**Systembeschreibung: Mikrocontroller- und Python-Kommunikation zur Signalverarbeitung und Visualisierung**

Diese Dokumentation liefert eine umfassende und technisch fundierte Darstellung der Architektur, Funktionsweise und Kommunikation eines eingebetteten Gesamtsystems zur Erkennung, Analyse und Visualisierung von Stromsignalen in einem elektrischen Schaltkontakt. Die Systemarchitektur besteht aus zwei eng verzahnten Komponenten:

* **MCU-seitige Firmware (STM32G4-Serie)** zur präzisen Signalerzeugung, zur Steuerung von Zustandsübergängen und zur Kommunikation mit externen Geräten
* **Python-basierte PC-Software** zur Echtzeitdatenverarbeitung, grafischen Darstellung und Benutzerinteraktion über serielle Schnittstellen

Dabei ist zu beachten, dass zwei verschiedene Mikrocontroller zum Einsatz kommen:

* **MCU 1 (Signalgenerator)** erzeugt ein digitales GPIO-Signal (simuliertes LED-Signal), basierend auf einer Zeitfolge, das über Pin **PC13** ausgegeben wird. Dieses simulierte Signal stellt eine künstliche LED-Pulssequenz dar.
* **MCU 2 (Analyse-MCU, STM32G474)** empfängt dieses Signal über **PA1 (GPIO-Eingang)** und misst gleichzeitig über **PA0 (ADC-Eingang)** die Stromkurve eines externen Systems.

**1. Mikrocontroller-Firmware (STM32G474CEUx, Analyse-MCU)**

**Hardwareaufbau und verwendete Peripherie:**

* **GPIO PA0**: Eingang des ADCs, über den kontinuierlich der Stromverlauf gemessen wird.
* **GPIO PA1**: GPIO-Eingang für das LED-Signal, das vom Signalgenerator (MCU 1) über PC13 eingespeist wird.
* **GPIO PA5**: (Nur bei MCU 1) Dient als digitaler Ausgang zur Erzeugung von simulierten LED-Signalen.
* **GPIO PC13**: (Nur bei MCU 1) Ausgabe des LED-Signals, das zur Analyse-MCU (PA1) geführt wird.
* **TIM2**: Timer mit einer Auflösung von 1 µs. Er wird zur hochgenauen Zeitsteuerung der Signalumschaltungen eingesetzt und arbeitet interruptgesteuert.
* **USART1 (UART)**: Serielle Schnittstelle zur Kommunikation mit einem externen PC oder Embedded Host. Wird zur Übertragung von Datenpaketen, Messwerten und Kommandos verwendet.
* **Ringpuffer (ringbuffer.c)**: Pufferstruktur zur asynchronen, interruptunabhängigen Zwischenspeicherung von Messdaten vor der seriellen Übertragung.

**Ablauf und Funktionslogik:**

1. **Systeminitialisierung**: Beim Systemstart erfolgt die Konfiguration aller relevanten Peripherie-Komponenten (GPIOs, Timer, UART, ADC). Die MCU verbleibt danach im Bereitschaftszustand.
2. **Hauptprogramm (Main Loop)**: Die Firmware wartet auf ein externes Ereignis wie den Start des LED-Signals an PA1 oder ein UART-Kommando.
3. **Interruptverarbeitung:**
   * **EXTI Interrupt (PA1)**: Bei Flankenwechsel wird der Start oder das Ende eines LED-Signals erkannt, was als Trigger für die Stromanalyse dient.
   * **TIM2 Interrupt**: Dient ggf. zur hochfrequenten Auswertung von Zuständen und ADC-Werten.
4. **UART-Empfang und Kommandoverarbeitung:**
   * Die Firmware reagiert auf Kommandos wie "START", "STOP", "RESET" oder Konfigurationsdaten zur Betriebsart. Diese werden im Hintergrund empfangen und ausgewertet.

**Buffering-Struktur im Detail:**

* **ADC-Puffer**: Der ADC wird (z. B. über DMA) mit einer bestimmten Abtastrate betrieben. Die Resultate werden in einem internen ADC-Puffer abgelegt. Bei Bedarf können sie über Interrupt oder Polling ausgelesen und weiterverarbeitet werden.
* **GPIO-Ereignispuffer**: Zustandswechsel auf PA1 (vom LED-Signal) werden über Interrupts erfasst. In einigen Modi erfolgt eine Zeitstempelung zur Bestimmung der Pulsdauer.
* **Ringbuffer (für UART-Ausgabe)**: Alle gesampelten oder berechneten Daten werden zuerst in einen zyklischen Ringpuffer geschrieben. Der UART-Transmitter greift bei Sende-Bereitschaft auf diesen Puffer zu. So wird gewährleistet, dass auch bei kurzzeitig hohen Datenraten keine Informationen verloren gehen.

