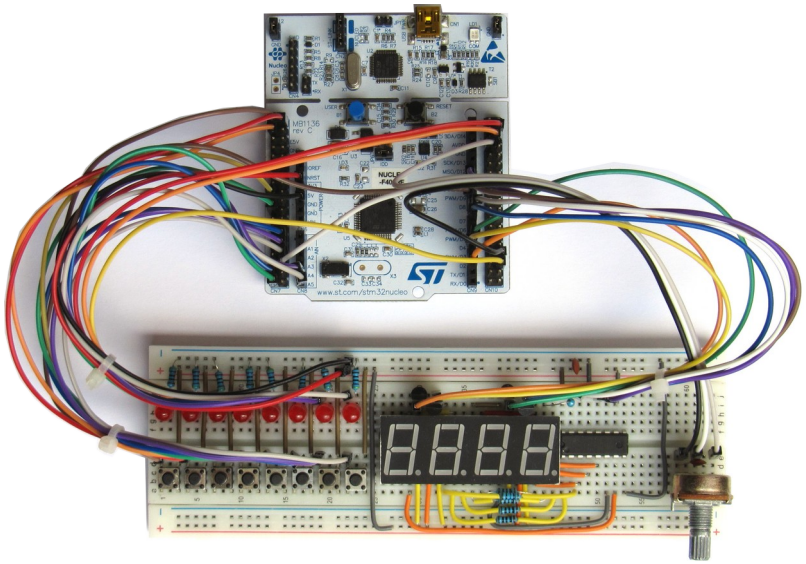
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

Mikroprozessoren-Labor



Versuch 5, Optimierung und RTC

Versuchsvorbereitung und Durchführung

Stand: 25. April 2021

I. Versuchsbeschreibung

In diesem Versuch werden Aufgaben zu Performance-Optimierung und Nutzung der Echtzeituhr (Real Time Clock, RTC) bearbeitet.

Da es Ihr letzter Laborversuch ist, ist die Zeitaufteilung bewusst flexibel gehalten. Die Bearbeitungszeit für die beiden Aufgaben sollte in etwa gleich sein und es sollte möglich sein, eine Aufgabe als Vorbereitung und die andere während des Labors zu bearbeiten.

Die Teilnahme am Labortermin ist auch für diesen Versuch Pflicht. Sie können den Labortermin aber flexibel nutzen und z. B. beide Aufgaben vorbereiten und die Zeit im Labor verwenden, um mit den Betreuern offene Fragen zu alten Laboraufgaben zu klären.

Für diesen Versuch muss kein Laborbericht abgegeben werden. Die Versuchsbeschreibung enthält dennoch, wie bei den Versuchen vorher, Abschnitte mit der Kennzeichnung „Schriftliche Aufgabe“. Diese Abschnitte sollen Sie darauf hinweisen, dass es an dieser Stelle der Versuchsdurchführung sinnvoll ist, ein Ergebnis zu notieren oder über eine Fragestellung nachzudenken.

II. Aufgaben

1 Optimierung der Ansteuerung des Schieberegisters

In der Vorlesung wurden verschiedene Ansätze und Techniken für die Performance-Optimierung vorgestellt. Ziel dieser Aufgabe ist es, die Befüllung des Schieberegisters in der Klasse `CShift-RegisterOutputExtender` soweit möglich zu optimieren.

1.1 Ausgangssituation

Laden Sie das Startprojekt aus Moodle herunter und schreiben Sie eine Funktion `task1`, die die 7-Segment-Anzeige ansteuert. Sie können nach Belieben Code aus einem der vorhergehenden Labore nutzen und „statisch“ z. B. die letzten Stellen Ihrer Matrikelnummer anzeigen oder Sie können den vom ADC gelieferten Wert anzeigen oder Sie können, wenn Sie die Aufgabe 2 zuerst bearbeiten, die Uhrzeit anzeigen. Verwenden Sie die im letzten Labor entwickelte, Interrupt-gesteuerte Version von `CSevenSegmentDisplay`.

ausloten - to get into the bottom of matter

In dieser Aufgabe sollen Sie die Möglichkeiten zur Optimierung des Datentransfers in das Schieberegister ausloten. Wie bei jeder Optimierungsaufgabe brauchen Sie für die Beurteilung eine Messgröße. Definieren Sie ein neues Objekt vom Typ `DigitalOut` für die Ansteuerung eines bislang ungenutzten GPIO-Pins. Setzen Sie mit einer zusätzlichen, ersten Anweisung in der Implementierung Ihrer Methode `CShiftRegisterOutputExtender::prepareOutput` den Ausgang auf logisch 1 und mit einer zusätzlichen, letzte Anweisung in der Methode den Ausgang wieder auf logisch 0.

