Systemnahe Programmierung 2

DHBW Stuttgart, 2024
Prof. Dr.-Ing. Alfred Strey

Benotete Programmieraufgabe

In dieser Aufgabe soll mit dem Mikrocontroller ESP32-C3 und einem Ultraschallsensor ein digitales Entfernungsmessgerät realisiert werden. Zur Anzeige der Entfernung zum Objekt soll hier ein Streifen aus 16 Neopixel LEDs eingesetzt werden. Unterschreitet die Entfernung zum Objekt 50 cm, so soll die erste LED leuchten. Bei einer Distanz von weniger als 5 cm sollen alle 16 LEDs leuchten.

Hardware:

Der vorliegende Streifen ist aus 16 Neopixel LEDs vom Typ SK6812 aufgebaut. Hierbei handelt es sich um Multi-Color LEDs (RGB), die über ein serielles Signal mit speziellem Timing angesteuert werden müssen. Das Datenblatt finden Sie auf der Moodle-Seite des Labors. Für jede LED ist ein 24-Bit Wert zur Kodierung der Farben Grün, Rot und Blau mit jeweils 8 Bit erforderlich, der die Intensität der einzelnen Farben angibt:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Diese 24-Bit Muster müssen zunächst für die 1. LED, dann für die 2. LED, usw. seriell zur LED-Kette übertragen werden. Der Controller der 1. LED speichert die ersten 24 Bit und reicht die folgenden Bits an den Controller der 2. LED weiter, der dann die nächsten 24 Bit abspeichert und wiederum die restlichen Bits weiterleitet. Nach einer Pause von mindestens 50 µs werden die seriell übertragenen Daten von den LEDs übernommen.

Das Datenblatt des Ultraschallsensors finden Sie auf der Moodle-Seite des Labors. Es können Entfernungen zwischen 2 cm und ca. 300 cm gemessen werden. Der Ultraschallsensor benötigt eine Spannung von `VCC` = 5V; als Triggersignal für die Leitung `TRIG` genügt ein Impuls der Höhe von 3,3V**.** Das Echo des Ultraschallsensors an der Leitung `ECHO` hat eine Spannung von 5V ist **unbedingt über einen Spannungsteiler auf 3.3V** zu begrenzen und an einen GPIO Pin anzuschließen.

Software:

Auch wenn hier ein Streifen aus 16 LEDs vorliegt, ist die Software allgemein zu halten, so dass sie für eine variable Anzahl von LEDs funktioniert. Zur Speicherung der aktuellen Anzeige kann hierzu im SRAM des ESP32-C3 z.B. ein Feld

```
#define LEDS 16
uint8_t pixels[LEDS*3]; // Komponenten für Grün, Rot und Blau je LED
```

angelegt werden.

1) Schreiben Sie eine Routine `display()`, die den aktuellen Inhalt des Buffers `pixels` seriell an die LED-Kette ausgibt. Für die Implementierung des erforderlichen Timings der Ansteuer-Signale ist gemäß der Aufgabenstellung für Ihr Team eine der folgenden Methoden zu wählen:

– **Bit-Banging**: Es kann durch explizites Setzen und Löschen eines einzelnen Bits eins GPIO Pins das serielle Signal in der benötigten Form erzeugt werden, wobei das gewünschte Zeitverhalten durch entsprechende Verzögerungen im Programm erreicht wird.

  Hinweis: Hierzu ist es sinnvoll, die Programmierung möglichst direkt in (Inline-)Assembler durchzuführen, da es mit reinem C-Code und dem C-Compiler problematisch ist, die benötigten Verzögerungen genau zu realisieren. Im Assembler können Verzögerungen einfach mit `nop` Instruktionen oder Schleifen aus `nop` Instruktionen realisiert werden.

– **PWM**: Prinzipiell stellt das benötigte Ausgabesignal je Bit ein PWM-Signal dar, das mit Hilfe der PWM-Einheit des ESP32-C3 erzeugt werden kann. Hier ist zur Erzeugung der Signalformen für den Bitwert 0 und den Bitwert 1 das Tastverhältnis entsprechend zu variieren.

