
Systemnahe Programmierung II
M.Sc. Benedikt Klein - DHBW Ravensburg Campus Friedrichshafen

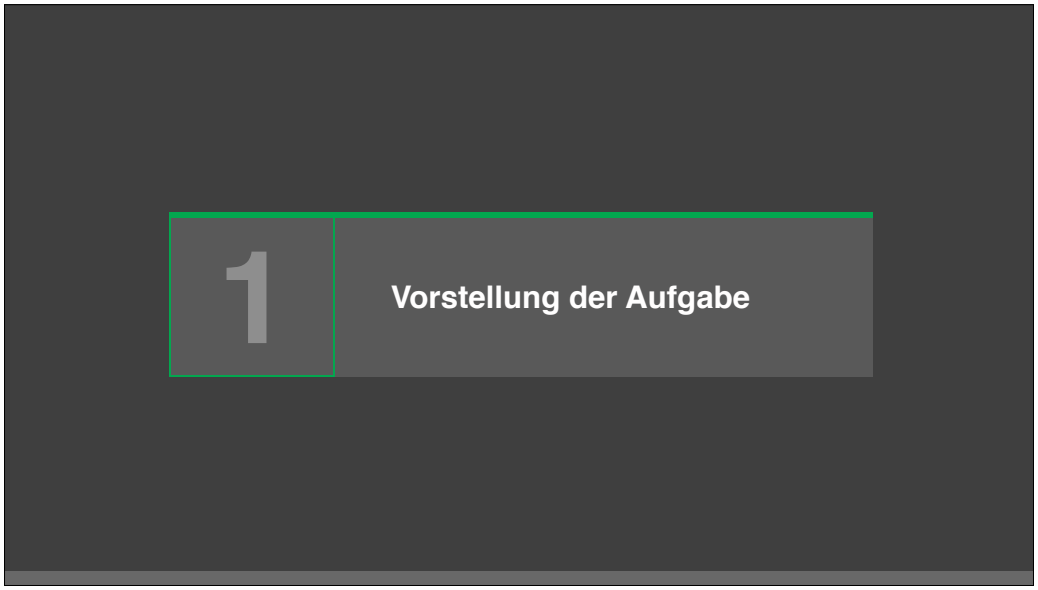
- Voraussetzungen
  - Vorlesung: Systemnahe Programmierung I
  - Grundlagen digitaler Systeme (inkl. Elektrotechnik)
- Zeitlicher Aufwand
  - 33h Vorlesung
- Prüfungsleistung
  - benotete Projektarbeit
  - Teamarbeit (4-5 Personen pro Team)
- Kontakt
  - b.m.w.klein@arcor.de
- Working Space
  - Raspberry Pi oder virtuelle Umgebung (ARM-Core)
  - Rasbian (Linux)
  - GCC, GDB
  - Embedded Projekt-Hardware

LITERATUR

- Larry D. Pyeatt: Modern Assembly Language Programming with the ARM Processor, Newnes, 1. Auflage 2016, ISBN 978-0-12-803698-3
- Tobias Häberlein: Technische Informatik - Ein Tutorium der Maschinenprogrammierung und Rechnertechnik, 1. Auflage 2011, ISBN 978-3-8348-1372-5
- Eben Upton: Learning Computer Architecture with Raspberry Pi, Wiley, 1. Auflage 2.9.2016, ISBN 978-1-119-18393-8
- ARM Infocenter, http://infocenter.arm.com, letztes Abrufdatum: 10.01.2018, Achtung: ggf. Registrierung notwendig
- ARM Developer-Center, https://developer.arm.com/, letztes Abrufdatum: 10.01.2018, Achtung: ggf. Registrierung notwendig
- Azeria Laboratories, https://azeria-labs.com/, letztes Abrufdatum: 10.01.2018

Inhalt

1. Vorstellung der Aufgabe
2. Systemkomponenten & Aufbau
3. Aufgabenstellung
4. Hinweise


1
Vorstellung der Aufgabe

Anforderungen an die Aufgabenstellung

1. Vorstellung der Aufgabe

- Systemnahe Programmierung
  - Vertiefung der Vorlesungsinhalte
  - Programmiersprache Assembler
  - Hardwarenahe Programmierung
- Aufwand:
  - 4-5 Personen pro Team
  - ca. 33h Vorlesungsstunden
- Vorkenntnisse:
  - Softwareentwicklung / anderer Programmiersprachen
  - Elektrotechnik (Basic)
  - Digitaltechnik (Basic)

