


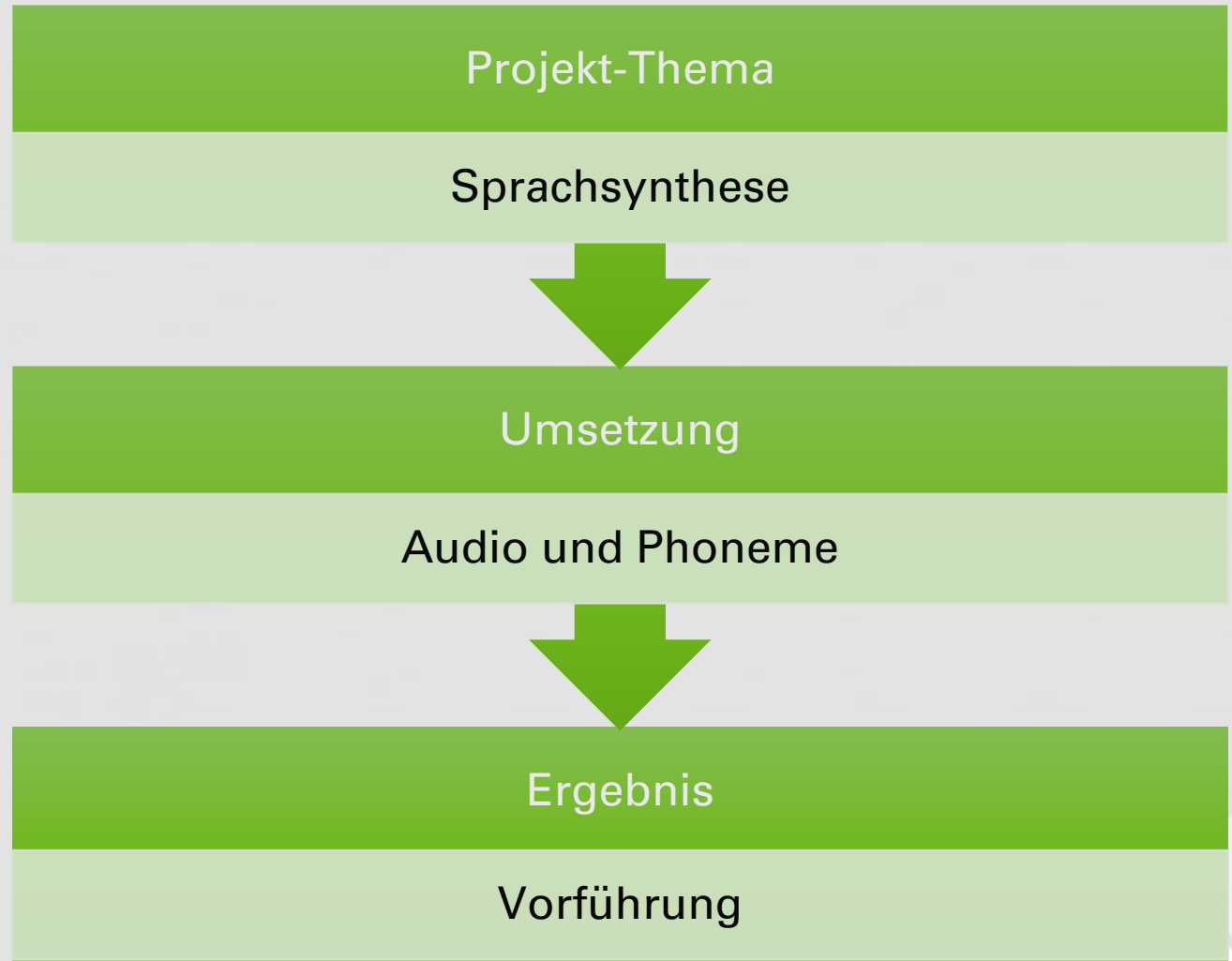
# Morse to Speech

Entwurf Digitaler Systeme (EDS)

Kirsebauer – Neumann



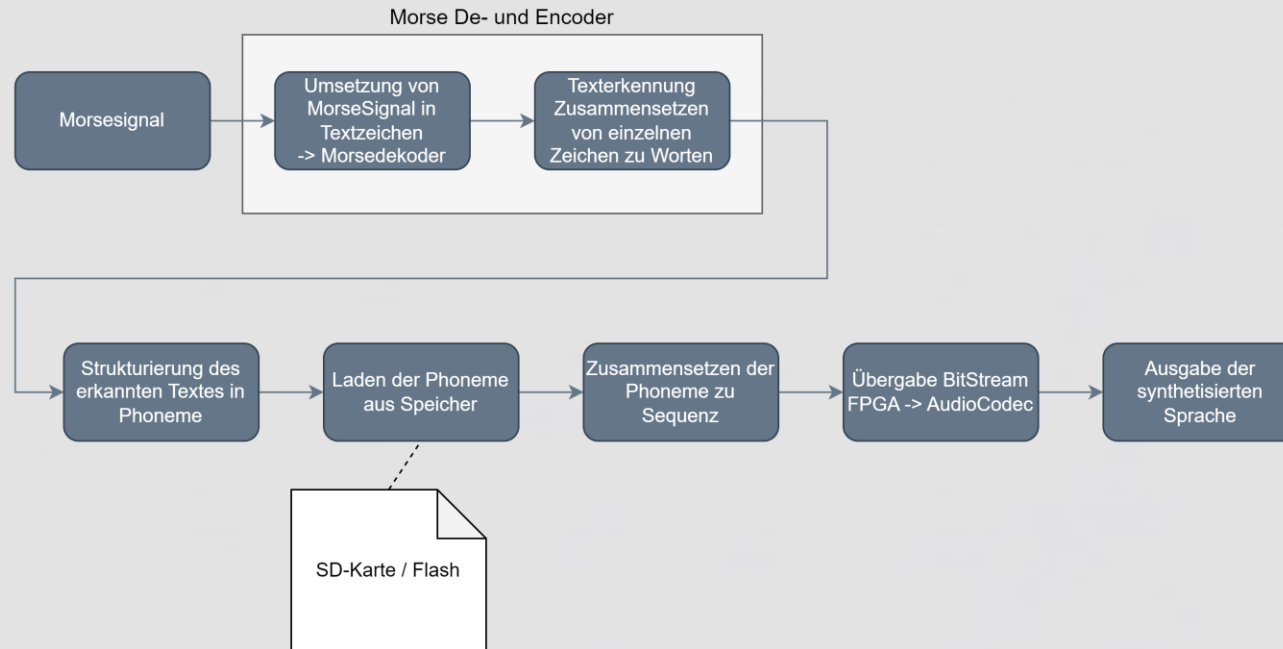
# Inhalt



# Projektthema

## Konkatenative Sprachsynthese auf einem FPGA

- Proof of Concept mit Phonemsynthese
- Hardwarebeschreibung bietet Möglichkeit für Erweiterung auf Diphonsynthese
- als Segmentvorrat orientiert sich an Phonemen aus Quelle [1], zuzüglich Tonsegmente für x, y, z

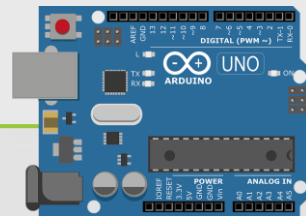


# Projekt-Thema

## Moderne Sprachsynthese

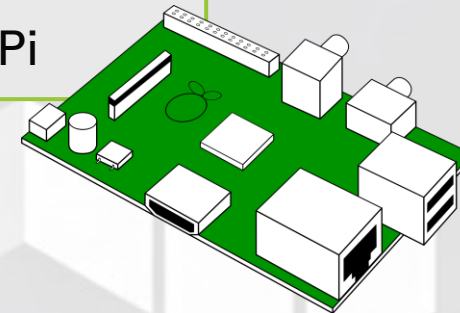
### Voraufgenommene Tonsequenzen

- Wörter (z.B. Talkie Bibliothek für Arduino)
- ganze Sätze



### KI / Deep Learning

- z.B. Sprachausgabe ChatGPT, Thorsten-Voice etc.
- Piper TTS für Raspberry Pi

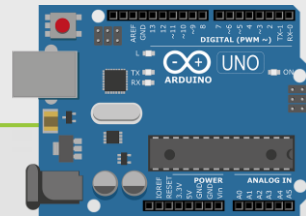


# Projekt-Thema

## Moderne Sprachsynthese

### Voraufgenommene Tonsequenzen

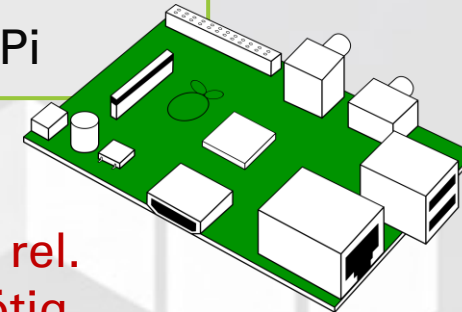
- Wörter (z.B. Talkie Bibliothek für Arduino)
- ganze Sätze



