Mobiles Hardware-Praktikum 2017

Jan Burchard, Tobias Schubert, Bernd Becker

Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau

Übungsblatt 3 (24 Punkte)

Abgabe: 24.05.2017 bis 17:00 Uhr

Ziel:

Die Funktionen des ATmega328 Microcontrollers werden durch hardwarenahe Befehle gesteuert.

Hinweise:

Beachten Sie die Folien zur hardwarenahen Programmierung des Arduino und die Code-Beispiele auf der Website. Diese enthalten viele Tipps und Beispiele. Beachten Sie außerdem, dass ihr Code ausreichend kommentiert sein sollte, insbesondere wenn Register geschrieben oder gelesen werden.
Vergessen Sie nicht einen Pin als Ausgang zu definieren wenn Sie ihn verwenden.

Sie können die Aufgaben 1-3 und 4-8 jeweils zusammen in einer Datei abgeben.

An den kommenden Donnerstagen (19.05. und 26.05.) findet auf Grund der Feiertage keine Vorlesung und damit auch keine Fragestunde statt. Bei Fragen und Problemen können Sie über das Forum mit uns in Kontakt treten.

Aufgabe 1 (2 Punkte):

Entwickeln Sie eine Funktion “setPin12(boolean high)” **nach dem Vorbild aus der Vorlesung**. Diese soll ohne “digitalWrite()” sondern mit direktem Schreiben in das entsprechende Register Pin 12 auf 1 ($high = true$) oder 0 ($high = false$) setzen.
Hinweis: Arduino Pin 12 entspricht dem Register PORTB, Bit 4.

Aufgabe 2 (2 Punkte):

Entwickeln Sie eine Funktion “setPin12Asm(boolean high)”. Diese soll Pin 12 auf 1 oder 0 setzen und dabei Assembler-Code verwenden. Hinweis: Sie müssen keinen Assembler für das erkennen des Branches verwenden (if). Nur das schreiben des Pins selbst muss in Assembler erfolgen.

Aufgabe 3 (3 Punkte):

Vergleichen Sie Ihre Implementierung von Aufgabe 1 und 2 indem Sie messen wie lange es dauert Pin 12 mit den Funktionen jeweils 100,000$\times$ abwechselnd auf 0 und 1 zu setzen. Speichern Sie dazu die Start- und Endzeit (“millis()”) in einer Variable und geben Sie über die serielle Schnittstelle die jeweils gemessene Zeit aus.
Schreiben Sie einen kurzen Kommentar mit Ihren Ergebnissen in die Datei.

Aufgabe 4 (4 Punkte):

Entwerfen Sie eine Funktion die mit Hilfe von Timer 2 und eines Interrupts an Pin 12 ein Signal mit einer Frequenz von 100Ḣz erzeugt.[1] Verwenden Sie wie in der Vorlesung den Timer im CTC Modus. Testen Sie Ihre Implementierung indem Sie den Pin mit dem Piepser verbinden.

[1] Es sollten also 200 Wechsel pro Sekunde erfolgen damit das Signal an Pin 12 eine Frequenz von 100 Hz hat.

Aufgabe 5 (4 Punkte):

Schreiben Sie eine Funktion mit einem Parameter für die gewünschte Frequenz an Pin 12. Diese soll zwischen 100 Hz und 2 kHz liegen. Die Funktion soll das Output Compare Register und möglicherweise den Prescaler ändern, aber nicht den gesamten Timer verändern.[2] Beachten Sie, dass es sich um einen 8 Bit Timer handelt.[3]

Aufgabe 6 (5 Punkte):

Schreiben Sie eine Funktion die Timer 1 so konfiguriert, dass er alle 1 ms einen Interrupt triggert.[4] Führen Sie eine neue globale Variable "volatile uint32_t tCount"[5] ein, die bei jedem Interrupt (in der ISR) inkrementiert wird (Initialwert: 0). Testen Sie Ihre Implementierung indem Sie in der "loop()" Anweisung regelmäßig (z.B. 1x pro Sekunde) den aktuellen Wert von "tCount" über die serielle Schnittstelle ausgeben.
Initialisieren Sie die serielle Schnittstelle dazu einmalig im "setup()" Teil mit "Serial.begin(38400);". Jetzt können Sie über "Serial.println("text");" bzw. "Serial.println(variable);" Text bzw. den Inhalt von Variablen an die serielle Schnittstelle senden. Diese können Sie mit dem "Serial Monitor" (Button oben rechts in der Arduino IDE) empfangen und anzeigen.

Aufgabe 7 (2 Punkte):

Führen Sie einen neuen globalen Array "uint32_t duration [10]" ein. Füllen Sie das Array mit 10 Werten zwischen 500 und 2500. Führen Sie außerdem neue globale Variablen "volatile uint32_t sCount" und "volatile uint8_t index" ein (jeweils initialisiert mit 0).
Erweitern Sie die ISR aus Aufgabe 6, so dass Sie folgende Funktionalität hat:
Wenn $tCount \geqslant sCount$ wird $index$ inkrementiert und $sCount = tCount + duration[index]$. Beachten Sie dabei, dass $index$ maximal den Wert 9 haben darf bevor er wieder auf 0 gesetzt werden muss. Toggeln Sie außerdem Pin 11 jedes mal wenn sich der $index$ ändert. Verbinden Sie diesen Pin mit einer LED um Ihre Implementierung zu prüfen. Herzlichen Glückwunsch! Sie können jetzt Pins zu gewählten Zeitpunkten im Hintergrund an und abschalten!

Aufgabe 8 (2 Punkte)

Führen Sie einen neuen globalen Array "uint32_t frequency [10]" ein und füllen Sie es mit Werten zwischen 100 und 2000. Wenn Ihre ISR aus Aufgabe 6/7 den $index$ inkrementiert soll die Frequenz von Pin 12 (Aufgabe 5) auf den in $frequency[index]$ gespeicherten Wert gesetzt werden. Herzlichen Glückwunsch! Sie können jetzt im Hintergrund eine Melodie abspielen während Ihr Hauptprogramm läuft.

Abgabe:

Archivieren Sie Ihr Programm in einer ZIP-Datei und laden Sie diese im Übungsportal hoch. Ihre Abgabe sollte 2 Programm mit einer Lösung für Aufgaben 1-3 und 4-8 enthalten. Bewertet werden Ihre Implementationen von Aufgabe 1 - 8. Überprüfen Sie die Archiv-Datei auf Vollständigkeit.

[2] Setzen Sie die initialen Timer Einstellungen im "setup()" Teil Ihres Programms.
[3] OCR2A und OCR2B können also maximal einen Wert von 255 enthalten.
[4] Achtung: Wählen Sie den CTC Modus mit TOP = OCR1A, nicht den mit TOP = ICR1!
[5] Das Keyword $volatile$ wird benötigt um dem Compiler mitzuteilen, dass sich der Wert von "tCount" auch außerhalb des "normalen" Programmablaufs ändern kann.