Hochschule RheinMain
Fachbereich Design Informatik Medien
Prof. Dr. Robert Kaiser

Echtzeitverarbeitung

SS 2021
LV 4511 / LV 8481

Übungsblatt 2
Laborversuch
Abgabe: 4. Woche (10.05.2021)

Aufgabe 2.1. (Zeitgetriebene Aktionen – SoftPWM):

Schreiben Sie eine C-Funktion mySoftPWM(), die eine PWM in Software umsetzt (siehe auch [1]). Die Parameter sind die Zykluszeit in μs, der Tastgrad (duty cycle) im Wertebereich 0 bis 100 und der „Pin“ auf dem die PWM ausgegeben werden soll. Die Funktion soll PWM durchführen, bis das Programm beendet wird. Verwenden Sie die passenden Funktionen von wiringPi für das Schlafen zwischen den Takten.

Schreiben Sie ein kleines C-Programm, das eine PWM mit Ihrer Funktion auf dem GPIO 18 (HAT „PWM“) ausgibt. Schließen Sie das Oszilloskop an GND und den herausgeführten GPIO Pin 18 an und betrachten Sie das Signal. Variieren sie die Zykluszeit und den Tastgrad.

Was passiert, wenn Sie die Zykluszeit sehr gering wählen?

Versuchen Sie (zum Beispiel mit `screen` oder per Netzwerkverbindung mit `ssh`), mehrere Konsolen auf dem Raspberry Pi einzusetzen und lassen Sie in einem Fenster Ihr PWM-Programm laufen, während Sie im zweiten Fenster Last erzeugen, zum Beispiel indem Sie den MD5-Hash von `/dev/zero` berechnen lassen. Falls Sie nur eine Konsole haben, so starten Sie Ihr Programm als Hintergrundprozess.

Was passiert mit dem Muster der PWM? Warum?

Aufgabe 2.2. (Zeitgetriebene Aktionen – Hardware PWM):

Der Raspberry Pi kann an dem GPIO 18 (HAT „PWM“) auch eine „PWM in Hardware“ bereitstellen. Die wiringPi-Bibliothek stellt hierfür Funktionen bereit. Schreiben Sie ein kleines C-Programm, das diese HW-PWM Funktionalität benutzt.

Betrachten Sie das Signal am Oszilloskop. Wiederholen Sie auch den Lasttest. Was ist der Unterschied zwischen Software- und Hardware-PWM?

Aufgabe 2.3. (Zeitgetriebene Aktionen – Externer Taktgenerator):

Auf dem EchtzeitHat ist ein kleiner Microcontroller (μC) verbaut, der Takte mit einstellbarer Frequenz erzeugen kann. Der Ausgang des μC ist auf HAT oben rechts (OSC) ausgeführt. Über das Potentiometer kann die Frequenz verstellt werden.

Schließen Sie das Oszilloskop an den OSC-Pin und GND an. Schauen Sie sich das Signal an.

Messen und dokumentieren Sie die möglichen Frequenzen.

Aufgabe 2.4. (Zeitgetriebene Aktionen – Externe Ereignisse):

Um auf äußere Ereignisse reagieren zu können, müssen diese erkannt und in der Programmlogik verarbeitet werden. Für das Erkennen von Ereignissen gibt es mehrere Möglichkeiten, bspw. Polling und Interrupts.

Am einfachsten ist das aktive Abfragen der Zustände von IOs in einer Schleife, ähnlich funktionieren auch SPSen.

Schreiben Sie ein kleines C-Programm, das in einer Schleife den GPIO 27 abfragt und wenn dieser aktiv wird (also der Taster SW2 gedrückt), dann soll der GPIO 18 an und nach kurzer Zeit (1 μs) gleich wieder ausgeschaltet werden. Nach jedem Zyklus soll das Programm eine bestimmte, per Kommandozeilenoption wählbare Zeit lang schlafen. Verwenden Sie hier die Funktion `delayMicroseconds()` aus der wiringPi-Bibliothek.

Experimentieren Sie im Weiteren mit verschiedenen Zeitwerten.

Schließen Sie das Oszilloskop mit dem Kanal 1 an dem GPIO 18 und mit dem Kanal 2 an den GPIO 27 (den Taster) an. Setzen Sie Kanal 2 als Trigger und schalten Sie das Oszilloskop in den „Single-“ (bzw. bei Scopy: „Normal-“) Mode.

Starten Sie Ihr Programm, drücken Sie den Taster und messen Sie mehrmals die Zeit zwischen dem Taster-Ereignis und der Reaktion des Systems. Variiert diese? Was ist Ihre gemessene, was ist die theoretische minimale bzw. maximale Verzögerung (und der Durchschnitt)?

Variieren Sie die Zeit der Delay-Funktion. Erzeugen Sie wieder Rechenlast auf dem Linux-System. Was erkennen Sie? Was sind die Grenzen? Wie hoch sind die minimalen, die maximalen und durchschnittlichen Dauern, bis auf einen Tasterdruck reagiert werden kann?

Aufgabe 2.5. (Ereignisgetriebene Aktionen – Interrupts):

Die wiringPi-Bibliothek stellt Funktionen für eine Interrupt-Behandlung im Userspace bereit. Normalerweise erfolgt die Interrupt-Behandlung in den Gerätetreibern im Kernelspace.

Schreiben Sie ein kleines C-Programm, welches für den Taster 2 (GPIO 27) einen Interrupt-Handler registriert. In dem Handler wird wieder GPIO 18 an- und ausgeschaltet. Wiederholen Sie ihre Versuche aus der vorherigen Aufgabe. Was sind die Unterschiede?

Aufgabe 2.6. (Vorbereitung nächste Woche):

(a) Machen Sie sich mit dem Laden und Entfernen von Kernel-Modulen unter Linux vertraut.

(b) Recherchieren Sie zur Interrupt-Behandlung hinterher, die wiringPi bereitstellt. Wie ist diese umgesetzt? Wo liegen die Grenzen bzw. mögliche Probleme?

A. (*):

Literatur

[1] http://rn-wissen.de/wiki/index.php?title=Pulsweitenmodulation

[2] http://de.wikipedia.org/wiki/Oszilloskop

[3] http://www3.physik.uni-stuttgart.de/studium/praktika/ap/pdf_dateien/Allgemeines/OsziAnleitung.pdf

[4] http://wiringpi.com/

[5] http://wiringpi.com/reference/

B. (Raspberry Pi GPIO Pins):

Raspberry Pi – GPIO-connector

HAT	WiringPi	GPIO	Name	Header		Name	GPIO	WiringPi	HAT
			3.3V	1	2	5V			
FanSoftPWM	8	2	SDA	3	4	5V			
Fan Tacho	9	3	SCL	5	6	GND			
	7	4		7	8	TxD	14	15	
			GND	9	10	RxD	15	16	
	0	17		11	12		18	1	PWM/GPIO18
SW2	2	27		13	14	GND			
SW1	3	22		15	16		23	4	
			3.3V	17	18		24	5	
	12	10	MOSI	19	20	GND			
	13	9	MISO	21	22		25	6	
	14	11	SCLK	23	24	CE0	8	10	
			GND	25	26	CE1	7	11	
				27	28				
DRV_A_en	21	5		29	30				
DRV_A_in1		6		31	32		12	26	DRV_A_in2
DRV_B_in1	23	13		33	34				
DRV_B_in2	24	19	MISO	35	36				
DRV_B_en	25	26		37	38	MOSI	20	28	GPIO20
			GND	39	40	SCL	21	29	GPIO21