- Nachrichten müssen deterministisch sein

### KI / Deep Learning

- z.B. Sprachausgabe ChatGPT, Thorsten-Voice etc.
- Piper TTS für Raspberry Pi



- Internetverbindung oder rel. hohe Rechenkapazität nötig

# Projekt-Thema

## Konkatenative Sprachsynthese

- Konkatenative  $\Leftrightarrow$  Zusammenketten
- Voraufgenommene Tonsequenzen werden möglichst kurz gewählt
- Sequenzen dienen als Bausteine um **alle möglichen Kombinationen** von Lauten in einer Sprache abzudecken
- $\rightarrow$  auch nicht deterministische Nachrichten ausgebbar
- für einen buchstabierende Sprachsynthese würden z.B. schon alle Buchstaben reichen, um jedes mögliche Wort zu bilden

P A L M E

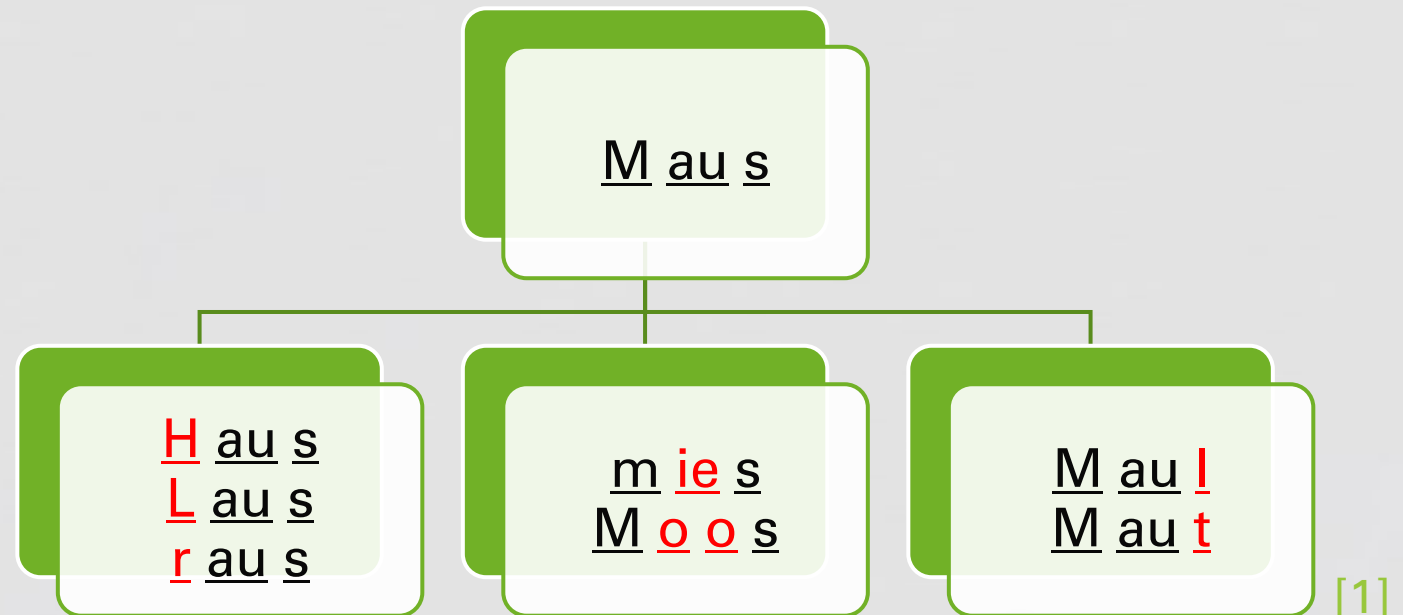
L A M P E

A M P E L

# Projekt-Thema

## Konkatenative Sprachsynthese

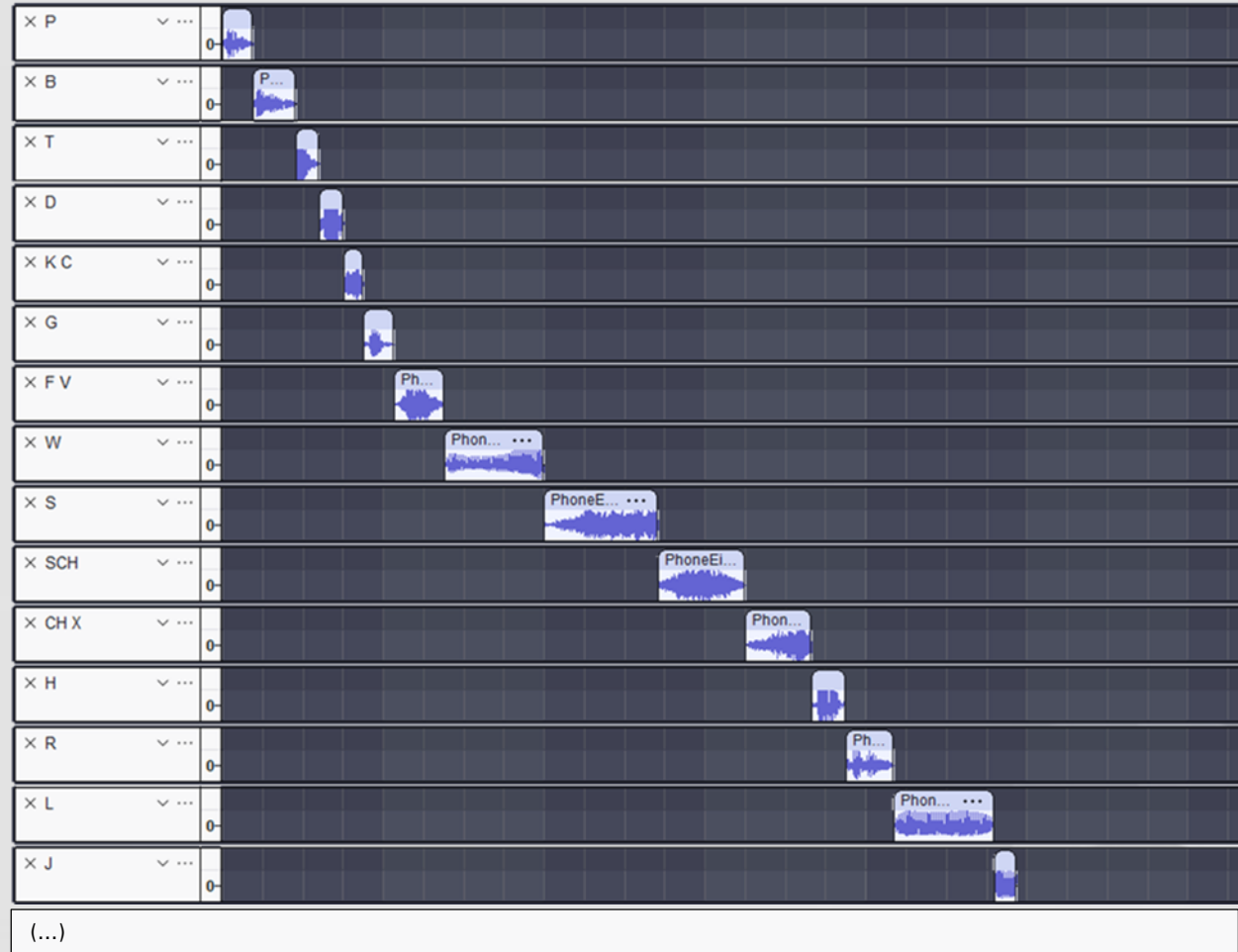
- kleinste bedeutungsunterscheidende Laute einer Sprache sind die Phoneme
- Die deutsche Sprache hat ca. 40 Phoneme
- Sprachsynthese in den 80ern basierte auf Kombination von Phonemen z.B. ATARI Voice Box
- In den 2000 Jahren: Diphonsynthese, nicht Phoneme sondern Phonemübergänge werden kombiniert (z.B. The MBROLA Project) 40 Phoneme ergeben theoretisch 1600 Diphone