Schließen Sie den Logic-Analyzer an den gewählten Ausgang an. Messen Sie die Zeit, für die der Ausgang auf High-Pegel liegt. Das ist die Zeit, die für die Ausführung der Methode benötigt wird.

✎ **Schriftliche Aufgabe D-1**: Notieren Sie den Wert als Referenzwert für die Optimierung.

✎ **Schriftliche Aufgabe D-2**: Welche Aussage können Sie zur Messgenauigkeit machen?

1.2 Einschalten der Optimierung im Compiler

Die Optimierungseinstellungen des Compilers haben sowohl Einfluss auf die Ausführungsgeschwindigkeit als auch auf die Größe des generierten Codes. Damit Sie auch dafür einen Referenzwert haben, führen Sie „Clean Project“ und „Build Project“ aus. Am Ende der bei der Übersetzung in der Konsole erzeugten Ausgabe sehen Sie, wie groß Ihr Programm ist:

```
   text        data         bss         dec         hex     filename
  27084         776        1312       29172        71f4     MIT-Labor_5.elf
```

Dabei steht „text“ für die Größe des Programm-Codes (Maschineninstruktionen), „data“ für die Größe des initialisierten Datenbereichs (z. B. Werte von Konstanten) und „bss“ für den für Variab-

len benötigten, nicht initialisierten Schreib-/Lesespeicher[1]. Interessant ist hier vor allem der Wert für „text“.

✎ **Schriftliche Aufgabe D-3**: Notieren Sie die Programmgröße für die Einstellung „keine Optimierung“.

Stellen Sie jetzt in den Projekteinstellungen sowohl für den C als auch für den C++-Compiler die Optimierung „-Ofast“ ein, führen Sie „Clean Project“ und „Build Project“ aus und starten Ihr Programm.

✎ **Schriftliche Aufgabe D-4**: Notieren Sie die Programmgröße. Messen und notieren Sie, welche Zeit der µC jetzt für die Übertragung der Daten in das Schieberegister benötigt.

Wechseln Sie auf die Optimierungseinstellung „-Os“ (führen Sie wieder Clean/Build durch) und starten Ihr Programm erneut.

✎ **Schriftliche Aufgabe D-5**: Notieren Sie die Programmgröße. Messen und notieren Sie, welche Zeit der µC mit „-Os“ für die Übertragung der Daten in das Schieberegister benötigt.

Bleiben Sie unabhängig vom Ergebnis bei dieser Compiler-Einstellung, da sie den am besten verständlichen Assembler-Code liefert. Sie werden ganz am Ende nochmal die Einstellung wechseln.

1.3 Optimieren auf C-/C++-Ebene

1.3.1 Code-Analyse

Schaut man sich die Anweisung für das Setzen oder Zurücksetzen eines Pins mit Hilfe der Klasse `DigitalOut` und dem überladenen Zuweisungsoperator an, kann man zunächst vermuten, dass der Vorgang sehr ineffizient ist. Wenn Sie sich die Definition des Zuweisungsoperators in der Klassendefinition von `DigitalOut` ansehen, werden Sie allerdings feststellen, dass die Methode, die den Operator überlädt, als inline-Methode definiert ist. Kopieren Sie die Zuweisung aus Ihrem Code in einen Texteditor und ersetzen Sie sie durch die Anweisungen, die die Methode implementieren.

Sie sollten jetzt im Wesentlichen den Aufruf einer `write`-Methode sehen. Wenn Sie sich die Definition dieser Methode anschauen, werden Sie wiederum feststellen, dass sie als inline-Methode definiert ist. Ersetzen Sie daher den Aufruf der Methode `write` durch die Anweisungen, die die Methode implementieren. Wiederholen Sie den Vorgang bis alle inline-Funktionsaufrufe „aufgelöst“ sind.

Sie sollten, nachdem Sie alle Ersetzungen korrekt durchgeführt haben, in Ihrem Texteditor nur noch eine `if`-Bedingung mit je einer einzigen Zuweisung im `if`- bzw. `else`-Block stehen haben. Der C++-Code ist also gut strukturiert, aber dennoch sehr effizient.

