Hier ist die Dokumentation für das angegebene C++-Programm:

**Programmbeschreibung**

Dieses C++-Programm steuert ein System, das auf drei Betriebsmodi ausgelegt ist: **Semi-Automatik**, **Automatik** und **Manuell**. Es beinhaltet Fehlerbehandlung, Kommunikationsschnittstellen und LEDs zur Anzeige des Betriebsstatus. Die Hardware wird über I2C und GPIO angesteuert, während verschiedene Zustände und Befehle verarbeitet werden.

**Dateistruktur und Imports**

Das Programm verwendet die folgenden benutzerdefinierten Header-Dateien:

* **DevOptions.h**: Konfigurationseinstellungen.
* **PinMap.h**: Zuweisung der Hardware-Pins.
* **I2C.h**: Verwaltung der I2C-Kommunikation.
* **Memory.h**: Speicherzugriff und Datenpersistenz.
* **Logger.h**: Protokollierung von Nachrichten.
* **LED.h**: Steuerung der LEDs.
* **CAN.h**: (nicht genutzt in diesem Codeabschnitt, möglicherweise für CAN-Kommunikation).
* **Controller.h**: Steuerung von Motoren.

**Hauptkonstanten und Variablen**

**Konstanten**

* **dt\_loop (2 ms)**: Zeitintervall für die Hauptschleife.

**Globale Variablen**

* **old\_loop\_time**: Verfolgt die vergangene Zeit zwischen Schleifendurchläufen.
* **curr\_state**: Gibt den aktuellen Betriebsmodus an.
  + 1: Semi-Automatik
  + 2: Automatik
  + 3: Manuell
  + 4: Fehler
* **err\_shout**: Kontrolliert die Fehlerprotokollierung.
* **Controller-Objekte**: controller\_l (links) und controller\_r (rechts).
* **LED-Objekte**: led\_red (rot) und led\_green (grün).

**Programmfluss**

**1. setup()**

Initialisierung der Hardware und der Controller:

1. Initialisierung der Pin-Zuweisung (via PinMap).
2. Initialisierung des Loggers und der I2C-Kommunikation.
3. Erstellung der Controller-Objekte mit ihren zugehörigen Pins und initialen Einstellungen.
4. Initialisierung der LED-Objekte.
5. Protokollierung der Startmeldung.

**2. loop()**

Wird in regelmäßigen Intervallen (dt\_loop) ausgeführt und enthält die Hauptlogik:

* **I2C-Register-Updates**: Synchronisation von Status und Motorpositionen mit den I2C-Registern.
* **Moduswechsel**: Verarbeitung des aktuellen Modus und Zustandsänderungen basierend auf Eingaben.
* **Fehlerbehandlung**: Automatische Fehlerkorrektur und Protokollierung.

**Betriebsmodi**

1. **Semi-Automatik (State 1):**
   * Motorsteuerung basierend auf semiautomatischen Regeln.
   * Aktivierung der grünen LED.
   * Umschaltung in andere Modi basierend auf Schalterstellungen.
2. **Automatik (State 2):**
   * Motorsteuerung basierend auf Referenzwerten aus dem I2C-Register.
   * Verbindungskontrolle über das Heartbeat-Register.
   * LED-Anzeigen anpassen und Umschaltung in andere Modi.
3. **Manuell (State 3):**
   * Motorstillstand (Halt).
   * Möglichkeit, virtuelle Endanschläge zu speichern.
   * LED-Anzeigen anpassen und Umschaltung in andere Modi.
4. **Fehler (State 4):**
   * Motorbewegung zur Fehlerkorrektur (Home).
   * Protokollierung von Fehlern und Aktivierung der roten LED.
   * Fehlerzustand bleibt bestehen, bis er manuell zurückgesetzt wird.

**3. update\_state(short state)**

Wechselt den aktuellen Zustand des Systems:

1. Beendet aktuelle Motoraktivitäten.
2. Setzt die I2C-Referenzregister zurück.
3. Protokolliert den Wechsel in den neuen Zustand.

**4. update\_status()**

Aktualisiert den Status des Systems und verarbeitet Befehle aus dem I2C-Befehlsregister:

* Speichert neue virtuelle Endanschläge.
* Setzt Fehler zurück.
* Veranlasst einen Übergang aus dem Fehlerzustand.
* Aktualisiert Fehler-Flags und Zustandsbits im Statusregister.

**5. I2C-Ereignisfunktionen**

* **receiveEvent(int howMany)**: Empfängt Daten über I2C.
* **requestEvent()**: Beantwortet I2C-Anfragen.

**Funktionen der Klassen**

1. **Controller**: Verarbeitet Motorsteuerung (Halt, Semi-Automatik, Automatik, Fehlerkorrektur).
2. **LED**: Steuert die LEDs (z. B. Blinken, Ein-/Ausschalten).
3. **Memory**: Speichert und liest die virtuellen Endanschläge.
4. **Logger**: Protokolliert Ereignisse und Statusänderungen.
5. **I2C**: Schnittstelle zur Kommunikation und Registerverwaltung.

**Fehlerbehandlung**

* Überwacht Watchdog- und Timeout-Fehler.
* Protokolliert spezifische Fehler basierend auf Status-Bits.
* Verhindert wiederholte Fehlermeldungen (err\_shout).

**Zusammenfassung**

Das Programm ist ein robustes Steuerungssystem, das mehrere Modi unterstützt, eine präzise Motorsteuerung ermöglicht und eine zuverlässige Fehlerbehandlung bietet. Es ist modular aufgebaut, was die Erweiterbarkeit erleichtert.