# Umsetzung Aufzeichnung Audio



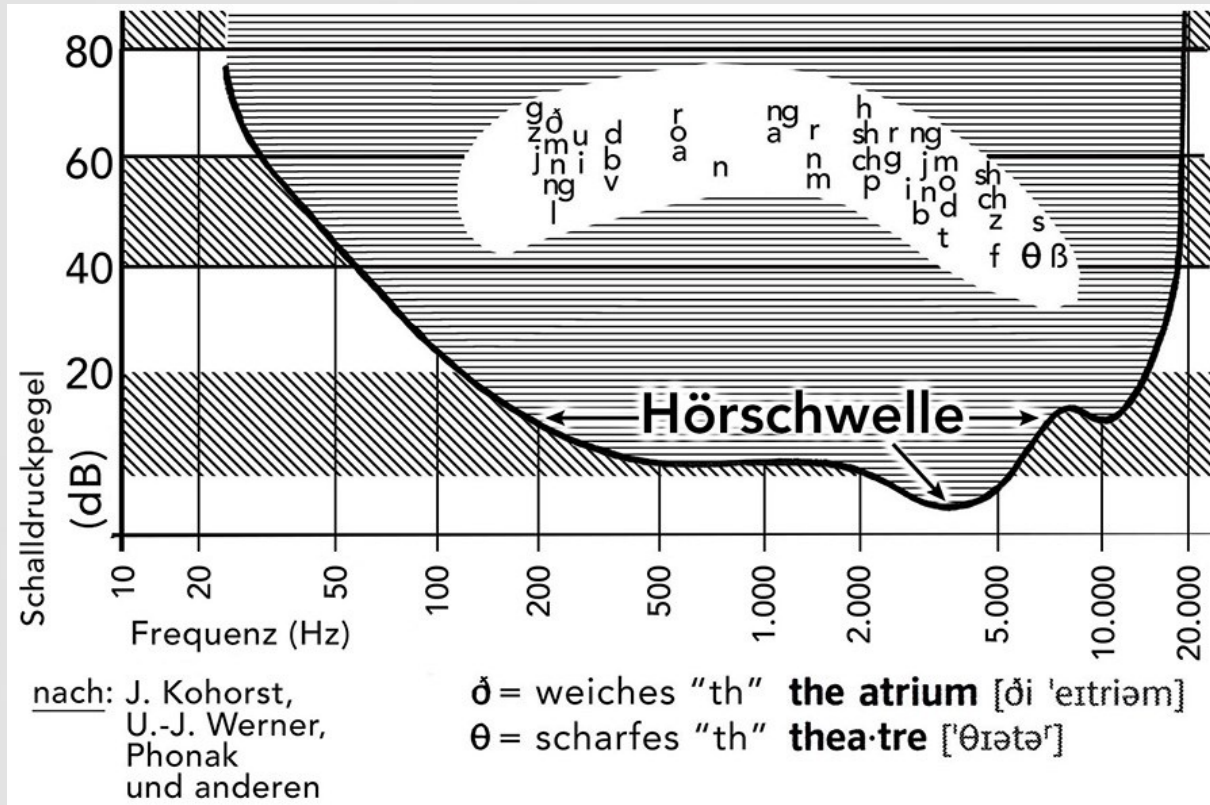
PhoneEinzeln\_Normalisiert\_EndeGeschnitten\_ueberarbeitet\_32kHz\_XYZ.aup3



# Umsetzung

## Audio

- original Morse De- und Encoder nutzt 48 kHz und 16 Bit Audio um 500 Hz Ton abzuspielen
- menschliche Stimme im Bereich von 200 Hz bis 8 kHz
- => Audio in diesem Projekt 32 kHz und 16 Bit Mono-Audio



[2]

# Audiocodec

## Wolfson WM8731

- Benötigt für den USB-Mode 12MHz als Inputsignal, die intern auf die Abtastrate von 32kHz herunter geteilt werden
- Problem: Wird durch den clock\_divider mit 12,5MHz gespeist
- -> Daher Rauschen in der Sprachausgabe

## • PLL (Phased-Locked-Loop)

- Erzeugt einen Ausgangstakt durch Synchronisation mit einem Eingangstakt (Referenzsignal)
- Eignet sich anstelle eines reinen Taktteilers zur Erzeugung beliebiger Takte
- Dabei wird das geteilte Ausgangssignal ( $f_{out} / N$ ) mit dem geteilten Eingangssignal ( $f_{ref} / M$ ) durch einen Phasen-/Frequenzdetektor verglichen und bei Bedarf  $f_{out}$  (ein VCO spannungsgesteuerter Oszillator) nachgeregelt.
- Im locked Zustand wird  $f_{out}$  konstant gehalten, es gilt  $f_{out}/N=f_{ref}/M \rightarrow f_{out}=(N/M)*f_{ref}$

# Umsetzung

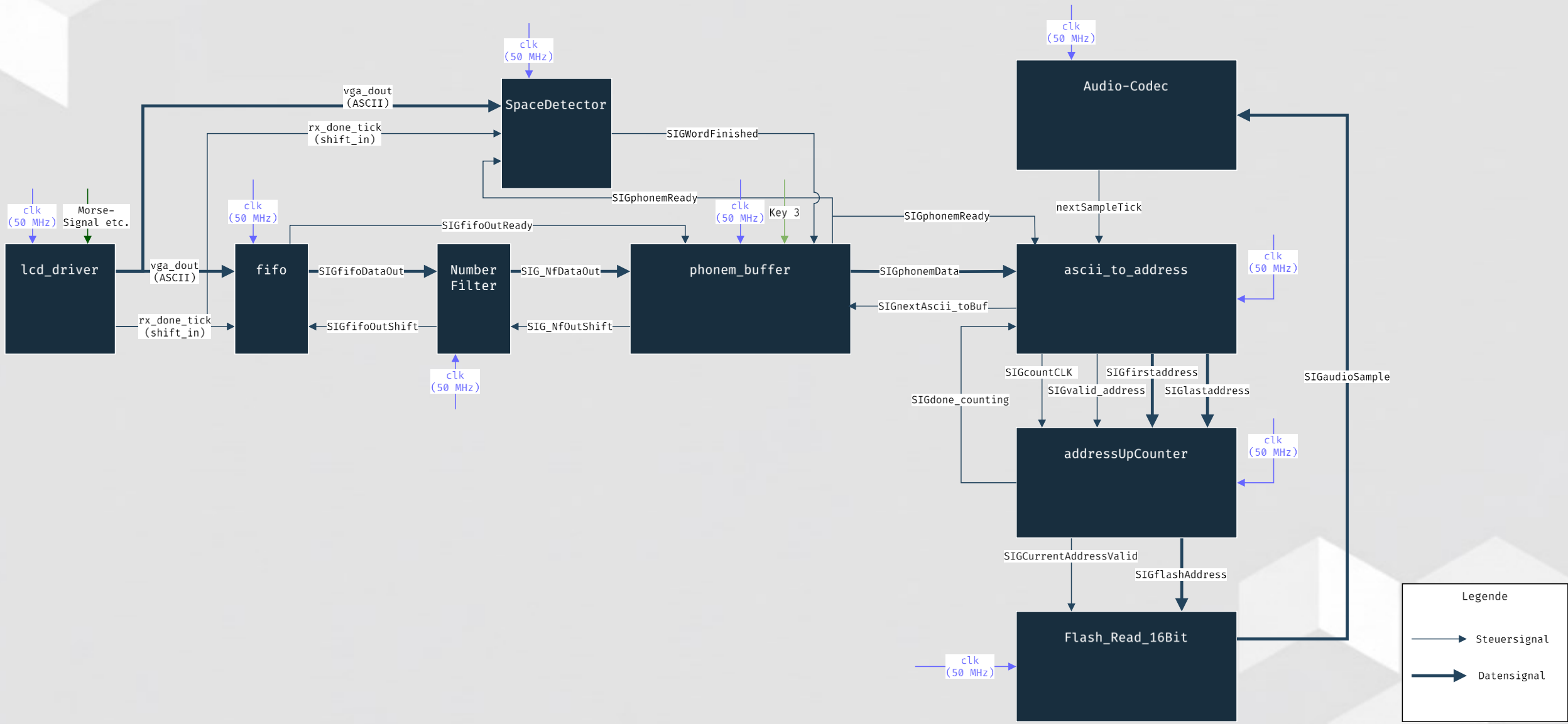
## Audio Speicher

- Flash bietet 8 Mbyte
- => 4.000.000 16 Bit Wörter
- Mit der Abtastrate ergeben sich 2 Minuten Audio aus dem Flash