Projektidee: M&M-Sortiermaschine

1. Vorstellung der Aufgabe

https://www.youtube.com/watch?v=ceGlMV4sHnk

2 Systemkomponenten & Aufbau

Zugang über den WLAN-AP der M&M-Sortiermaschine

2. Systemkomponenten & Aufbau

- SSID: MMSortRPi
- Password: dhbw-RP!MMS0r7
- IP des RPi: 192.168.4.1

- User: pi
- Password: raspberry

Pinbelegung am Raspberry Pi Zero W

2. Systemkomponenten & Aufbau

Raspberry Pi (GPIO)	Signal	Label	Hardware
2	Output	SER	7-Segment
3	Output	SRCLK	7-Segment
4	Output	nSRCLR	7-Segment
5	Output	RCLK	7-Segment
6	Output	A	7-Segment
7	Output	B	7-Segment
8	Input	nBTN1	Taster
9	Input	nBTN2	Taster
10	Input	nBTN3	Taster
11	Output	nRSTOut	Outlet
12	Output	StepOut	Outlet
13	Output	StepCW	Color-Wheel
14	TX	UART_TX	Serielle Com.
15	RX	UART_RX	Serielle Com.

Raspberry Pi (GPIO)	Signal	Label	Hardware
16	Output	DirCW	Color-Wheel
17	Output	nRSTCW	Color-Wheel
18	Output	ledSig	Color LEDs*1
19	Output	GoStop	Feeder
20	Input	nHallCW	Hallsensor des Color-Wheels
21	Input	nHallOutlet	Hallsensor des Outlets
22	Input	colorBit0	Farberkennung
23	Input	colorBit1	Farberkennung
24	Input	colorBit2	Farberkennung
25	Input	objCW	Objektsensor des Color-Wheels
26	Output	DirOut	Outlet
27	Output	nSLP	Sleep-Signal für den Co-Prozessor

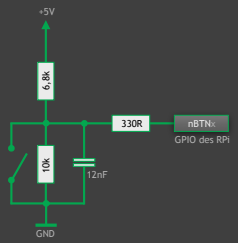
nXX - Hierbei handelt es sich um Low-Aktive-Signale.

*1: GPIO18 - ledSig: Bitte konfiguriert diesen Pin nur dann, wenn ihr nicht die WS2812B-Bibliothek des Dozenten verwenden wollt. Anderenfalls kann die Bibliotheksfunktion den Ausgang nicht korrekt nutzen!

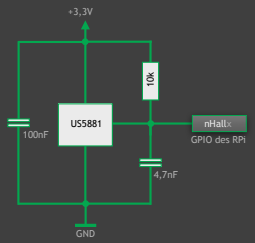
Schaltungselemente und deren Verwendung

2. Systemkomponenten & Aufbau

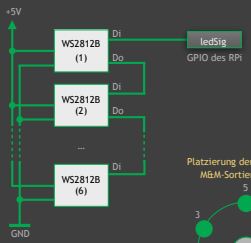
- Taster:



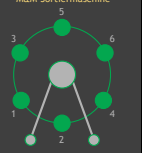
- Hallsensoren:



- Color-LEDs:

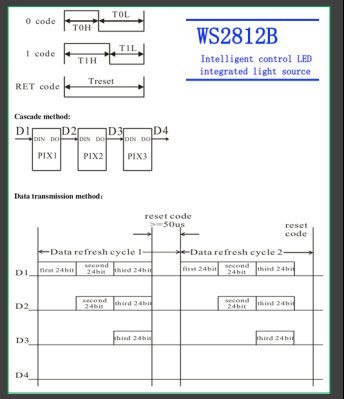


Platzierung der LEDs in der M&M-Sortiermaschine



Schaltungselemente und deren Verwendung

- Ergänzung zu den Color-LEDs:
  - Jede WS2812B LED verfügt über einen Controller der, über ein definiertes Datenprotokoll, die anzuzeigende Farbe übermittelt bekommt. Diese Farbe behält der Controller bei, bis die Spannung des Systems wegfällt oder eine neue Farbe zur Anzeige übertragen wurde.
  - Die Datenübertragung erfolgt über den GPIO-Pin *„ledSig“*, wie er zuvor in der Tabelle dargestellt wurde.
  - Verwendungsfall: Bare-Metal-Systems
    - Hier ist, dem Datenblatt folgend, das Datenprotokoll zur Steuerung der LEDs der WS2812B zu implementieren.
  - Verwendungsfall: Linux-Basis (Raspbian)
    - Hier ist entweder ebenfalls das Datenprotokoll zur Steuerung der LEDs zu implementieren oder die bereitgestellte Bibliothek zu nutzen.

