PULSE-WIDTH MODULATION

Dämmen der RGB LED mittels Pulsweitenmodulation

Christian Janeczek 5AHITT

Inhaltsverzeichnis

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
M0FAULT0	30 53 63	PF2 (4) PD6 (4) PD2 (4)	I	TTL	Motion Control Module 0 PWM Fault 0.
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	
				TTL	

tm4c123gh6pm.pdf

page 1229

Aufgabenstellung

Basiernd auf der Recherche zu PWM soll ein Example aus der tivaware-Toolchain angepasst werden um folgende 4 Modi per Buttondruck zu implementieren: 25%, 50%, 75% und 100%. SW1 soll damit beim ersten Drücken den ersten Modi aktivieren und bei jedem weiteren Drücken zum nächsten Modi wechseln. Zur Kontrolle soll der aktive Modus per UART ausgegeben werden.

Dokumentieren Sie den Einsatz von PWM, der Zeit und das Ansprechen der LEDs in einem Protokoll.

Anforderungsanalyse

- **Jegliche erhaltene Informationen sollen via UART auf der Konsole ausgegeben werden**
  Die Konfiguration des UART-Outputs ist von Nöten

- **Definition des ButtonHandlers im Startup sowie Implementierung dessen im eigentlichen Source-File**
  Den ButtonHandler bei GPIO Port F festlegen und in dem eigentlichen Source-File implementieren

- **Initialisierung des PWM-Moduls**
  Lesen des Datenblattes und der Driver-Library um Informationen bezüglich Pulse-Width-Modulation zu sammeln

- **Erhöhen des Duty Cycles mit jeder Betätigung der Switches**
  Nach jedem erfolgreichen Betätigen der Switches soll der Duty Cycle um 25% steigen. Bei 100% geschieht ein Reset

Designüberlegung

Die geschriebene Applikation soll in Header-Files zerlegt werden, um die Erweiterung des Codes zu gewährleisten. Wenn zum Beispiel ein Upgrade erfolgt und weitere Register verwendet werden müssen, dann sollen diese in einem eigenen Header-File als Globale Variable deklariert werden.

Die notwendigen Funktionen der Driver-Library sollen in richtiger Reihenfolge in Methoden Einsatz finden.

Die Definition eines ButtonHandlers ist von Nöten, der auf das Betätige eines Switches lauscht.

Nach jedem erfolgreichen Betätigen eines Switches soll der Duty Cycle mittels PWM gesetzt werden und die LED "dimmen".

Technologiebeschreibung

Die Pulsweitenmodulation(PWM) ist eine Modulationsart, bei der eine technische Größe(z.B. elektrische Spannung) zwischen zwei Werten wechselt. Dabei wir bei konstanter Frequenz der Tastgrad eines Rechteckpulses moduliert, also die Breite der ihn bildenden Impulse.[1]

Aufwandsabschätzung und Arbeitszeitaufzeichnung

Janeczek:

DATE	PHASE	TASK	ESTIMATION	ACTUAL	COMMENT
05.02.2015	Implementierung	Die Konfiguration von UART	0:30:00	1:00:00	Eine Zeile der Konfiguration ist wesentlich, beziehungsweise die Reihenfolge im Sinne der Benutzung von virtuellen Maschinen
	Implementierung	Analyse des Example-Codes PWM-Invert	1:00:00	0:45:00	Mit Hilfe der Kommentare, die im Example-Code geschrieben wurden und der Dokumentation, die in der Driver-Library zu finden ist, war die Intention des Codes klar übersichtlich und verständlich.
	Implementierung	Schreiben der eigentlichen Funktionalität	2:00:00	3:00:00	Aufgrund mangelnden Verständis musste im Datasheet und in der Driver-Library nachgeschlagen werden, um die Funktionen erfolgreich zu implementieren.
	Dokumentation	Schreiben des Protokolls	1:00:00	1:30:00	Technologiebeschreibung, PWM Konfiguration, PWM-Clock, Code-Snippets, etc.
		SUM	4:30:00	6:15:00	