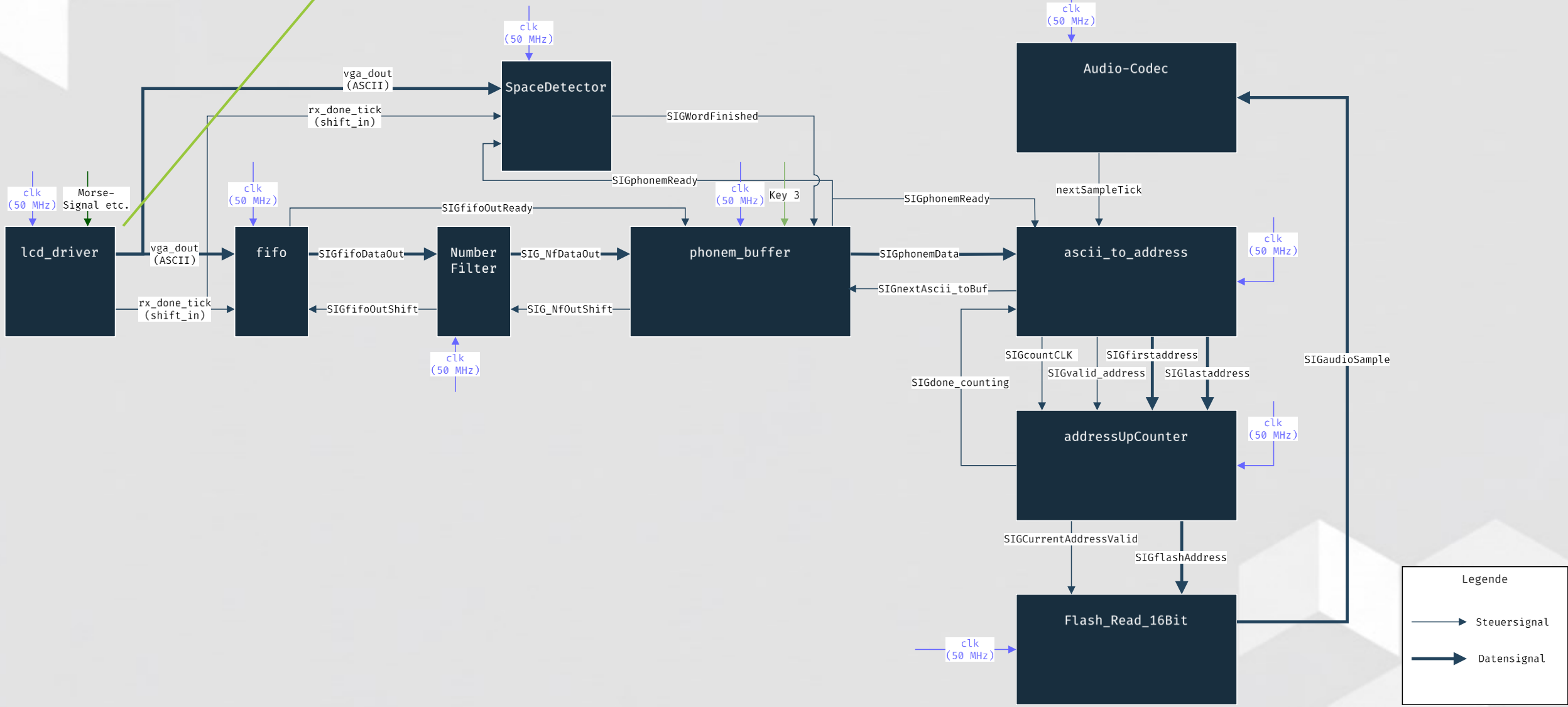
$$\frac{1}{32 \cdot 10^3 \text{ Hz}} \cdot 4 \cdot 10^6 \cdot \frac{1}{60 \text{ s}} \approx 2 \text{ min}$$

- Dateien können über das DE2\_115\_ControlPanel vom PC in den Flash abgelegt werden