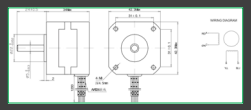


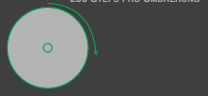
Schaltungselemente und deren Verwendung

2. Systemkomponenten & Aufbau

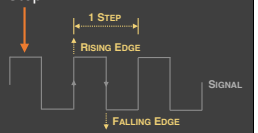
- Motortreiber für Color-Wheel und Outlet
  - Nema 17 Motor

COMMON RATEING		SPECIVICATIONS	
STEP ANGLE	1.8°±5%	VOLTAGE	12V
PHASES	2	CURRENT	0.33A
INSULATION RESISTANCE	100Mohm(500V DC)	INDUCTANCE	46±20% Mh
CLASS OF INSULATION	B	RESISTANCE	34±10%
WEIGHT	0.20Kg	HOLDING TORQUE	0.23N.M





  - EasyDriver v4.5 mit NEMA 17 Motor
    - Color-Wheel-EasyDriver ist konfiguriert im 1/8-Step Modus (resultiert in 1600 Steps / Umdrehung)
    - Outlet-EasyDriver ist konfiguriert im 1/2-Step Modus (resultiert in 400 Steps / Umdrehung)
    - erwartet Stepping der einzelnen Schritte durch den Master (hier Raspberry Pi) am Stepx-Pin
    - Richtungsvorgabe durch High- (CCW) oder Low-Signal (CW) am Dirx-Pin
    - nRSTx-Signal & nSLP-Signal zum Betrieb des Motors auf High-Level setzen
      - Ist das nSLP-Signal nicht auf diesem Level, sendet der Co-Prozessor den Motortreiber in den Schlafmodus.



Der EasyDriver verwendet den Allegro A3967 IC, welcher eine maximale Step-Frequenz von 500kHz ermöglicht. Dies ist natürlich ein idealisierter Wert, welcher in dem jeweiligen System nicht erzielt werden muss.

Signale von und zu dem Co-Prozessor

2. Systemkomponenten & Aufbau

Sleep-Command:
Durch diesen GPIO-Pin kann der „Sleep-Modus" des Co-Prozessors aktiviert werden. Dies bedeutet, dass die Motor-Treiber in ihren Sleep-State gebracht werden und somit keinen Strom verbrauchen. Dadurch werden die beweglichen Teile nicht in Position gehalten (blockiert) und können bei Bedarf manuell bewegt werden.

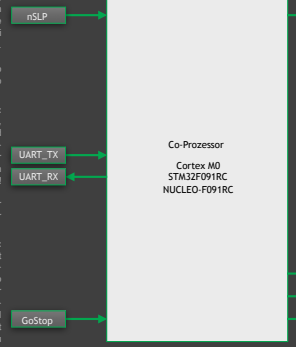
High-Level —> Wake-Up
Low-Level —> Sleep

UART-Communication:
Durch diese Pins kann eine serielle Kommunikation, zur Übermittlung von weiterführenden Befehlen und Daten, zwischen dem Raspberry Pi und dem Co-Prozessor etabliert werden. Diese Funktion ist zur Zeit noch nicht realisiert und wird im aktuellen Entwicklungsstand nicht benötigt!

TX —> Übermittlung von Daten zum Co-Prozessor
RX —> Empfang von Daten vom Co-Prozessor

Feeder-Command:
Durch diesen GPIO-Pin kann der Feeder gestartet oder gestoppt werden. Nach dem Erhalt des Start-Befehls betreibt der Co-Prozessor den Feeder so lange bis der Stop-Befehl gesendet wurde. Ist der Zulauf durch M&Ms blockiert, so pausiert der Co-Prozessor den Feeder selbstständig. In diesem Fall ist kein erneutes Senden der Signale erforderlich. Ist der Sortiervorgang beendet ist das STOP-Signal zu senden.