  Hinweise:

  1) Die PWM-Unit darf hier kein kontinuierliches Signal erzeugen, sondern immer nur eine Periode ausgeben, d.h. das Bit `LEDC_DUTY_START_CH0` in Register `LEDC_CH0_CONF1_REG` ist zu löschen.

  2) Das Tastverhältnis darf erst geändert werden, nachdem eine Periode abgelaufen ist, d.h. der *overflow*-Wert erreicht ist. Dies kann durch dadurch erreicht werden, dass der *Overflow*-Interrupt aktiviert wird und das entsprechende Interrupt-Bit im Register `LEDC_INT_RAW_REG` abgefragt wird. Das Interrupt-Bit muss zu Beginn einer jeden Periode zurückgesetzt werden.

  3) Es ist empfehlenswert, vor der Ausgabe eines Musters an den LED-Streifen einen Reset der PWM-Einheit durchzuführen, indem das Reset-Bit in `LEDC_TIMER0_CONF_REG` gesetzt und unmittelbar darauf wieder gelöscht wird.

– **SPI**: Es kann die synchrone serielle SPI-Schnittstelle (GP-SPI2) benutzt werden, um das benötigte serielle Signal zu erzeugen. Hier sind je LED-Ausgabebit 4 oder 8 Bit zu generieren, welche die Signalform für den Bitwert 0 oder 1 repräsentieren. Diese werden zu längeren Datenworten zusammengefasst und über den Ausgang `MOSI` zum LED-Streifen übertragen. Da der Controller in den LEDs sich über die Flanken des Signals synchronisiert, wird der Takt `SCKx` der SPI-Schnittstelle hier nicht benötigt (kann aber zu Debugzwecken auf dem Oszilloskop kontrolliert werden). Das Datenblatt des SPI-Bausteins im ESP32-C3 Mikrocontroller finden Sie in Kap. 27 des *Technical Reference Manuals* des ESP32-C3. Für die Programmierung stehen in der auf der Moodle-Seite bereitgestellten Datei `spi.h` die beiden Funktionen `spi_init()` und `spi_send(char* buffer, int size)` zur Verfügung, wobei die Funktion `spi_send()` eine Anzahl von `size` Bits aus dem Puffer `buffer` sendet. Die in den Funktionen verwendeten Datenstrukturen sind auf folgender Webseite dokumentiert:
  https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32c3/api-reference/peripherals/spi_master.html

– **UART**: Man kann den seriellen Ausgabestrom auch durch Einsatz der UART-Unit erzeugen, indem man eine hohe Taktrate wählt und die Signalform für jedes Bit durch ein geeignetes 8-Bit Paket mit Start- und 1 oder 2 Stop-Bits erzeugt. Das Datenblatt der UART-Einheit im ESP32-C3 Mikrocontroller finden Sie in Kap. 26 des *Technical Reference Manuals* des ESP32-C3. Für die Programmierung steht in der auf der Moodle-Seite bereitgestellten Datei `uart.h` die Funktion `uart_init()` zur Verfügung, die die UART-Einheit initialisiert. Die in der Funktion benutzten Parameter sind geeignet anzupassen und auf folgender Webseite dokumentiert:
  https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32c3/api-reference/peripherals/uart.html

  Hinweise:

  1) Von den beiden UART Controllern `UART_NUM_0` und `UART_NUM_1` wird der 1. Controller bereits für den Betrieb als Monitor eingesetzt; es ist hier daher der 2. Controller `UART_NUM_1` zu verwenden.

  2) Für die korrekte Polarität (0 im Ruhezustand) des Ausgabesignals `TXD` muss bei der Initialisierung zusätzlich die Funktion `uart_set_line_inverse(UART_NUM_1, UART_SIGNAL_TXD_INV)` aufgerufen werden.

  3) Die Ausgabe erfolgt mit der Funktion `uart_write_bytes(UART_NUM_1, char* buf, int len)`, wobei `buf` ein Zeiger auf einen Buffer ist und `len` die Anzahl der auszugebenen Bytes darstellt

Testen Sie vor Anschluss des LED-Streifens Ihre Routine `display()`, indem Sie am Oszilloskop die Korrektheit des Timings überprüfen!

2) Generieren Sie zunächst periodisch auf der Leitung `TRIG` zum Ultraschallsensor einen Impuls der Dauer 10 µs und schauen Sie auf dem Oszilloskop das Echo-Signal an, wenn Sie die Distanz zu einem Objekt verändern.