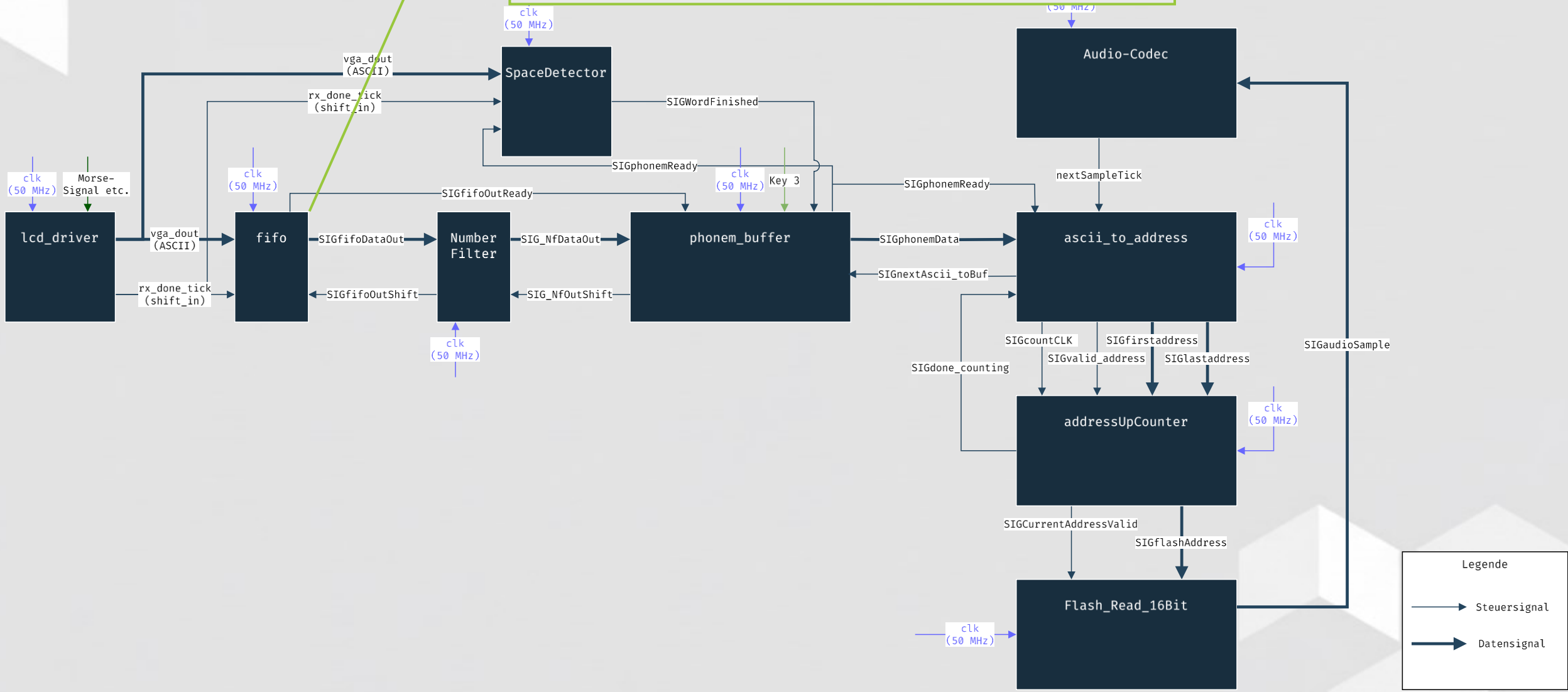


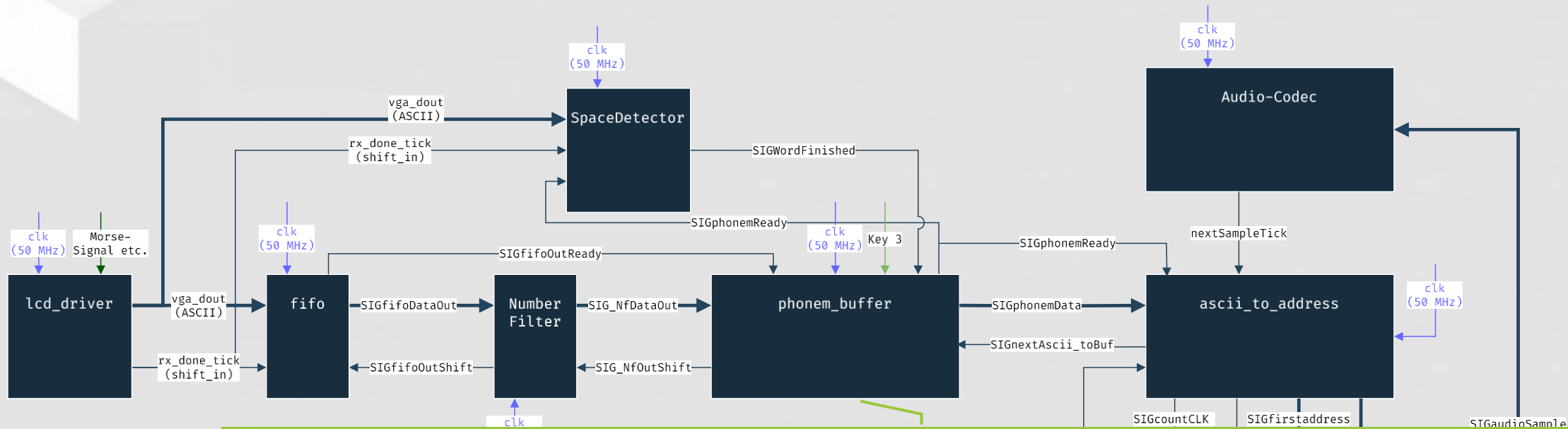


• ordnet Morse-Code jeweiliges ASCII-Zeichen zu

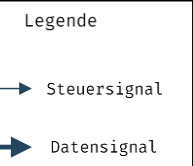
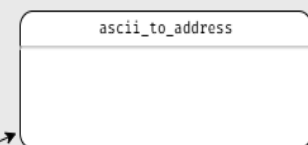
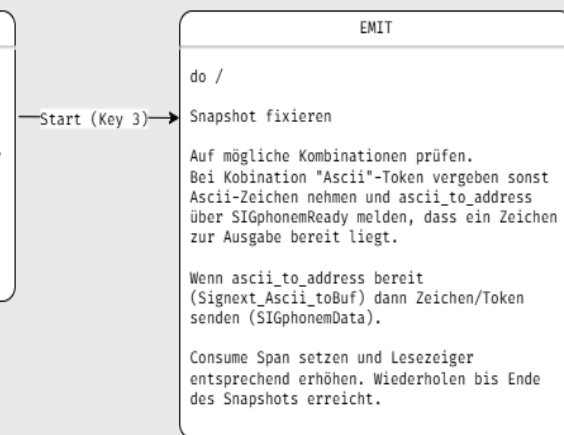
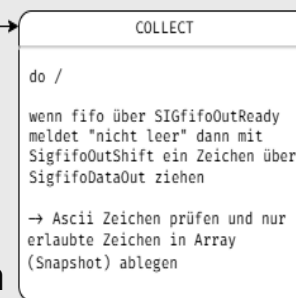


- FIFO-Speicher
- entkoppelt Morsegeschwindigkeit von Geschwindigkeit der Sprachsynthese
- Platz für 128 ASCII-Zeichen

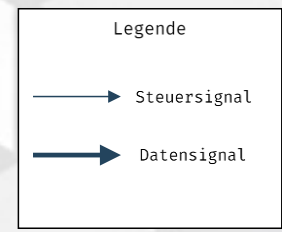
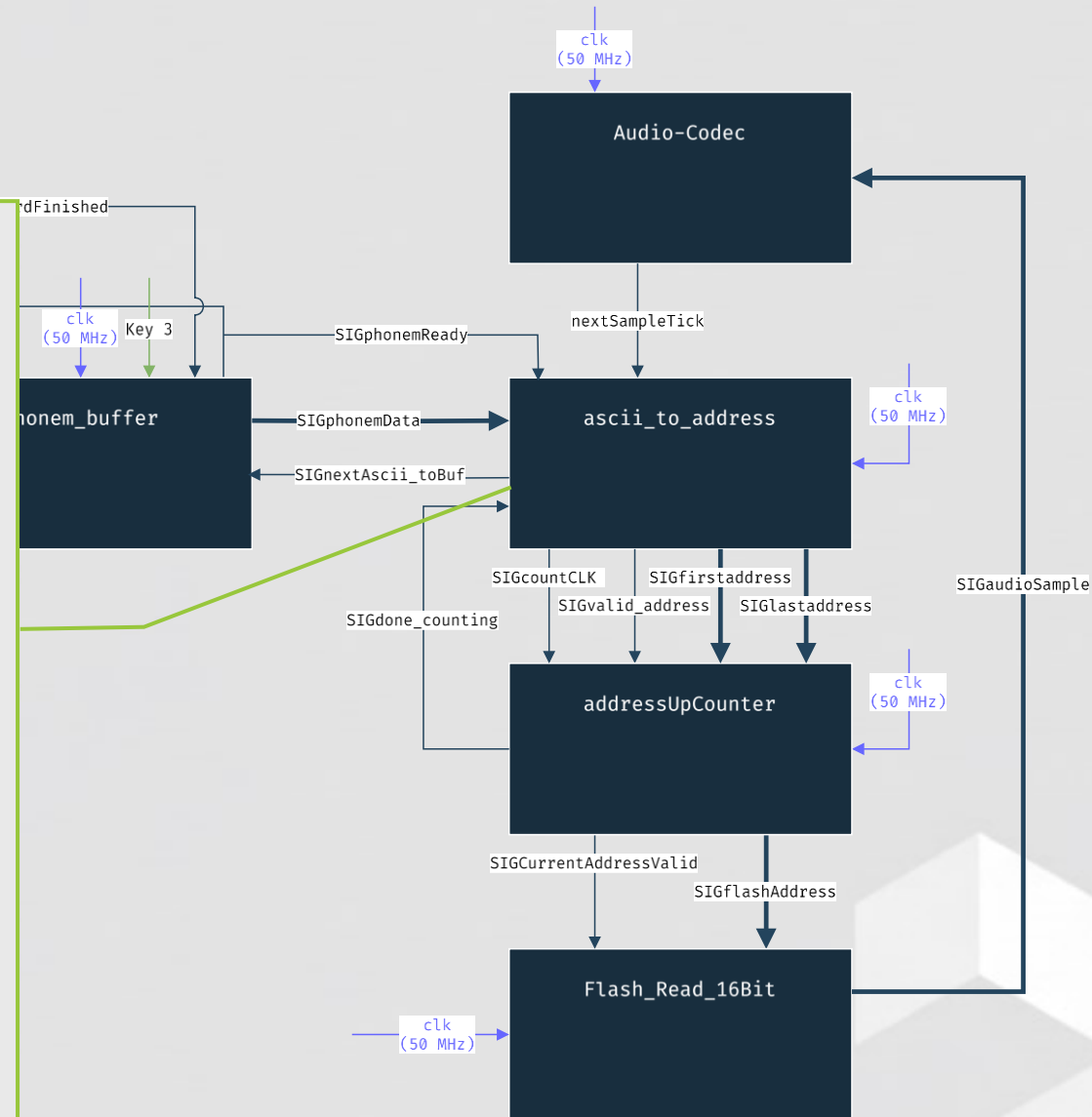
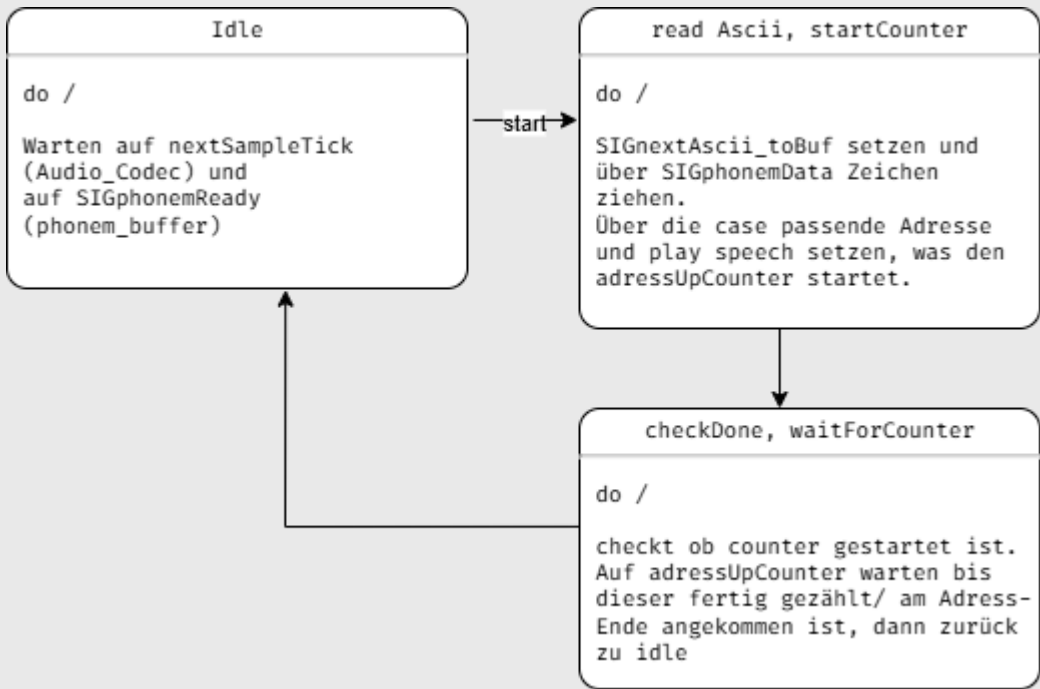




