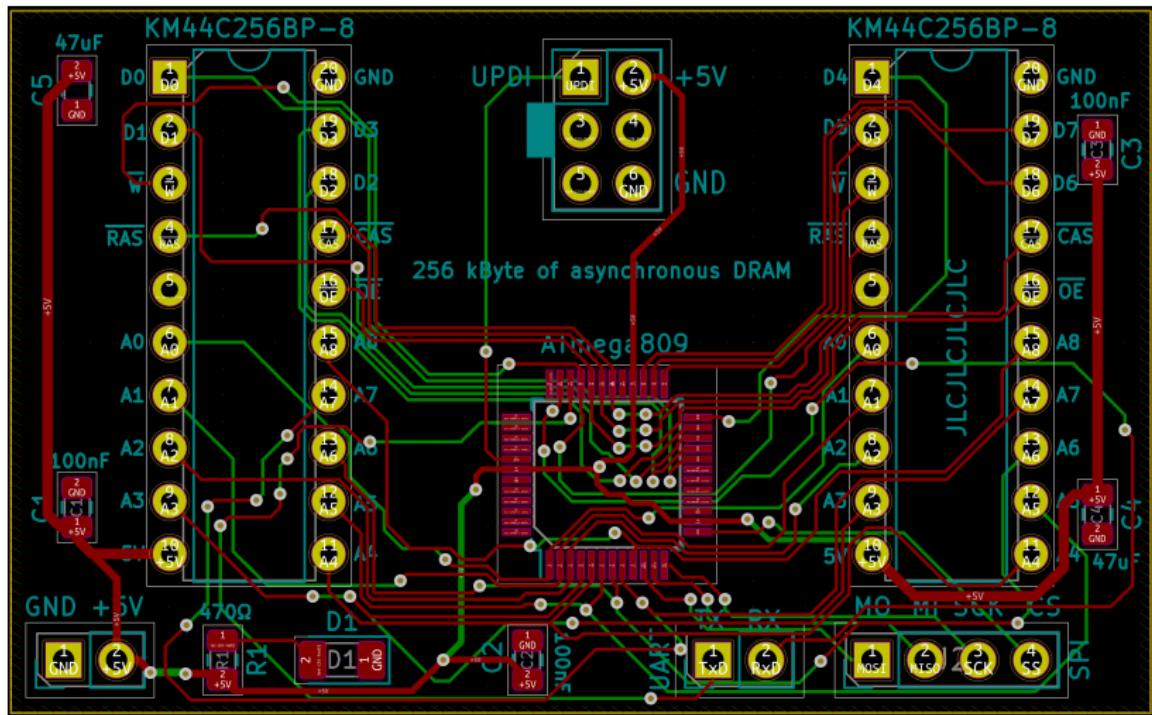
Das PCB

Schematic



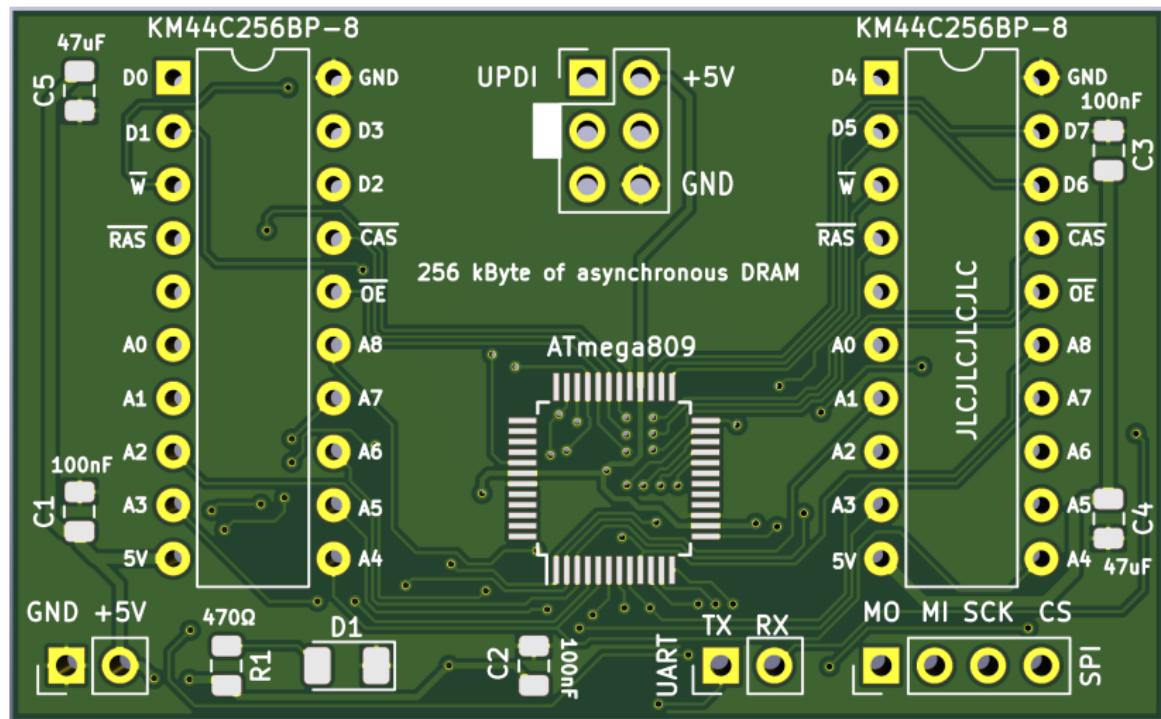
Das PCB

Layout



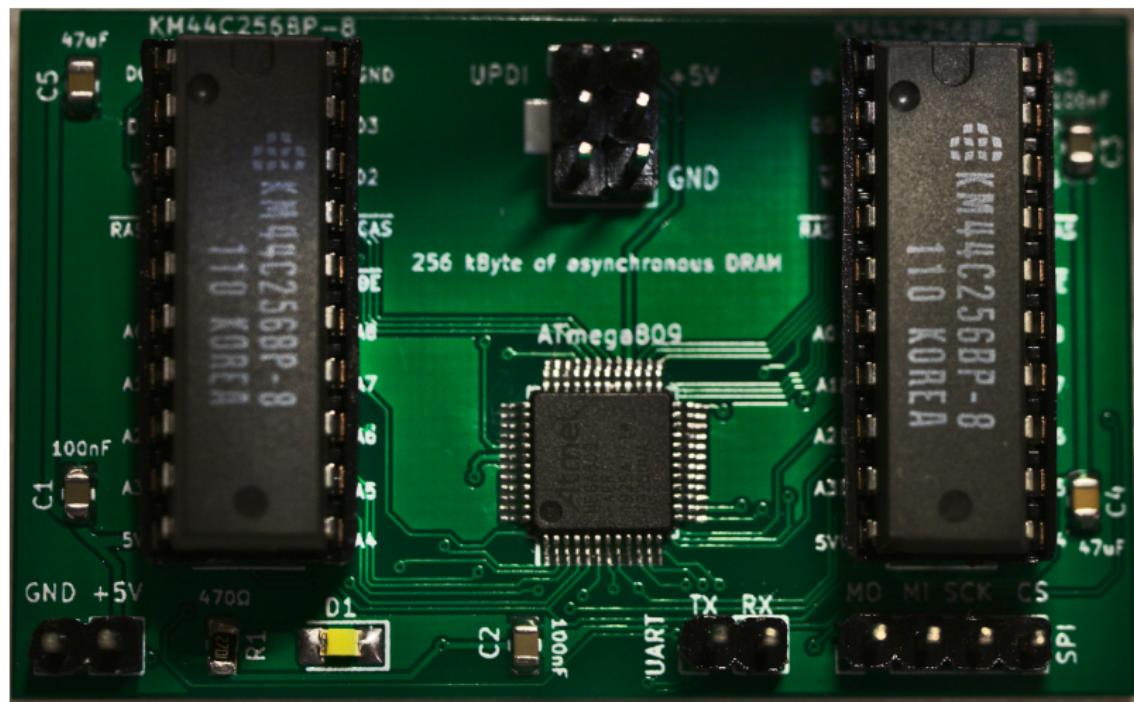
Das PCB

Layout



Das PCB

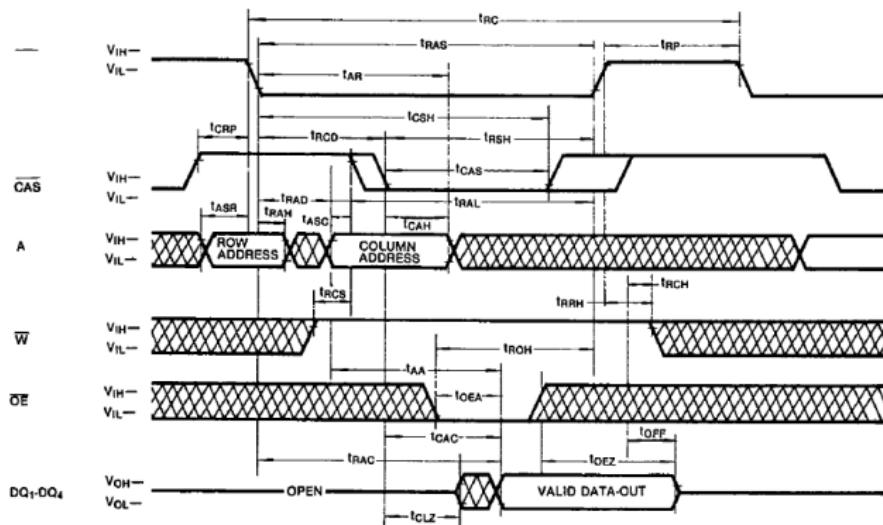
Fertige Platine



- 9 Adresseingänge
 - Werden sowohl für Spalten- als auch Zeilenadressen verwendet
 - $2^{18} = 262144$ RAM-Zellen