1.3.2 Spezialregister nutzen

Für eine weitergehende Optimierung ist einer der vielversprechendsten Ansätze die Nutzung des „GPIO port bit set/reset register“ (BSRR). Dafür benötigen Sie den Zugriff auf die Register des

1 Eine sehr schöne ausführliche Erklärung finden Sie z. B. hier: https://mcuoneclipse.com/2013/04/14/text-data-and-bss-code-and-data-size-explained/.

GPIO-Ports, das zu dem als `DigitalOut` übergebenen Ausgang gehört. Die Klasse `DigitalOut` folgt dem Prinzip des Information Hiding, d. h. wenn Sie ein Objekt vom Typ `DigitalOut` haben, können Sie nicht ohne weiteres auf den für die Implementierung verwendeten Zeiger auf die CMSIS `GPIO_TypeDef` zugreifen.

Es gibt einen Trick, für den Sie allerdings nicht über die notwendigen C++-Kenntnisse verfügen. Deshalb enthält das Ausgangsprojekt für das Labor eine Funktion `gpioPortInfo`, die Ihnen für ein Objekt vom Typ `DigitalOut` das verwendete Port (als Zeiger auf `GPIO_TypeDef`) und den verwendeten Pin (als Bit-Maske) liefert. Sie finden die Dokumentation der Funktion in der Datei GpioUtil.h.

Strukturieren Sie die Implementierung der Methode `CShiftRegisterOutputExtender::prepareOutput` jetzt so um, dass der Transfer der Werte in das Schieberegister in einer eigenen Funktion steht (`doShift`).

```
void doShift(GPIO_TypeDef* clockPort, uint32_t clockMask,
          GPIO_TypeDef* dataPort, uint32_t dataMask, uint8_t value) {
     // Schieben Sie den übergebenen Wert durch Ansteuern der BSRR-
     // Register für den Ausgang, der das Taktsignal liefert und den
     // Ausgang, der das Datensignal liefert in einer Schleife (!)
     // in das Schieberegister.

     // ToDo
}

void CShiftRegisterOutputExtender::prepareOutput(uint8_t value) {
     // Messausgang auf logisch 1
     // ...
     // Informationen für den direkten Zugriff auf die GPIO-Register holen.
     GPIO_TypeDef* clockPort;
     uint32_t clockMask;
     gpioPortInfo(*m_shiftRegisterClock, clockPort, clockMask);
     GPIO_TypeDef* dataPort;
     uint32_t dataMask;
     gpioPortInfo(*m_serialData, dataPort, dataMask);

     doShift(clockPort, clockMask, dataPort, dataMask, value);
     m_preparedOutput = value;
     // Messausgang auf logisch 0
     // ...
}
```

Implementieren Sie `doShift` unter Nutzung der BSRR-Funktionalität.

✎ **Schriftliche Aufgabe D-6**: Messen und notieren Sie, welche Zeit der µC mit diesem Ansatz für die Übertragung der Daten in das Schieberegister benötigt.

1.4 Optimieren auf Assembler-Ebene

Stellen Sie sicher, dass im Projekt für den C++-Compiler das Speichern der temporären Dateien bei der Übersetzung eingeschaltet ist. Suchen Sie im Assembler-Code von `CShiftRegisterOutputExtender` nach der Implementierung der Methode `doShift`, in der neue Werte in das Schieberegister geladen werden. Der Name der Methode sieht etwas kompliziert aus, ist aber erkennbar[2] (Alternativ können Sie auch einen Breakpoint auf `doShift` setzen und sich den Code im Disassembler ansehen.)

Analysieren Sie den Assembler-Code. Beachten Sie, dass der Compiler beim Optimieren manchmal die Reihenfolge der Anweisungen vertauscht.

✎ **Schriftliche Aufgabe D-7**: Wie viele Assembler-Befehle sind für die Realisierung der Schleife zuständig und wie viele für die Ansteuerung der Ausgang-Pins?

Betrachtet man das Verhältnis der Befehle, die die Schleifenfunktionalität implementieren (Test auf Ende, Inkrementieren des Zählers, Sprung). und der Befehle, die die Ausgänge ansteuern, so erweist sich die Implementierung der Schleife als ein signifikanter „Overhead“.

Da die Schleife immer acht mal durchlaufen wird und der in der Schleife ausgeführte Code nicht allzu umfangreich ist, liegt es nahe, die Schleife „aufzurollen“. D. h. die Anweisungen werden explizit acht mal in den Code geschrieben. Damit kann die Zeit für die Ausführung der Befehle zur Schleifenkontrolle eingespart werden. Diesen Vorgang bezeichnet man als „Loop Unrolling“ und er kann automatisch vom Compiler ausgeführt werden, wenn man ihm erlaubt, etwas mehr Speicherplatz für das Programm zu verbrauchen.