**2. Python-Software zur Signalvisualisierung und Analyse**

**Modulübersicht und Aufgaben:**

* **main.py**: Einstiegspunkt des Programms. Verknüpft GUI-Logik, Datenkommunikation und Visualisierung.
* **serial\_reader.py**: Führt den kontinuierlichen Empfang von UART-Daten in einem separaten Thread durch. Erkennt Frame-Grenzen und leitet Nutzdaten weiter.
* **frame.py**: Implementiert das Parsing der empfangenen Datenframes. Zerlegt die Datenstruktur in Header, Payload und interpretiert die Inhalte (z. B. ADC-Werte, Zeitstempel).
* **draw\_signal.py / plot\_adc\_data.py**: Realisieren die visuelle Darstellung der ADC- bzw. Stromdaten in Form eines Liniendiagramms. Unterstützt werden auch zeitliche Marker.
* **draw\_adc\_signal\_from\_hex.py**: Offline-Tool zur Analyse zuvor gespeicherter Rohdaten im HEX-Format. Ideal zur Nachbearbeitung und Auswertung von Versuchen.
* **requirements.txt**: Listet die erforderlichen Python-Bibliotheken auf, u. a. pyserial, numpy, matplotlib, für Installation und Reproduzierbarkeit.

**Ablauf der Python-Anwendung:**

1. **Verbindungsaufbau und Initialisierung**: Serielle Schnittstelle (z. B. COM-Port) wird geöffnet, Parameter wie Baudrate (115200) gesetzt. Empfangstask wird gestartet.
2. **Datenempfang und Dekodierung:**
   * Der Empfangs-Thread liest UART-Daten byteweise ein.
   * Ein Puffer sammelt empfangene Bytes, bis ein vollständiger Frame erkannt wird.
3. **Signaldarstellung und Datenverarbeitung:**
   * Dekodierte Werte (z. B. Strom, Spannung, Zeitstempel) werden in Echtzeit visualisiert.
   * Wahlweise werden Marker für Ereignisse eingeblendet (z. B. Trigger, Lichtbogenstart).
4. **Offline-Analyse und Export:**
   * Vorab gespeicherte Daten aus adc\_hex\_data.txt können eingelesen und visualisiert werden.
   * Exportoptionen für CSV-Logs, PNG-Plots oder interaktive Diagramme bestehen.

**Erweiterungs- und Integrationsmöglichkeiten:**

* Optionale GUI-Erweiterung mit Tkinter, PyQt oder Dash
* Unterstützung für weitere Datenquellen (z. B. SPI, CAN, Modbus)
* Integration von Analysefunktionen wie FFT, RMS-Berechnung, Impulslängenstatistik

**3. Kommunikationsprotokoll zwischen MCU und PC (UART)**

Das Kommunikationsprotokoll zwischen der Analyse-MCU (STM32G474) und dem PC basiert auf einer seriellen UART-Verbindung. Diese dient dem zuverlässigen, zeitlich geordneten Austausch von Messdaten, Steuerbefehlen und Systemstatus.

**1. Protokollübersicht**

Die Datenkommunikation erfolgt über UART mit fester Konfiguration. Jede Nachricht (Frame) hat eine klar definierte Struktur und beginnt sowie endet mit festen Steuerbytes, um die Integrität der Kommunikation zu sichern.

**Protokollparameter:**

* Baudrate: **115200 Baud**
* Datenformat: **8 Datenbits**, **gerade Parität (Even)**, **1 Stopbit** (8E1)
* Byteweise Übertragung, Paritätshandling durch UART-Hardware

**2. Paket- bzw. Frame-Struktur**

Jedes Datenpaket (Frame) besteht aus den folgenden Feldern:

|  |  |  |
| --- | --- | --- |
| **Feld** | **Länge (Bytes)** | **Beschreibung** |
| Start-Byte | 1 | Kennzeichnet den Paketbeginn (0xAA) |
| Paket-ID | 1 | Identifiziert den Typ des Pakets |
| Daten | n | Inhalt, abhängig vom Pakettyp |
| Checksumme | 1 | Fehlererkennung (optional) |
| End-Byte | 1 | Kennzeichnet das Paketende (0x55) |

**Beispielpaket:**

0xAA 0xA0 0x00 0xA0 0x55

* 0xAA → Start
* 0xA0 → Paket-ID: Anfragepaket (Typ A0)
* 0x00 → Daten (leer oder Parameter)
* 0xA0 → Checksumme (einfaches Beispiel: ID + Daten)
* 0x55 → Ende