- 4 Dateneingänge
 - Mit 2 Stück erhält man volle 256kByte
- 8ms Refresh Period
- Asynchroner DRAM \Rightarrow keinen Clock Eingang
- Neuere SDR-DRAM oder DDR-DRAM erfordern zu hohe Geschwindigkeiten

TIMING DIAGRAMS

READ CYCLE

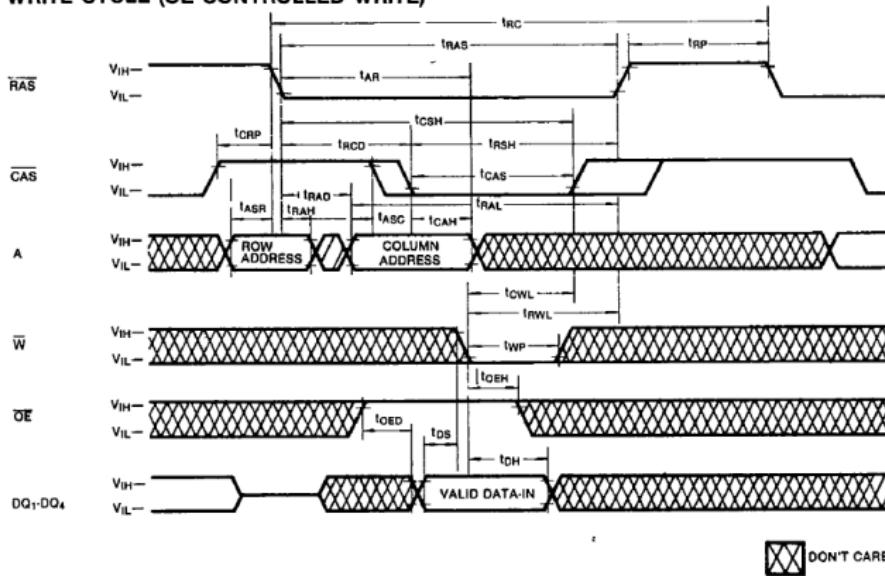


DON'T CARE

KM44C256BP-8

WRITE

WRITE CYCLE (OE CONTROLLED WRITE)