Wechseln Sie wieder auf den Optimierungsmodus „-Ofast“. Diese Einstellung beinhaltet die „Erlaubnis“, Schleifen aufzurollen. Vorausgesetzt Ihr Code in der Schleife ist nicht unnötig kompliziert, sollten Sie jetzt im Assembler-Code oder im Disassembler sehen können, dass die Schleife „aufgerollt“ wurde. Falls nicht, vereinfachen Sie den Code in der Schleife.

In meiner Lösung werden für das Übertragen eines Bits nur noch 6 Assembler-Befehle generiert, von denen 5 ausgeführt werden – ich wüsste nicht, wie man den Compiler hier „schlagen“ sollte, d. h. wie man die Aufgabe in einem selbst geschriebenen Assembler-Programm effizienter lösen können sollte.

Leider reicht die Auflösung unseres billigen Logik-Analysators nicht, um das Taktsignal am Schieberegister aufzuzeichnen. Aber da es mit einem Maschinenbefehl auf High-Pegel und mit dem nächsten wieder auf Low-Pegel gesetzt wird, ist davon auszugehen, dass es nur 1/84 MHz, also 12 ns auf dem High-Pegel liegt. Schauen Sie mal in das Datenblatt des SN74HC595. Wenn das so stimmt, haben wir damit die Timing-Anforderung des Schieberegisters verletzt, d. h. ein sicherer Betrieb der Schaltung ist nicht mehr garantiert, wir müssen unseren Code wieder weniger perfor-

2 Wie Sie wissen, können Methoden in C++ überladen werden, d. h. es kann mehrere Methoden mit gleichem Namen geben, wenn sich die Liste der Parametertypen unterscheidet. Assembler kennt aber nur einfache Bezeichner als Sprungmarken. Deshalb wird für jede Methode ein Bezeichner generiert, der den Klassennamen, den Methodennamen und die Typen der Parameter (als Folge von angehängten Buchstaben und Typnamen) enthält. Die genauen Regeln für den gcc finden Sie, wenn es Sie interessiert, unter https://itanium-cxx-abi.github.io/cxx-abi/abi.html-#mangling.

mant bekommen. Bauen Sie das Zurücksetzen des Taktsignals über das BSRR also am besten wieder aus.

2 Echtzeituhr (RTC)

Auf dem Chip des STM2F401RE befindet sich auch eine Echtzeituhr (Real Time Clock, RTC). Diese Uhr bzw. dieser Teil des Chips kann über den V_{BAT}-Anschluss unabhängig vom Rest mit Spannung versorgt werden und benötigt bei Raumtemperatur weniger als 1 µA für den Betrieb der Uhr.

Die Ansteuerung der Echtzeituhr ist in [RM-F401] beschrieben. Leider ist insbesondere die Initialisierung dieser Peripheriekomponente nicht ganz einfach. Im Startprojekt für das Labor finden Sie daher eine Klasse `CRtc`, mit der Sie die Echtzeituhr einfach nutzen können.

2.1 Anzeige der Uhrzeit

Schreiben Sie eine Funktion `task2` (falls Sie diese Aufgabe zuerst bearbeiten `task1`), die im „Super-Loop“ die von der Echtzeituhr gelieferte Zeit als Stunden und Minuten im 24-Stunden-Format auf der 7-Segment-Anzeige anzeigt.

Da an das Evaluations-Board keine Batterie angeschlossen ist, werden Sie auf der Anzeige natürlich nicht die tatsächliche Uhrzeit sehen, sondern die Zeit, die vergangen ist, seit Sie das Board an die Spannungsversorgung angeschlossen und das erste Mal die Uhr in Betrieb genommen (d. h. den Konstruktor von `CRtc` aufgerufen) haben. Die Echtzeituhr wird nur zurückgesetzt, wenn Sie die Spannungsversorgung unterbrechen. Bei einem Reset (Drücken des schwarzer Tasters auf dem Board) oder einer Neuprogrammierung läuft die Echtzeituhr weiter.