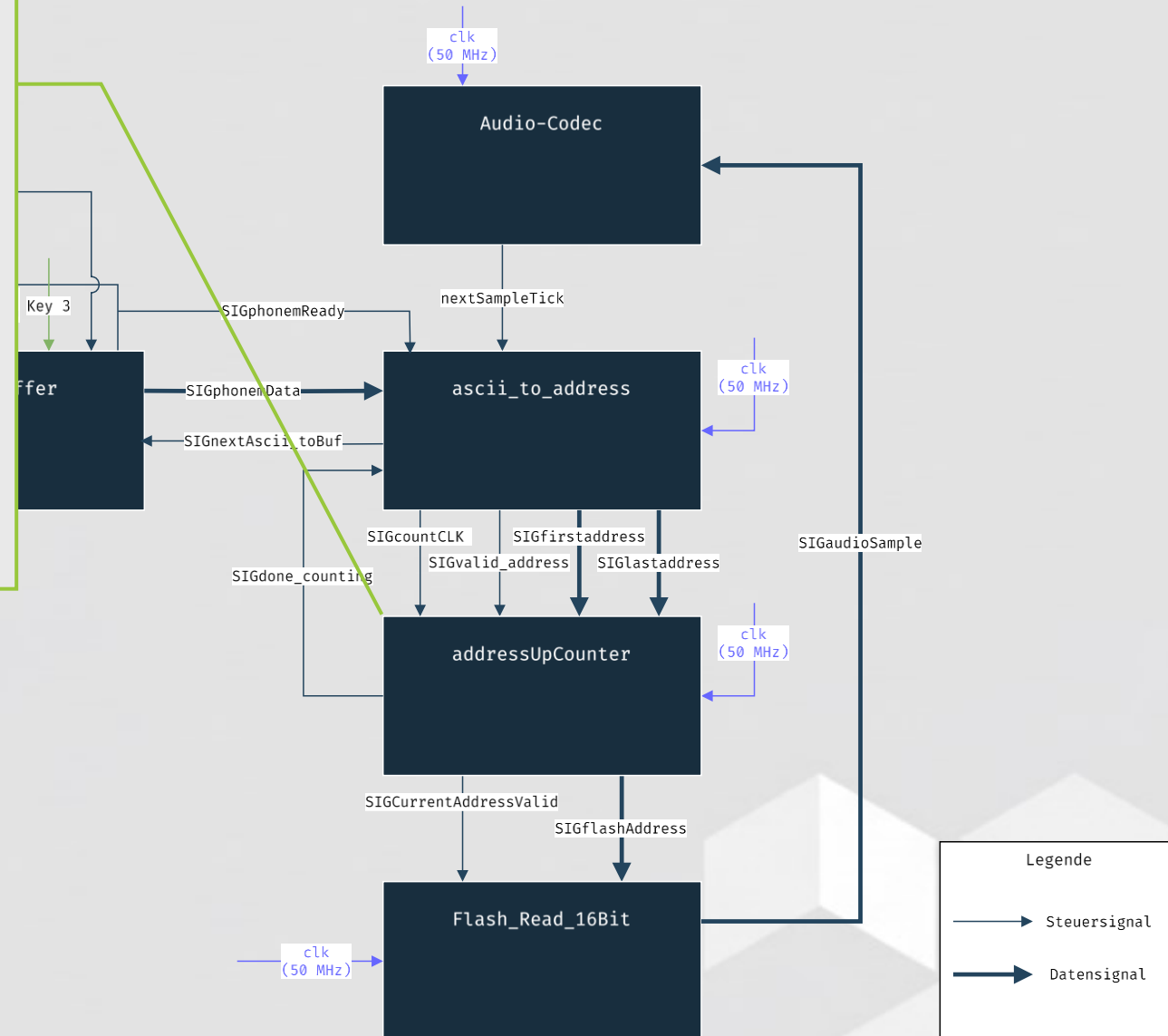
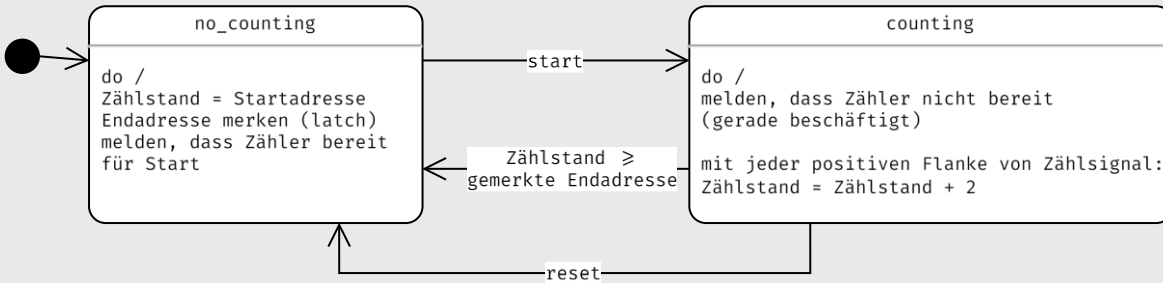
- Das Modul liest fortlaufend ASCII-Bytes aus der fifo in einen Puffer
- Bei Tastendruck werden Phonem-Kombinationen erkannt und „Token“ vergeben
- Gibt die „Token“ oder einzelne Ascii-Bytes nacheinander an ascii\_to\_address weiter