High-Level —> START
Low-Level —> STOP

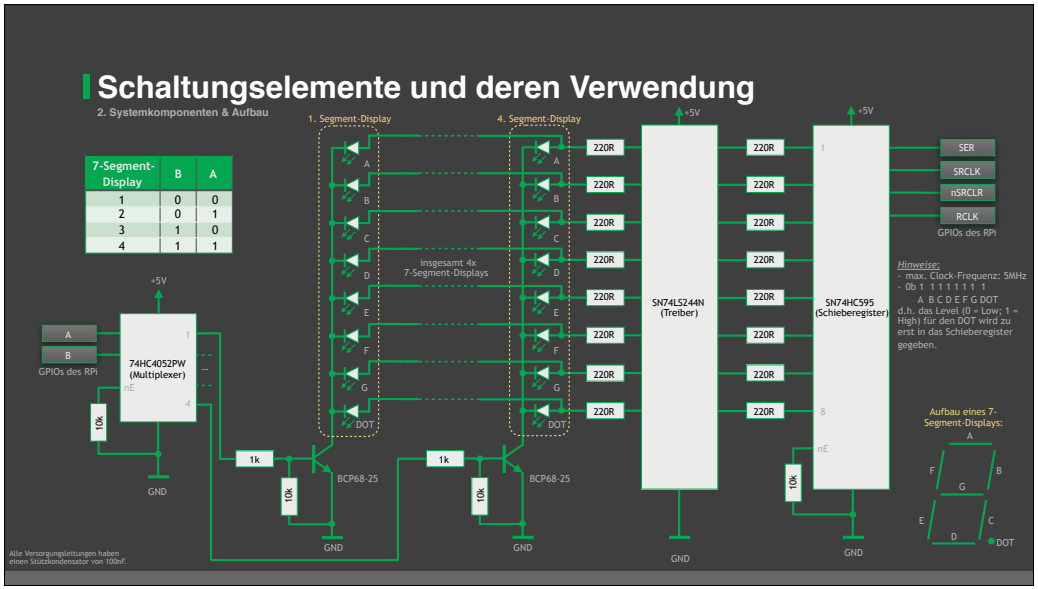


Color-Wheel Objektsensor:
Wird ein Objekt durch den Objektsensor im Color-Wheel erkannt, gibt der Co-Prozessor diese Information, über diesen GPIO-Pin, an den Raspberry Pi weiter.

High-Level —> Objekt erkannt
Low-Level —> kein Objekt erkannt

Color-Detection:
Die Farberkennung, d.h. die Auswertung des Sensors, erfolgt durch den Co-Prozessor. Diese Erkennung läuft permanent und muss nicht gestartet oder gestoppt werden. D.h. auch, dass nach dem Positionieren einer neuen M&M vor dem Sensor, eine kurze Wartezeit eingehalten werden muss. Nur so hat der Co-Prozessor eine Chance die Farben zuverlässig zu erkennen. Die Wartezeit muss durch die Studenten evaluiert werden. Die erkannten Farben beschränken sich auf die Auswahl der M&Ms und werden in Farbcodes abgebildet. Diese können an den GPIO-Pins colorBit0 bis colorBit2, entsprechend der dargestellten Tabelle, zur Auswertung herangezogen werden.

Color	color Bit2	color Bit1	color Bit0
Red	0	0	1
Green	0	1	0
Blue	0	1	1
Brown	1	0	0
Orange	1	0	1
Yellow	1	1	0
NA	0	0	0


Schaltungselemente und deren Verwendung
2. Systemkomponenten & Aufbau
7-Segment-Display | B | A
1 | 0 | 0
2 | 0 | 1
3 | 1 | 0
4 | 1 | 1
1. Segment-Display
4. Segment-Display
insgesamt 4x 7-Segment-Displays
+5V
A
B
GPIOs des RPi
74HC4052PW (Multiplexer)
nE
10k
GND
1k
BCP68-25
DOT
220R
SN74LS244N (Treiber)
SN74HC595 (Schieberegister)
SER
SRCLK
nSRCLR
RCLK
GPIOs des RPi
Hinweise:
- max. Clock-Frequenz: 5MHz
- 0b 1 1 1 1 1 1 1 1
A B C D E F G DOT
d.h. das Level (0 = Low; 1 = High) für den DOT wird zu erst in das Schieberegister gegeben.
Aufbau eines 7-Segment-Displays:
Alle Versorgungsleitungen haben einen Stützkondensator von 100nF.

3 Aufgabenstellung

Gesamt: 16P

Aufgabenstellung

3. Aufgabenstellung

1. Die Software kann wahlweise für das Linux-Betriebssystem des Raspberry Pi programmiert oder mit Hilfe eines Bare-Metal-Systems implementiert werden.
    1. Wie sieht der Boot-Vorgang des Raspberry Pis nach dem Anlegen der Versorgungsspannung aus? *(7P)*
    2. Wie wird ein Bare-Metal-System für den Raspberry Pi erzeugt? *(3P)*
    3. Was sind die Unterschiede zum „normalen“ Betrieb? Muss bei der Programmierung auf etwas geachtet werden? *(6P)*

Gesamt: 24P

Aufgabenstellung

3. Aufgabenstellung

2. Systemarchitektur / Dokumentation
   1. Für welche Systembasis entscheidet sich die Gruppe? *(6P)*
      - Bare-Metal-System oder Linux-Basis?
      - Bitte begründet eure Entscheidung.
   2. Wie ist die Software aufgebaut? Was passiert wann? Wie spielen die Komponenten zusammen? *(12P)*
      - Zur besseren Verdeutlichung der Software: Bitte zeichnet ein Ablaufdiagramm der Software und fügt diesem eine Erklärung in wenigen Sätzen hinzu.
      - Besteht die entwickelte Software aus mehreren Dateien / Modulen / Ebenen, so muss deren Zusammenspiel ebenfalls in einem Diagramm dargestellt werden.
   3. Kommentare im Code *(6P)*
      - Jede Funktion muss kurz beschrieben werden. Was ist die Aufgabe der Funktion? Welche Eingangssignale/-variablen werden erwartet? Resultiert eine Ausgabe zur Weiterverarbeitung?
      - Codezeilen werden bitte durch Eure Gedanken und den Sinn / Zweck kommentiert. Vermeidet bitte Kommentar wie z.B.: add r1,#1 @ addiere 1 zu r1 … besser: „Warum tue ich XY hier?“