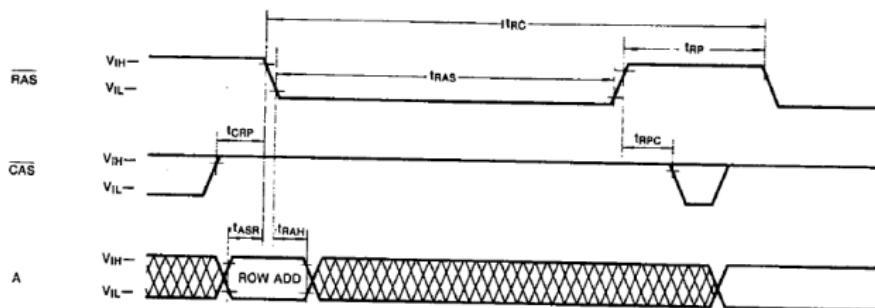
DON'T CARE

T-46-23-17

TIMING DIAGRAMS (Continued)

RAS-ONLY REFRESH CYCLE

Note: \overline{W} , \overline{OE} = Don't care



ATmega809

- Reagiert auf eintreffende SPI-Bytes
- Erkennt READ bzw. WRITE Befehle, steuert Adress- und Datenleitungen zum RAM entsprechend
- Führt zyklische $R\bar{A}S$ -Refreshes aus