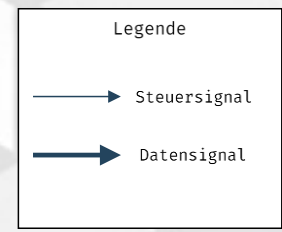
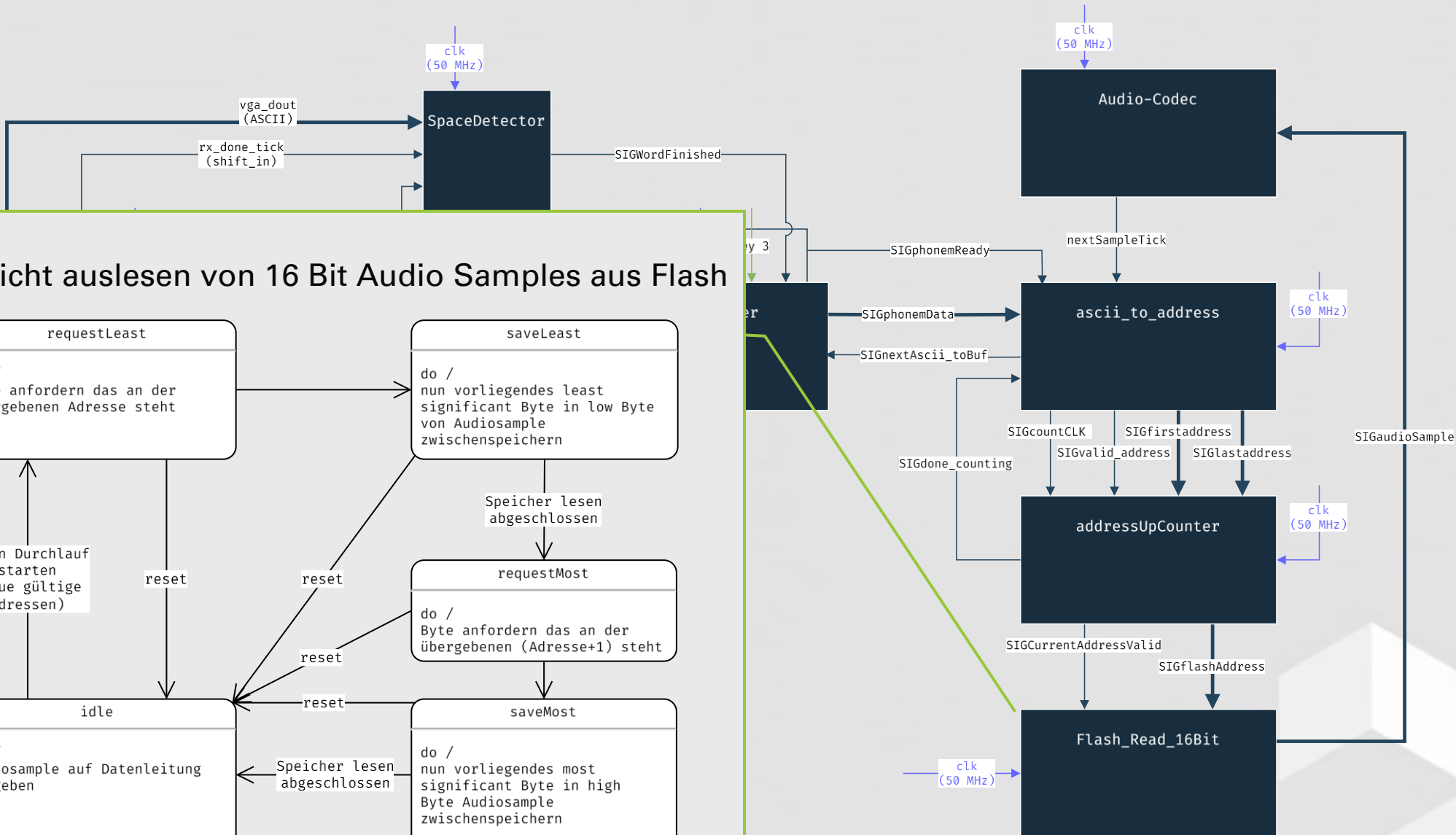
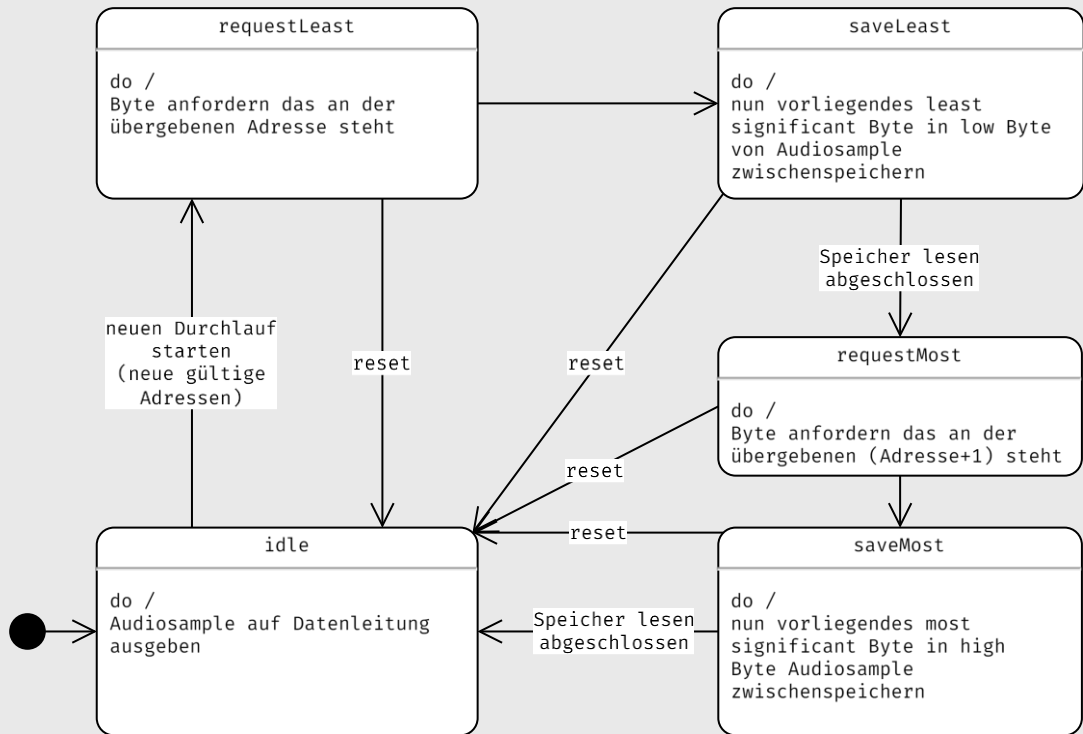
- Das Modul nimmt ein Zeichen entgegen und weist ihm die passende Adresse im Flash-Speicher zu und startet den adressUpCounter



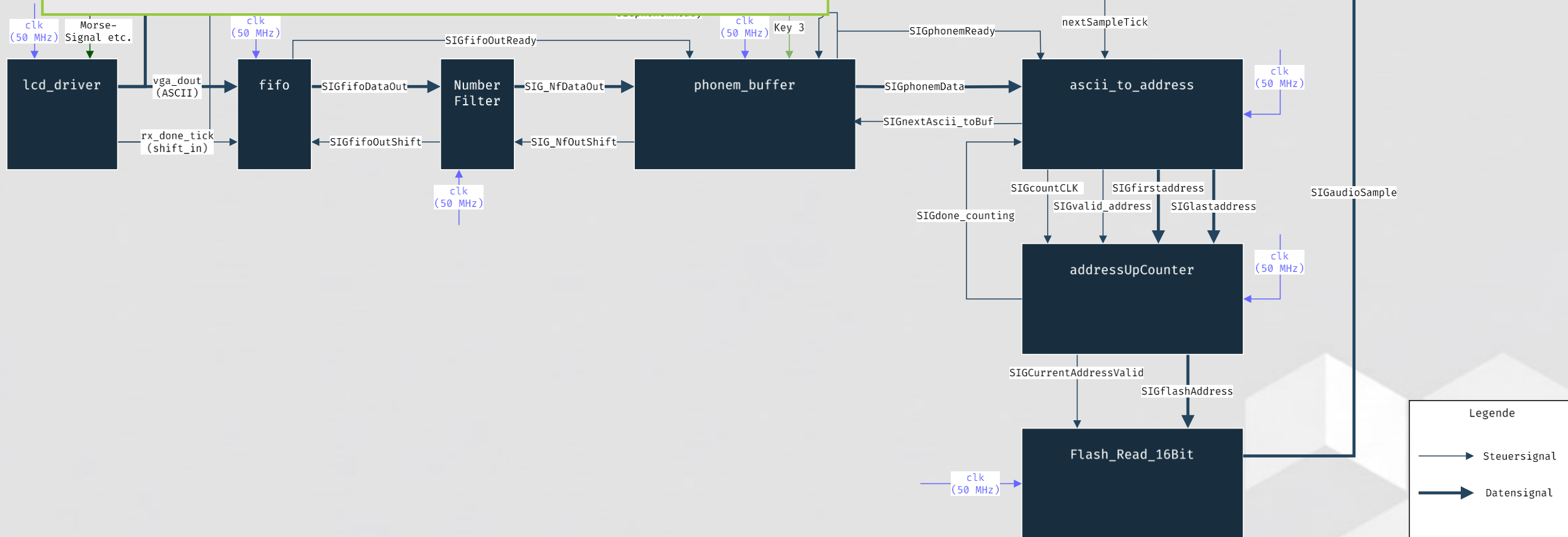
- das Modul zählt von der ersten Adresse (23-Bit-Zahl) bis zur letzten Adresse (23-Bit-Zahl) des jeweiligen Phonems
- die momentane Adresse (Zählwert) wird ausgegeben