Gesamt: 52P

Aufgabenstellung

3. Aufgabenstellung

3. Entwicklung der M&M-Sortiermaschine (Software)
   1. Programmiersprache Assembler (ARM-Assembler) ***(6P)***
      - Die Verwendung einer anderen Programmiersprache resultiert in 0 Punkten für die gesamte Aufgabe 3.
   2. Funktionsentwicklung zur Verwendung der Komponenten *(15P + 10P + 3P + 6P = **34P**)*
      - Color-Wheel — Auswertung Objekt-Sensor, Farberkennung, Motorsteuerung und Positionierung *(15P)*
      - Outlet — Motorsteuerung und Positionierung, Anzeige der M&M-Farbe via LEDs *(10P)*
      - Feeder — Start / Stop Signalgenerierung *(3P)*
      - Taster — Abfrage der Benutzereingabe (Start/Stop des Sortiervorgangs) *(6P)*
   3. Implementierung der Logik zur Sortierung der M&Ms (**12P** oder 12P - 7P = **5P**)
      - Wie die Sortiermaschine implementiert wird ist der Gruppe überlassen. Wichtig ist, dass sie am Ende in der Lage ist selbstständig die M&Ms zu sortieren. Bitte achtet lediglich darauf, dass die Bedienung in der Dokumentation erklärt ist und die Logik mit eurer dort geleisteten Beschreibung übereinstimmt.
      - Sollte die Logik nicht lauffähig sein, d.h. die Sortierung der M&Ms nicht funktionieren, so implementiert eine Demo-Software die zeigt, dass die einzelnen Komponenten unabhängig korrekt funktionieren. *(-7P)*

Gesamt: 8P
max. Punktzahl: 100P
davon max. Bonuspunkte: +16P

Aufgabenstellung

3. Aufgabenstellung

4. Abgabe der Aufgabe innerhalb der Frist
   - Dokumentation + Source (Software) *(4P)*
   - letzter Build (Executable) *(4P)*
     - bei einem Bare-Metal-System bitte inkl. Template & kernel.img (lauffähiges SD-Karten-Abbild)

späteste Abgabe:

08.03.2020 - 23:55 Uhr

Bonus: (falls noch keine 100P erreicht wurden)

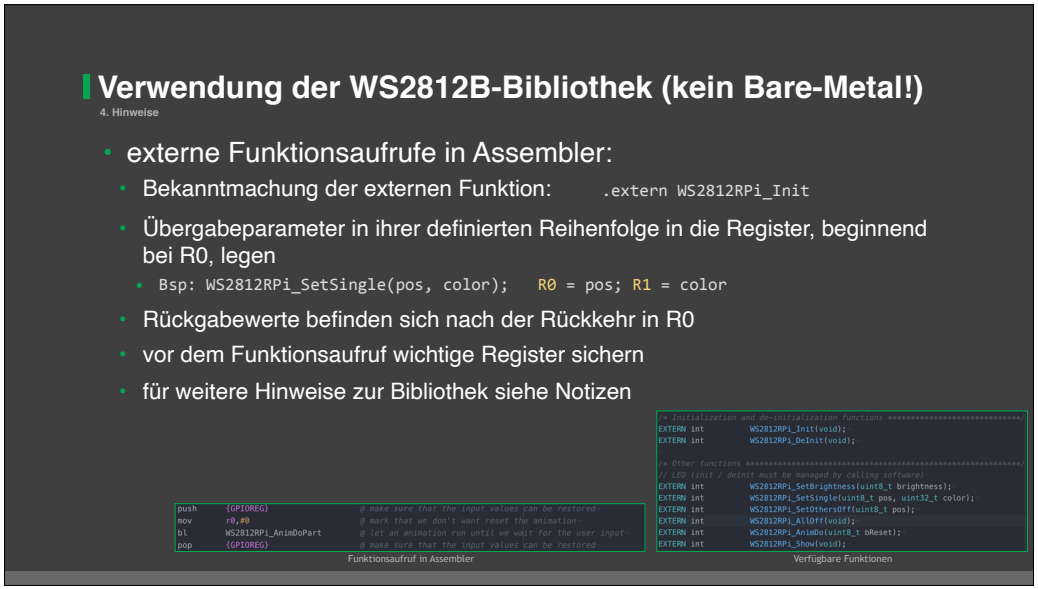
1. Input-Abfrage über Interrupts (an „sinnvollen“ Eingangssignalen) *(+4P)*
2. Verwendung des Hardware-Timers (z.B. für Wartezeiten) *(+4P)*
3. Eigene Implementierung der LED-Ansteuerung in Assembler *(+4P)*
4. Implementierung der 7-Segment-Anzeige-Ansteuerung inkl. Anzeige der sortierten M&Ms *(+4P)*