ATmega809

```
1 int main(void) {
2     initDRAMHandler(&dramHandler);
3
4     initCPU();
5     initSPI();
6     initTimer0();
7
8     while (1) {
9         if (dramHandler.hasPendingRefresh) {
10             dramHandler.refreshRASonly(&dramHandler);
11             dramHandler.hasPendingRefresh = false;
12         }
13         if (dramHandler.hasPendingBufferUpdate) {
14             dramHandler.processAndRespondBuffer(&dramHandler);
15             dramHandler.hasPendingBufferUpdate = false;
16         }
17     }
18 }
```

ATmega809

```
1 ISR(TCA0_CMP0_vect) {
2     dramHandler.hasPendingRefresh = true;
3
4     /* Clear interrupt flag */
5     TCA0.SINGLE.INTFLAGS |= (1 << TCA_SINGLE_CMP0EN_bp);
6 }
7
8 ISR(SPI0_INT_vect) {
9     if(SPI0.INTFLAGS & SPI_RXCIE_bm) {
10         const uint8_t data = SPI0.DATA;
11         dramHandler.buffer.push(&dramHandler.buffer, data);
12         dramHandler.hasPendingBufferUpdate = true;
13     }
14 }
```

ESP8266

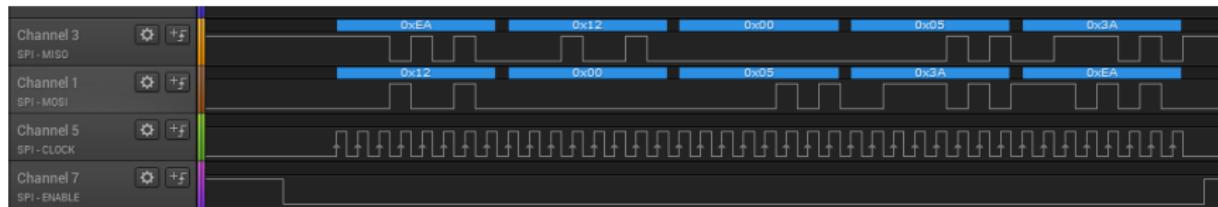
- Der ESP8266 agiert als SPI-Master
- Reagiert auf HTTP-Requests
- Sendet ggf. *READ*/*WRITE*-Befehle an ATmega809
- Zeigt Ergebnisse von READ-Befehlen an

ESP8266

WRITE-Befehl

- Besteht aus

$0x12, ADDR_MSB, ADDR_MLS, ADDR_LSB, DATA$



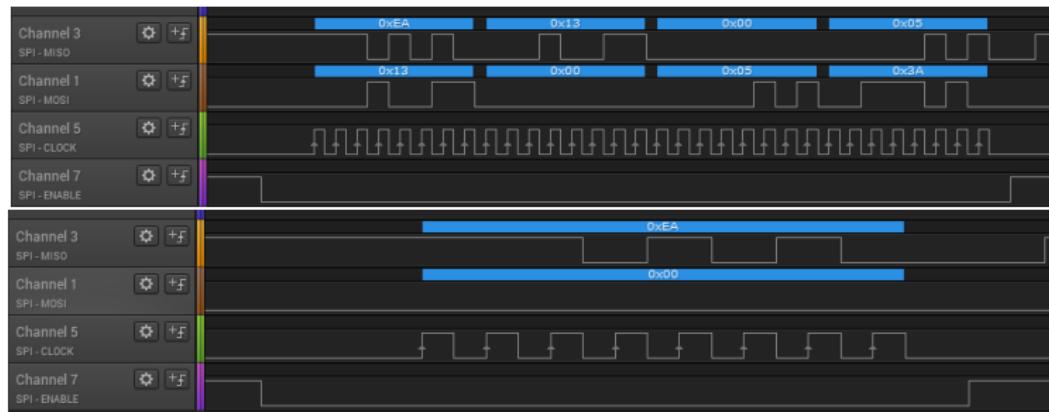
ESP8266

READ-Befehl

- Besteht aus

$0x13, ADDR_MSB, ADDR_MLS, ADDR_LSB$

RET_VAL



ESP8266

Web-API

- 192.168.4.1/get?addr=xxxxxx setzt READ-Befehl per SPI ab
 - Zeigt ausgelesenen Wert auf der Webseite an
- 192.168.4.1/set?addr=xxxxxx&value=yyy setzt WRITE-Befehl per SPI ab

Demo