Arbeitsdurchführung

Zuallererst bin ich die Anforderungsanalyse Schritt für Schritt durchgegangen und habe meine Arbeitsschritte in folgende Punkte gegliedert:

- **Lesen der Datenblätter(PLURAL) und der Driver-Library um Verständnis für PWM zu erlangen**
  Die Driver-Library[2] ist wundervolles Dokument, indem die Hintergründe der einzelnen Funktionen erläutert werden
- **Das richtige Konfigurieren des PWM-Moduls**
  Dem Verlauf der Konfiguration, wie im Datenblatt des Mikrocontrollers im Kapitel Pulse-Width-Modulation beschrieben. Die Clock kann hierbei beliebig mit dem Befehl ClockSet angepasst werden:

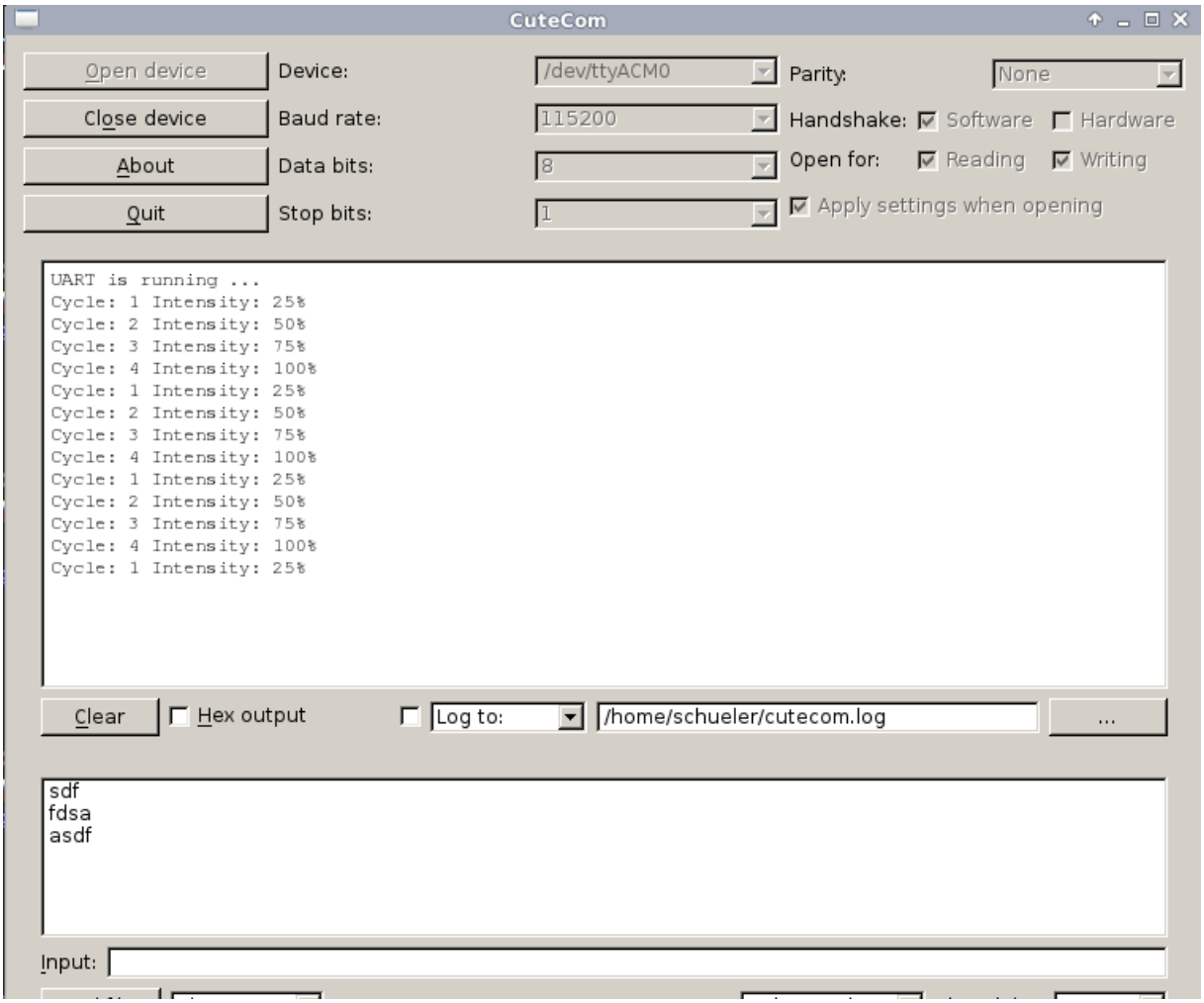
```
/*
* Set up the PWM clock
*/
ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

/*
* Enable the port for the LED
*/
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

/*
* Enable the PWM1 Generator
*/
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);

/*
* Configure the led ports for the PWM output signal
*/
GPIOPinConfigure(GPIO_PF1_M1PWM5);
GPIOPinConfigure(GPIO_PF2_M1PWM6);
GPIOPinConfigure(GPIO_PF3_M1PWM7);
GPIOPinTypePWM(GPIO_PORTF_BASE, LED_RED | LED_GREEN | LED_BLUE );

//Configure PWM generator Options
PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_UP_DOWN);
PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN);

//Set the period (expressed in clock ticks)
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, period);
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, period);

// Enable the PWM Generator
PWMGenEnable(PWM1_BASE, PWM_GEN_2);
PWMGenEnable(PWM1_BASE, PWM_GEN_3);
```