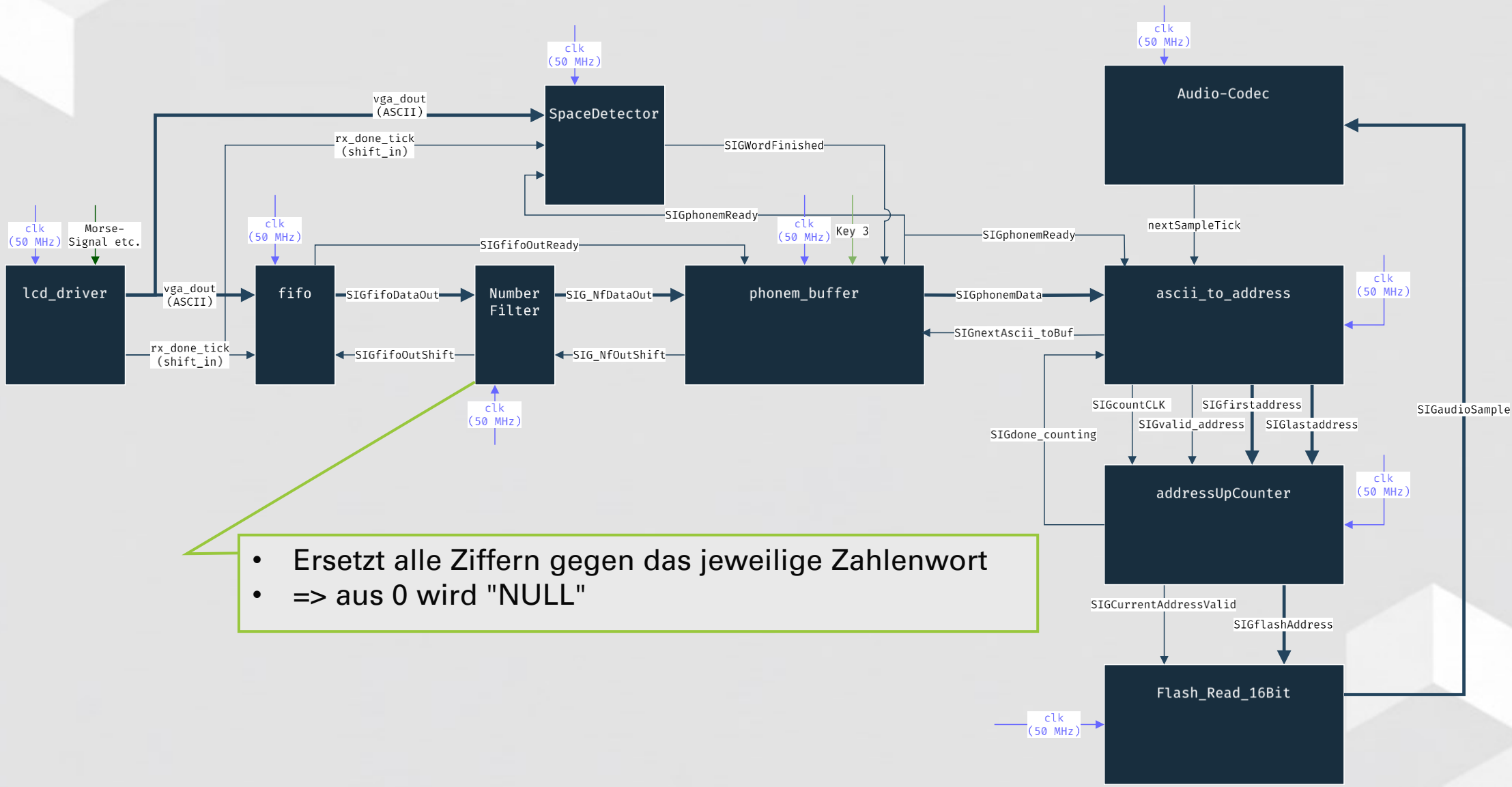


- ermöglicht auslesen von 16 Bit Audio Samples aus Flash

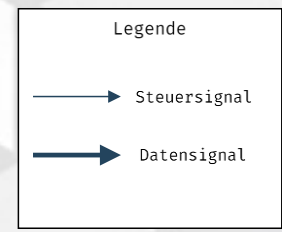


- spielt Morseton oder Sprache ab
- fordert neue Audiodaten mit 32 kHz an
- FPGA-Hardware arbeitet mit 50 MHz
- $\Rightarrow \frac{50 \cdot 10^6 \text{ Hz}}{32 \cdot 10^3 \text{ Hz}} \approx 1562$  Takte um nächstes Sample zu schicken

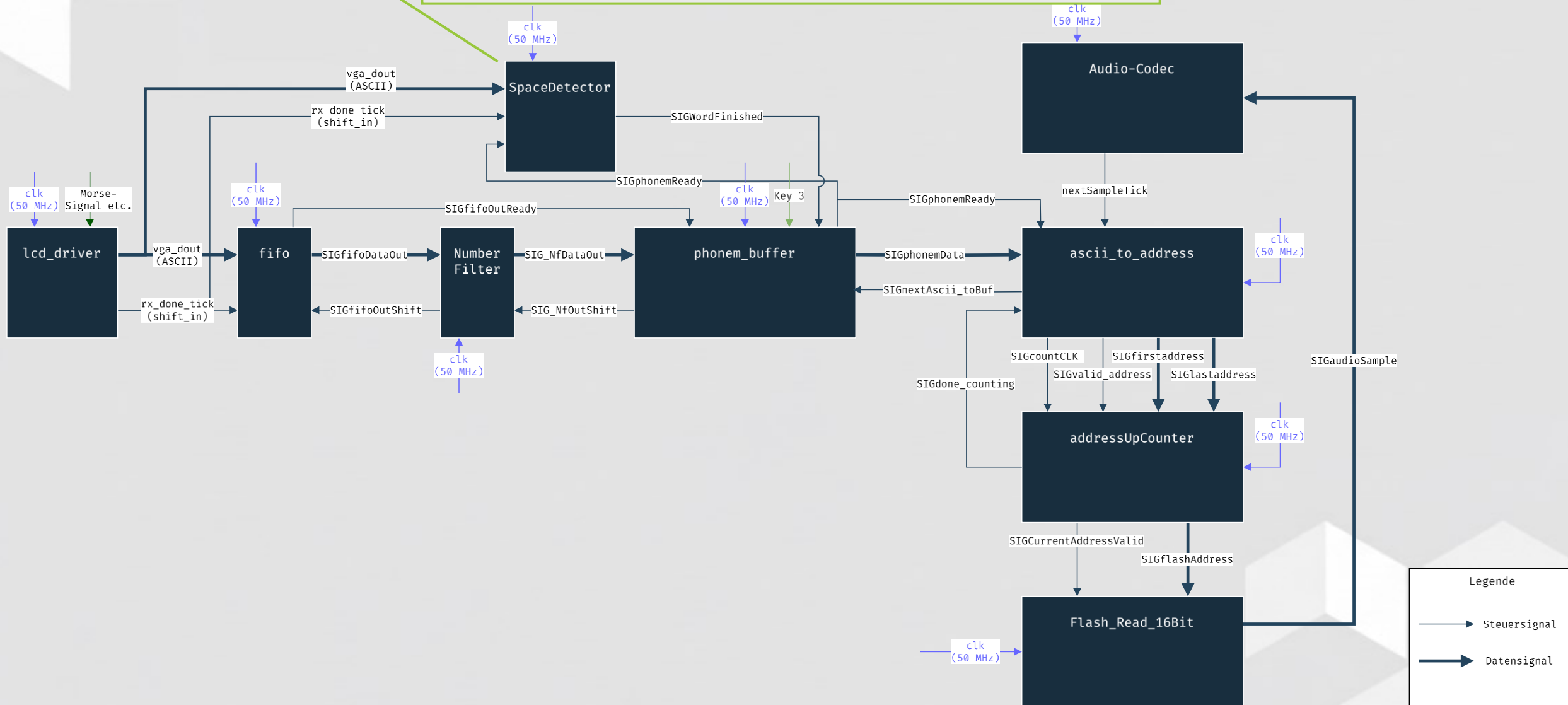




• Ersetzt alle Ziffern gegen das jeweilige Zahlenwort  
• => aus 0 wird "NULL"



- sucht in ASCII-Daten des lcd\_drivers nach einem Leerzeichen und löst die Sprachausgabe aus, wenn der phonem\_buffer dafür bereit ist (führt gerade keine Synthese aus)



# Ergebnis

Vorführung

# Quellen

[1] R. Wiese, *Phonetik und Phonologie*, 1. Auflage. in UTB Sprachwissenschaft, no. 3354. Paderborn: Wilhelm Fink, 2011.

[2] „web\_Sprachkurven\_Lautheit-Laute.jpg (900×596)“. Zugegriffen: 9. September 2025. [Online]. Verfügbar unter: [https://raumakustik-premium.de/wp-content/uploads/2023/06/web\\_Sprachkurven\\_Lautheit-Laute.jpg](https://raumakustik-premium.de/wp-content/uploads/2023/06/web_Sprachkurven_Lautheit-Laute.jpg)