2.2 Einstellen der Uhrzeit

Das Einstellen der Uhrzeit soll über die vier rechten Taster möglich sein. So lange der Taster ganz rechts gedrückt ist, wird die Einerstelle der Minutenanzeige im Sekundentakt erhöht. Nur die Einerstelle wird erhöht, es erfolgt kein Übertrag, d. h. auf z. B. „18:49“ folgt „18:40“. Das Drücken des zweiten Tasters von rechts bewirkt entsprechend eine Erhöhung (nur) der Zehnerstelle der Minutenanzeige (im Sekundentakt). Das Drücken des dritten Tasters von rechts bewirkt entsprechend ein Erhöhen der Einerstelle der Stundenanzeige und das Drücken des vierten Tasters von rechts ein Erhöhen der Zehnerstelle der Stundenanzeige.

Beim Erhöhen der Zehnerstelle der Stundenanzeige müssen Sie einen Sonderfall beachten. Wenn die Einerstelle kleiner vier ist, sind gültige Werte null, eins und zwei. Wenn die Einerstelle einen Wert größer oder gleich vier hat, ist zwei kein gültiger Wert für die Zehnerstelle.

Für die Implementierung können Sie den gleichen Ansatz wie bei der Ein-Knopf-Helligkeitsteuerung im vierten Labor wählen: ein `CPolledTimer`, mit dessen Hilfe Sie jede Sekunde die gedrückten Tasten auswerten.

Die Bearbeitung dieser Aufgabe hat kein neues Wissen oder neue Fähigkeiten erfordert, lediglich die Kompetenzen, die Sie im Verlauf des Labors erworben haben sollten. Sie ist gewissermaßen ein Selbsttest dieser Kompetenzen. Sie können die Bearbeitung der Aufgabe jetzt beenden und die Hard-

ware z. B. als Tischuhr und bleibende Erinnerung an dieses Labor verwenden. (In der Vertiefung AuI des Studiengangs EIT werden Sie die Hardware – Stand heute – allerdings mindestens teilweise in mindestens einem weiteren Labor wiederverwenden).

2.3 Erweiterungen

Die Bearbeitung der folgenden Vorschläge zur Erweiterung der Uhr ist freiwillig. Die Aufgaben können zur Übung und Vertiefung genutzt werden.

2.3.1 Sekundenanzeige als BCD

Zeigen Sie zusätzlich die Sekunden der aktuellen Uhrzeit auf den LEDs als BCD-Zahl an.

2.3.2 Sekundenanzeige als Balken

Zeigen Sie zusätzlich die Sekunden auf den LEDs als „wachsenden Balken“ an. Jede leuchtende LED steht damit für 1/9 einer Minute. Damit lässt sich zwar die Uhrzeit nicht sekundengenau angeben, aber man kann doch sagen, ob die angezeigte Uhrzeit (mehr oder weniger) „kurz vor“, „kurz nach“ einem Minutenwechsel oder zwischen zwei Minutenwechseln liegt.

2.3.3 Sekundentakt

Lassen Sie den Punkt zwischen der zweiten und der dritten Stelle der 7-Sekment-Anzeige im Sekundentakt (1 Hz) blinken. Diese Aufgabe lässt auf auf zwei Arten lösen. Sie können einen weiteren `CPolledTimer` mit einer Wartezeit von 500 ms anlegen und zum Toggeln des Dezimalpunkts verwenden. Allerdings erfolgt damit der Wechsel der angezeigten Zeit nicht synchron zum Blinken des Dezimalpunkts.

Die etwas anspruchsvollere Lösung besteht darin, den Sekundenwechsel bei der von der Echtzeituhr gelieferten Zeit zu erkennen, und die LED einzuschalten. Zusätzlich wird ein Timer gestartet, der nach 500 ms die LED wieder ausschaltet.

2.3.4 Sekundenanzeige

Modifizieren Sie die Klasse `CSevenSegmentDisplay` so, dass die Zuweisung eines negativen Werts an eine Anzeigestelle diese Stelle abschaltet.

Erweitern Sie jetzt die Funktionalität der Anwendung. Solange der Taster ganz links gedrückt wird, sollen auf den letzten zwei Stellen der 7-Segment-Anzeige die Sekunden der aktuellen Uhrzeit angezeigt werden. Die ersten beiden Stellen werden ausgeschaltet.

2.3.5 Zwei-Knopf-Zeiteinstellung

Sicher kennen Sie von anderen Geräten mit mehrstelligen Anzeigen die typische „Zwei-Knopf-Einstellmethodik“. Langes Drücken eines Tasters lässt die erste Stelle der Anzeige blinken, jedes weitere Drücken lässt die nächste Stelle blinken bis entweder alle Stellen durchgeschaltet sind oder ein Timeout eintritt. Während eine Stelle blinkt, schaltet ein Druck auf den zweiten Taster den an dieser Stelle angezeigten Wert weiter.