- **Die Implementierung des ButtonHandlers + Duty Cycle**
  Eine Variable, die hochgezählt wird um die einzelnen Duty Cycles zu simulieren. Mithilfe des PulseWidthSet Befehls, werden die einzelnen Duty Cycles letzten Endes gesetzt.

```
GPIOIntClear(BUTTONS_GPIO_BASE, ALL_BUTTONS);
        //Enabling the Output State for the PWM-signal
        PWMOutputState(PWM1_BASE, outbit, true);

        switch(cycle)
        {
                case 0:
                        cycle++;
                        //setting the duty cycle on 25%
                        PWMPulseWidthSet(PWM1_BASE, out, 1249);
                        UARTprintf("Cycle: %i Intensity: 25%%\n", cycle);
                        break;

                case 1:
                        cycle++;
                        //setting the duty cycle on 50%
                        PWMPulseWidthSet(PWM1_BASE, out, 2499);
                        UARTprintf("Cycle: %i Intensity: 50%%\n", cycle);
                        break;

                case 2:
                        cycle++;
                        //setting the duty cycle on 75%
                        PWMPulseWidthSet(PWM1_BASE, out, 3749);
                        UARTprintf("Cycle: %i Intensity: 75%%\n", cycle);
                        break;

                case 3:
                        //setting the duty cycle on 100%
                        PWMPulseWidthSet(PWM1_BASE, out, 4999);
                        UARTprintf("Cycle: %i Intensity: 100%%\n", cycle);
                        cycle = 0;
                        //resetting the counter variable
                        break;
        }
```

Testbericht

- Eine ordnungsgemäße Pulsweitenmodulation:



In diesem Fall wurden die Switches mehrere Male betätigt.

Conclusio

- Die Berechnung der Duty Cycles war ein sogenannter "Pain in the Ass".
- All in all: Pulse-Width-Modulation ist sexy.
- Wenn man in einer Virtuellen Maschine arbeiten sollte, muss man bei der Konfiguration der stillen Freundin UART, auf die Reihenfolge der Befehle achten, da eine falsche Reihenfolge zu einer Awkward Silence füehen kann.
- Mike brachte uns bei, dass das menschliche Organ "Auge" leicht zu manipulieren ist und ab einer gewissen Frequenz keinen Unterschied wahrnimmt.

Quellenangabe

[1] **Pulse Width Modulation,** Wikipedia**,** http://de.wikipedia.org/wiki/Pulsweitenmodulation

[2] **Driver-Library,** Tiva**,** https://github.com/mborko/tiva-template/blob/master/docs/SW-TM4C-DRL-UG-2.0.1.11577.pdf

[3] **Tiva Template,** Michael Borko**,** https://github.com/mborko/tiva-template