3) Schreiben Sie eine Funktion `readSensor()`, die mit dem Ultraschallsensor durch Auswertung des Echo-Impulses die Entfernung zum nächsten Objekt misst.

Hinweis: Die Schallgeschwindigkeit bei 20°C beträgt 343,2 m/s.

4) Implementieren Sie in der Routine `loop()` einen C-Code, der die Entfernungsmessung startet und die Entfernung auf dem LED-Display ausgibt.

5) Ein zeitkritische Teil der Routine `display()` soll unter Verwendung der RISC-V Assemblersprache implementiert werden. Hierzu soll der Inline-Assembler, ggf. mit der *Extended Inline Assembler Syntax* verwendet werden.

Minimalanforderung:

Wie oben beschrieben, müssen die serielle Ansteuerung der LEDs, die Entfernungsmessung und die Ausgabe der Entfernung auf dem LED-Streifen programmiert werden. Der Code ist sauber zu strukturieren und gut zu kommentieren. Es darf keine Gleitkomma-Arithmetik verwendet werden. Ein kleiner, nach Möglichkeit zeitkritischer Teil der Implementierung muss in RISC-V Assemblercode erfolgen.

Ideen für Ergänzungen/Optimierungen:

- Die Messwerte des Ultraschallsensors schwanken leicht, vor allem bei größeren Entfernungen. Sie können hier durch eine Mittelwertbildung und Ausschluss fehlerhafter Messungen eine stabilere Anzeige der Entfernung erreichen.
- Überlegen Sie, ob und wie Sie Ihre Lösung durch Verwendung von Interrupts (z.B. Timer) optimieren können.
- Variation der Helligkeit der LEDs durch ein externes Potentiometer.
- Nutzen Sie die Farbe der LEDs, indem Sie bei Verringerung der Distanz einen langsamen Übergang der Farbe von grün über gelb und orange nach rot realisieren.
- Nutzen Sie zusätzlich die mit GPIO Pin 8 verbundene Mehrfarben-LED (ebenfalls vom Typ SK6812) auf dem ESP32-C3 Board, um die Distanz in grün, gelb, orange und rot (bei weniger als 5 cm) anzugeben.
- Programmieren Sie gleitende Übergänge zwischen den LEDs, d.h. bei langsam kleiner werdender Distanz soll eine LED erst glimmen und dann immer heller werden, bevor die nächste LED aktiviert wird.
- Statt der vorgegebenen High-Level Funktionen für SPI bzw. UART kann die SPI-bzw. UART-Einheit auch über die Konfiguration und Nutzung einzelner Register programmiert werden.
- Schließen Sie über einen externen DAC und einen Operationsverstärker einen Lautsprecher an (Material kann bereitgestellt werden) und geben Sie auch eine akustische Warnung aus.
- … (*Eigene Ideen für Ergänzungen bzw. Optimierungen sind sehr willkommen*!)

Anmerkungen:

- **Die benotete Programmieraufgabe stellt eine Prüfungsleistung dar, die eigenständig zu erbringen ist. Jede Übernahme oder Weitergabe von Codefragmenten von/an andere Teilnehmer stellt einen Täuschungsversuch dar!**
- Abzugeben über die Moodle-Seite bis zum Ende des 1. Labortermins am 26.11. sind der aktuelle Stand des Codes; dieser darf bis zum zweiten Termin zwar noch ergänzt oder korrigiert, aber nicht durch anderen Code ersetzt werden. Insbesondere darf die gewählte Schnittstelle (Bit-Banging, PWM, SPI oder UART) nicht mehr geändert werden.
- Abzugeben über die Moodle-Seite bis zum 03.12. um 23:59 Uhr sind der kommentierte Quellcode sowie eine kurze Beschreibung Ihrer Implementierung (ca. 1 bis 2 Seiten, als pdf-Datei); hierin sollen vor allem die Beschaltung der Hardware (z.B. Angabe der verwendeten Pins des ESP32-C3 Boards), ein Foto vom Aufbau (das Ihr Programm in Anwendung zeigt), sämtliche implementierten Ergänzungen und Optimierungen und ggf. der Einsatz von Interrupts dokumentiert werden.