4 Hinweise

Raspberry Pi als Bare-Metal-System

4. Hinweise

- sehr guter Ausgangspunkt (inkl. Template):
  - https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/
- Erweiterung der virtuellen Umgebung:
  - für das Kompilieren des Bare-Metal-Templates in Linux-Mint
    - `sudo apt-get update`
    - `sudo apt-get install gcc-arm-none-eabi`
  - Gemeinsame Ordner (Shared Folder)
    - Datenaustausch zwischen Host und Virtual Box (Linux Mint nicht Raspbian)
    - Ordner in Virtual Box definieren
    - Ordner in Linux Mint anlegen:
      - `cd /home/dhbw`
      - `mkdir mntDir`
    - Einbinden in Linux Mint:
      - `sudo mount -t vboxsf <name_des_ordners> /home/dhbw/mntDir`



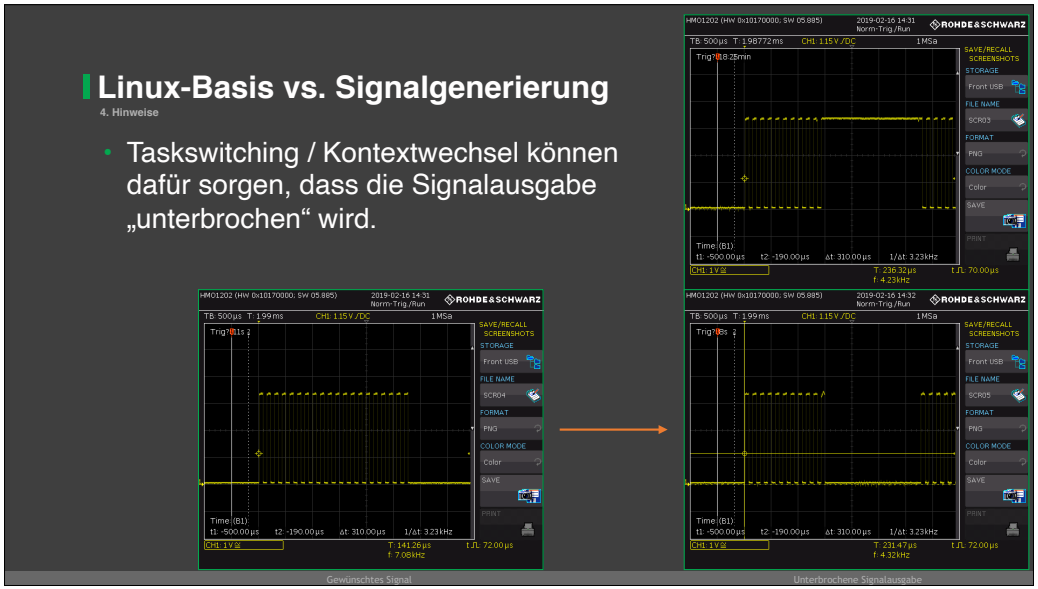
Die Bibliothek ist so aufgebaut, dass sie möglichst effizient die Hardware des Raspberry Pi nutzt. Aus diesem Grund sind ein paar Initialisierungen durch die aufrufende Software notwendig, welche nach dem Beenden wieder sicher zurückgenommen werden müssen.

1. Die Funktion WS2812RPi_Init ist aufzurufen, bevor die Funktionen der Bibliothek zum ersten Mal verwendet werden.
2. Dann können WS2812RPi_SetBrightness, WS2812RPi_SetSingle, WS2812RPi_SetOthersOff, WS2812RPi_AllOff, WS2812RPi_AnimDo und WS2812RPi_Show beliebig aufgerufen werden.
    1. WS2812RPi_SetBrightness ermöglicht die Änderung der Helligkeit. Der Übergabewert muss zwischen 0 und 100 liegen. Der Wert ist für alle LEDs gültig. Damit die Änderung sichtbar wird muss die Funktion WS2812RPi_Show im Anschluss aufgerufen werden.
    2. WS2812RPi_SetSingle lässt es zu eine LED an einer bestimmten Position (siehe Folien) mit einer bestimmten Farbe leuchten zu lassen. Die Position ist von 1 bis 6 definiert. Der Farbcode ist als 24Bit RGB abzubilden, d.h. z.B. Rot bekommt den Übergabewert 0xFF0000. Auch hier ist WS2812RPi_Show im Anschluss aufzurufen.
    3. WS2812RPi_SetOthersOff ermöglicht es alle LEDs, bis die an einer bestimmten Position, auszuschalten. Auch hier ist WS2812RPi_Show im Anschluss aufzurufen.
    4. WS2812RPi_AllOff schaltet alle LEDs aus.
    5. WS2812RPi_AnimDo führt eine Animation auf den LEDs aus und muss zyklisch hintereinander aufgerufen werden damit die Animation sichtbar wird. Zyklisch bedeutet hier z.B. alle 100ms. Bitte achtet auf eine Wartezeit, sollten die 100ms zu kurz sein, kann dieser Wert einfach höher gesetzt werden. Dies hat den Vorteil, dass die Software nicht blockierend wirkt, d.h. es können z.B. Taster-Abfragen während der Wartezeit durchgeführt werden. Der Übergabewert sollte immer 0 sein, bei einer 1 wird die Animation zurückgesetzt und dies ist selten wirklich notwendig.
    6. WS2812RPi_Show sendet die neuen Farbinformationen an die LED-Controller und aktualisiert somit deren Konfiguration.
3. Nachdem die Funktionen der Bibliothek nicht mehr benötigt werden muss WS2812RPi_DeInit aufgerufen werden.