**3. Pakettypen und deren Inhalte**

|  |  |  |
| --- | --- | --- |
| Paket-ID | Inhalt/Bedeutung | Nutzdaten (Beispiel) |
| A0 | ADC-Daten | 2 oder mehr Bytes (z. B. ADC1-Wert, ADC2-Wert) |
| B0 | GPIO-Daten | 1 Byte Bitmaske für mehrere GPIO-Zustände |
| C0 | Zeitstempel | 4 Bytes (z. B. 32-Bit-Zähler in Mikrosekunden) |
| D0 | Dummy/Test/KeepAlive | 0–1 Byte Dummy-Daten |

|  |  |  |
| --- | --- | --- |
| Paket-ID | Typ | Beschreibung |
| A0 | Anfrage | Anfrage vom PC an die MCU (z. B. „Sende ADC-Wert“) |
| B0 | Antwort | Antwort mit Daten von der MCU (z. B. Messwert) |
| C0 | Steuerkommando | Befehl vom PC an die MCU (z. B. „Start“, „Stop“) |
| D0 | Statusmeldung | Status-/Fehlermeldung der MCU |

**4. Beispielhafte Paketstruktur und Inhalte**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start | Paket-ID | Nutzdaten | Checksumme | Ende |
| 0xAA | 0xA0 | 0x03 0xE8 | 0x8B | 0x55 | // ADC-Daten: 1000 |
| 0xAA | 0xB0 | 0x1F | 0xCF | 0x55 | // GPIO-Daten: Bitmaske 0x1F |
| 0xAA | 0xC0 | 0x00 0x00  0x01 0x5E | 0x9F | 0x55 | // Zeitstempel: 0x0000015E |
| 0xAA | 0xD0 | 0x00 | 0xD0 | 0x55 | // Dummy-Paket |

**Beschreibung der Pakete:**

* **A0 (ADC-Daten):** Enthält die aktuellen oder letzten ADC-Messwerte der MCU. Die Anzahl der Bytes hängt von der Auflösung und Kanalzahl ab.
* **B0 (GPIO-Daten):** Überträgt den Status mehrerer GPIO-Pins als Bitmaske (z. B. Bit 0 = PA1 HIGH/LOW).
* **C0 (Zeitstempel):** Liefert einen 32-Bit-Zeitwert in Mikrosekunden zur Synchronisierung der Messwerte.
* **D0 (Dummy):** Kann regelmäßig gesendet werden, um Verbindungsstatus oder Empfangsbereitschaft zu überprüfen.

**Anfrage (A0) vom PC an die MCU:**

0xAA 0xA0 0x00 0xA0 0x55

**Antwort (B0) der MCU mit ADC-Wert = 1000 (0x03E8):**

0xAA 0xB0 0x03 0xE8 0xB3 0x55

**5. Steuerkommandos (C0)**

Über spezielle C0-Pakete können verschiedene Aktionen ausgelöst werden:

|  |  |  |
| --- | --- | --- |
| Code | Funktion | Beispielpaket |
| 0x01 | Start der Messung | 0xAA 0xC0 0x01 0xC1 0x55 |
| 0x02 | Stop der Messung | 0xAA 0xC0 0x02 0xC2 0x55 |
| 0x03 | Einzelmessung (Trigger) | 0xAA 0xC0 0x03 0xC3 0x55 |
| 0x04 | Kontinuierlicher Modus | 0xAA 0xC0 0x04 0xC4 0x55 |
| 0x05 | Pause | 0xAA 0xC0 0x05 0xC5 0x55 |
| 0x06 | Fortsetzen nach Pause | 0xAA 0xC0 0x06 0xC6 0x55 |
| 0x07 | Parameter setzen (z. B. Rate) | 0xAA 0xC0 0x07 0x64 0xCB 0x55 (z. B. 100 Hz) |

Die Checksumme ist projektabhängig und kann als einfache XOR- oder Summenfunktion aus ID + Daten berechnet werden.

**6. Empfangslogik in Python**

* serial\_reader.py liest kontinuierlich UART-Daten.
* Beim Erkennen der Startsequenz (0xAA) wird ein neuer Frame geöffnet.
* Nach Empfang der vollständigen Länge + End-Byte wird das Paket an frame.py zur Interpretation übergeben.
* Plausibilitätsprüfung über Paketstruktur und Checksumme (sofern verwendet).

**7. Synchronisierung und Zeitstempel**

Die Datenpakete des Typen B0 enthalten oft zusätzlich:

* 2 × 16-Bit-Werte (z. B. ADC1/ADC2)
* 1 × 32-Bit-Timestamp (Mikrosekunden) → Dadurch präzise Analyse und Darstellung der Kurven im Zeitverlauf.