Realisieren Sie diese Einstellmethodik für die Zeitanzeige.

2.3.6 Helligkeitssteuerung

Ermöglichen Sie es, mit dem Potentiometer die Helligkeit der Anzeige einzustellen.

2.3.7 „Nachtabsenkung“

Vermindern Sie automatisch um 23:00 die Helligkeit um z. B. 30% und erhöhen Sie sie um 8:00 wieder.

2.3.8 Bedienung über die serielle Schnittstelle

Schaffen Sie eine Möglichkeit, die aktuelle Uhrzeit über die serielle Schnittstelle einzustellen. Wenn Sie die „Nachtabsenkung“ implementiert haben, sollten auch die An- und Abschaltzeiten dafür über die serielle Schnittstelle einstellbar sein.

2.3.9 Automatische Helligkeitssteuerung

(Ergänzung von Holger Frank)

In Ihrer Bauteile-Sammlung befindet sich ein lichtabhängiger Widerstand (Light depentant resistor – LDR) mit dem Sie eine automatische Helligkeitssteuerung der 7-Segmentanzeige realisieren können.

Ermitteln Sie zunächst den Widerstand, den der LDR bei Dunkelheit und starker Beleuchtung annimmt. (Für den Fall, dass Sie kein Multimeter mit Widerstandsmessung besitzen, wird unten ein alternativer Weg beschrieben.)

Um es möglichst dunkel zu machen, stülpen Sie eine schwarze Haube über den LDR, die auch an den Seiten abdunkelt. Für den Hellwert ist z.B. die Beleuchtung mit der Handy-Taschenlampe aus ca. 5cm Abstand eine Möglichkeit.

Legen Sie mit den ermittelten Widerstandswerten des LDR den lichtabhängigen Spannungsteiler von Abbildung 1 aus und bauen Sie ihn auf dem Breadboard auf[3]:

Beachten Sie hierzu folgende Punkte:

- Wird R1 sehr klein gewählt, ist der Querstrom I_q durch den Spannungsteiler beim minimalem Wert des LDR (Hell) entsprechend groß und belastet die Stromversorgung des Systems unnötig stark.

3 Wenn Sie kein Multimeter mit Widerstandsmessung besitzen, können Sie natürlich diese Schaltung auch in Kombination mit unserem in Labor 3 entwickelten „Multimeter“ verwenden, um sich einem sinnvollen Wert für R1 experimentell zu nähern. Beginnen Sie mit einem hohen Widerstand für R1 (z. B. 1 MOhm – da sollte keine Gefahr bestehen, dass I_q so groß wird, dass der LDR zerstört werden kann) und verbinden Sie den Messeingang statt mit dem Abgriff (mittleren Anschluss) des Potentiometers mit dem Anschluss PA_5. Verringern Sie den Wert von R1 so lange, bis Sie zwischen hell und dunkel einen signifikanten Spannungsunterschied messen können. Verbinden Sie dann den Anschluss des Messeingangs wieder mit dem Potentiometer.

- Je größer R1 gewählt wird, desto geringer fällt die Spannung am Mittelabgriff des Spannungsteilers bei maximalen Wert des LDR (Dunkel) aus. => Der Wertebereich des Analogeingangs ist dann unnötig eingeschränkt.

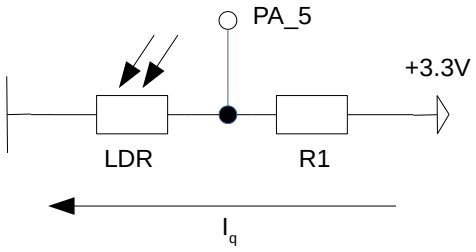


Bild 1: Spannungsteiler mit LDR

Realisieren Sie in der Software einen weiteren Analog-Eingang an PA_5 und ermitteln Sie den digitalen Hell- und Dunkelwert mit dem Debugger.

Skalieren Sie im letzten Schritt den ausgelesenen Wert so, dass sich bei maximaler Helligkeit der Wert 100 und bei minimaler Helligkeit der Wert 5 ergibt, so dass der Wert direkt mit `CSevenSegmentDisplay::setBrightness()` verwendet werden kann (Die Geradengleichung lässt grüßen).