Linux-Basis vs. Signalgenerierung

4. Hinweise

- Taskswitching / Kontextwechsel können dafür sorgen, dass die Signalausgabe „unterbrochen“ wird.



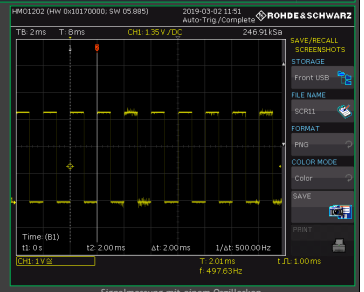
Dies stellt für das Steppen eines Motors, mit den hier genutzten Motortreibern, nur ein kleines bis kein Problem dar. Will man jedoch eine Kommunikation implementieren ist das zeitliche Verhalten deutlich entscheidender und daher können diese „Unterbrechungen“ zu falschen Übertragungen führen.

Zeit: Verwendung ohne Hardware-Timer

4. Hinweise

- Implementierung einer Wartezeit:
  - Betrachtung inkl.:
    - Zählwert in *WAITREG* legen (*MOV WAITREG,…*)
    - Sprung zur Funktion (*bl wait_do*)
    - Umschalten des Pin-Levels (zu Beginn und am Ende)
  - zählt vom Startwert runter auf 0
  - Zählwert für 1ms = 485.451
    - andere Art der Implementierung = anderer Zählwert
- Ergänzungen zu Timer:
  - $T_{timer} = 1 / f_{SystemClock} = 1 / f_{timer} = 1 / 8MHz = 0{,}125\mu s$
    - $f_{timer} \neq f_{SystemClock}$ wenn ein Pre-Scaler verwendet wird
  - *16-Bit:* $ticks_{max} = 2^{16} - 1 = 65535$
  - $t_{overflow} = ticks_{max} \times T_{timer} = 65535 \times 0.125\mu s = 8.19875ms$

```
@ ------------------------------------------------------------------------
@ wait helper
@   param:      WAITREG - must contain the start value to count from
@   return:     none
@ ------------------------------------------------------------------------
wait_do:
        subs    WAITREG, #1             @ count down the value of r6 and update flags
        bne     wait_do                 @ if not equal -> keep on counting
        bx      lr
```



Signalmessung mit einem Oszilloskop

arm_freq = Frequency of ARM in MHz. Default 700
core_freq = Frequency of GPU processor core in MHz. For models prior to the Pi2, this has an impact on ARM performance since it drives the L2 cache. The ARM on the Pi2 has a separate L2 cache. Default 250

Die aktuelle Frequenz des Raspberry Pi abfragen: sudo cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq —> Resultiert auf der MM-Sortiermaschine auf dem Wert 1.000.000 was 1GHz entspricht.
Auf dem Raspberry Pi wird ein Clock-Cycle benötigt für die Instruktionen. Bitte zur genaueren Betrachtung noch einmal den ersten Teil der Vorlesung heranziehen, inkl. der im Raspberry Pi verwendeten Pipeline.

Rückschluss auf die Taktfrequenz (ein Versuch):
Zeit für das Dekrementieren um 1: x = 0,001s / 485451 = $2{,}06*10^{-9}$s
Taktfrequenz: $f_{sysclk} = 1 / 2{,}06*10^{-9}s = 485{,}451MHz$ —> 2 Instruktionen = 485,451MHz * 2 = 970,902MHz

Achtung: Je nach Auslastung setzt der Raspberry Pi die Taktfrequenz herunter, sie bleibt jedoch bei mindestens 700MHz.

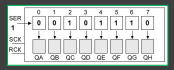
Gründe warum die Rechnung nicht genau 1GHz ergibt:
- Linux-OS Verwaltung
- Berechnung betrachtet nur die zwei Befehle zum Abarbeiten der „wait_do“-Funktion
- Pipeline und ggf. „Füll-Operationen“, da z.B. ein geforderter Wert aus dem Memory noch nicht verfügbar ist, nicht klar ersichtlich
- Befehle zum Setzen / Löschen der Pin-Level inkl. deren physikalische Verzögerung bis zum Erscheinen an dem Signalausgang

Quellen:
https://elinux.org/RPiconfig#cite_note-freq_relationship-9
https://electronics.stackexchange.com/questions/165675/how-to-calculate-the-time-of-one-single-tick-on-microcontroller

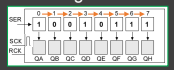
Verwendung eines Schieberegister-ICs

4. Hinweise

1. Gewünschtes Level (0 oder 1) an Pin „SER“ anlegen und nSRCLR auf High-Level setzen. (Ein Low-Level würde zum Reset der Logik führen.)



2. Generierung des Takts an Pin „SRCLK“ oder „SCK“



1 Eine steigende Flanke an diesem Pin sorgt dafür, dass der aktuell an „SER“ anliegende Wert als neuer Wert für Bit 0 übernommen wird.

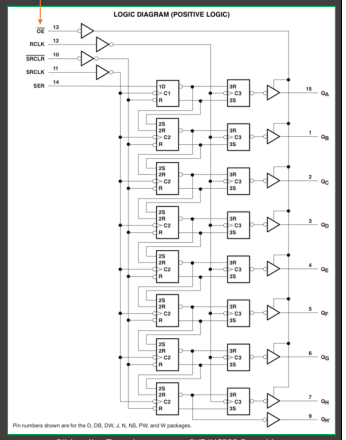
2 Alle anderen Werte werden von dem IC selbstständig „weitergeschoben“.

3. Erzeugung des Übernahmetaktes an Pin „RCLK“ oder „RCK“

1 Ein Low-High Puls (d.h. eine steigende Flanke) sorgt dafür, dass die Inhalte der Schieberegister in das Ausgangsregister übernommen werden

4. Pin „nSRCLR“ auf Low-Level setzen

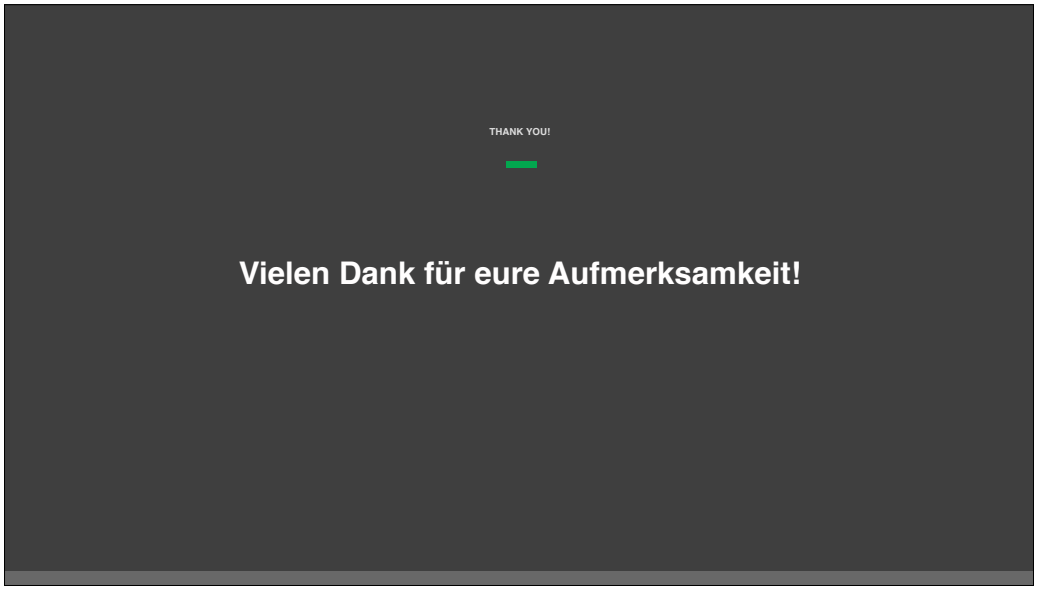
In diesem System liegt nOE, durch die Verwendung eines Pull-Downs, ständig auf dem Low-Level. Dadurch sind die Ausgänge immer aktiv. An diesem Pin ist somit keine Signalgenerierung notwendig.



Bildquelle: Texas Instrument - SN74HC595 Datenblatt

Warum existiert Pin 9 (QH')?
Durch diese Sonderstellung liegt das letzte Bit direkt am Ausgang QH' an. Dies ermöglicht es einen weiteren IC an diesen Baustein zu verknüpfen, um somit beliebig viele Schieberegister hintereinander zu schalten (kaskadieren).

Quelle:
https://www.mikrocontroller.net/articles/AVR-Tutorial:_Schieberegister#Funktionsweise (ebenso Bildquelle)
https://learn.adafruit.com/adafruit-arduino-lesson-4-eight-leds/the-74hc595-shift-register


THANK YOU!
Vielen Dank für eure Aufmerksamkeit!