Versuch 6 – Automat

In diesem Versuch wird für einen Reaktionstester ein kleiner Automat entworfen und dann in C programmiert. Gemessen wird die Zeit vom Aufleuchten einer LED bis zum Tastendruck in Millisekunden. Dazu ist dieselbe Hardware wie in V4 (Multiplexanzeige) nötig. Der Versuch kann vor oder nach V5 (Timer/Counter) durchgeführt werden, da dafür selbst kein Timer eingestellt werden muss.

1 Entwurf des Automaten

Zunächst sei der Ablauf für die Bedienung gegeben (Ablauf 1):

Normalerweise befindet sich der Reaktionstester in Ruhe. Die *LED* ist dabei an und auf dem Display erscheint der Schriftzug *Idle*. Der Automat bleibt solange in Ruhe, bis die Taste *K2* gedrückt wird. Nach dem Tastendruck wartet der Reaktionstester zunächst einmal eine Sekunde. Dabei sind *LED* und Anzeige ausgeschaltet. Nach der ersten Sekunde wartet der Automat eine zufällig eingestellte Zeit zwischen 4 und 6 Sekunden weiter. *LED* und Anzeige bleiben ausgeschaltet. Wird während dieser Zeit *K2* gedrückt, dann geht der Reaktionstester in einen Fehlerzustand, in dem die *LED* und Anzeige wieder eingeschaltet werden. Die Anzeige zeigt dann *Err.1* an. Wenn die zweite (zufällige) Wartezeit ohne Druck auf *K2* abgelaufen ist, dann beginnt die Messung der Reaktionszeit. In diesem Zustand ist die *LED* eingeschaltet, das Display bleibt aus. Die Zeit wird dabei in Millisekunden mitgezählt. Der Startwert für die Zeit beim Eintritt in den Messzustand ist 0. Falls die Taste *K2* nun nicht innerhalb einer Sekunde gedrückt wird, dann geht der Reaktionstester in einen Fehlerzustand, in dem die *LED* und Anzeige wieder eingeschaltet werden. Die Anzeige zeigt dann *Err.2* an. Wird die Taste innerhalb einer Sekunde gedrückt, dann wechselt der Reaktionstester in die Ergebnisanzeige. Die *LED* geht aus, die Anzeige wird eingeschaltet und die im Messzustand mitgezählte Zeit (das ist die Reaktionszeit) wird angezeigt. In den Fehlerzuständen als auch bei der Ergebnisanzeige bleibt der Reaktionstester, bis die Taste *K1* gedrückt wird. In dem Fall wechselt er wieder in den Ruhezustand

Ablauf 1: Bedienung Reaktionstester

Sie können sich zur Illustration des Ablaufs das Testprogramm mittels Bootloader auf den µC laden. Die Taste *K1* ist die Boot-Taste und die Taste *K2* die zweite Taste.

Für die Signale nehmen Sie die Symbole *K1*, *K2*, *L* und *A* nach Tabelle 1.

K1	Taste 1
0	Nicht gedrückt
1	Gedrückt

K2	Taste 2
0	Nicht gedrückt
1	Gedrückt

L	LED
0	Aus
1	An

A	Anzeige
0	Aus
1	An

Tabelle 1: Signale

Welche der Signale sind Eingänge, welche sind Ausgänge?

An sich brauchen Sie eine Uhr, um die Zeit zu messen. Da der Automat mit 1000 *Hz* laufen soll, kann man aber auch ganz gut **in den Zuständen mitzählen**, seit wie vielen Automatentakten sich der Automat schon in einem Zustand befindet und kann damit die hier erforderlichen Zeiten im Bereich von 0 bis 6000 *ms* bestimmen.

Dazu können Sie im Zustandsübergangsdiagramm eine Variable z benutzen, die sowohl als Ausgang als auch Eingang benutzt werden kann. Als Ausgang können Sie die Variable **in einem Zustand** setzen (z.B.: $z = 1000, z = zufall$) oder ändern (z.B.: $z = z - 1$). Als Eingang können Sie die Variable **als Bedingung an einem Übergang** benutzen (z.B.: *wenn* $z > 1000$ *dann*). Später im Programm ist das dann ebenfalls eine Variable, z.B.: *uint16_t z*. Abbildung 1 illustriert die Verwendung an einem Beispiel.

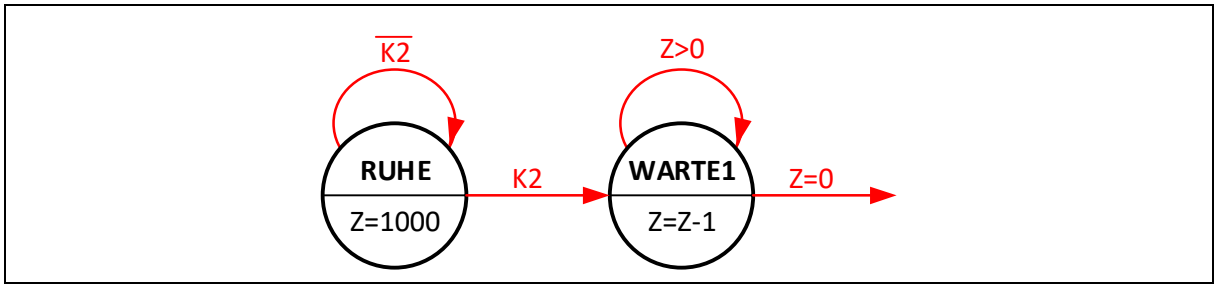

Abbildung 1: Variable in einem Zustandsübergangsdiagramm

Zeichnen Sie jetzt ein Zustandsübergangsdiagramm. Geben Sie die Aktionen oder Ausgaben am besten in einer Ausgabetabelle (Tabelle 2) an. Rechnen Sie mit ca. 8 Zuständen, je nach Vorgehen können Sie auch mehr brauchen – es gibt viele Lösungen.

Zustand	z			Anzeigetext
RUHE	z=1000			Idle
WARTE1	z=z-1			-[1]

Tabelle 2: Ausgabetabelle

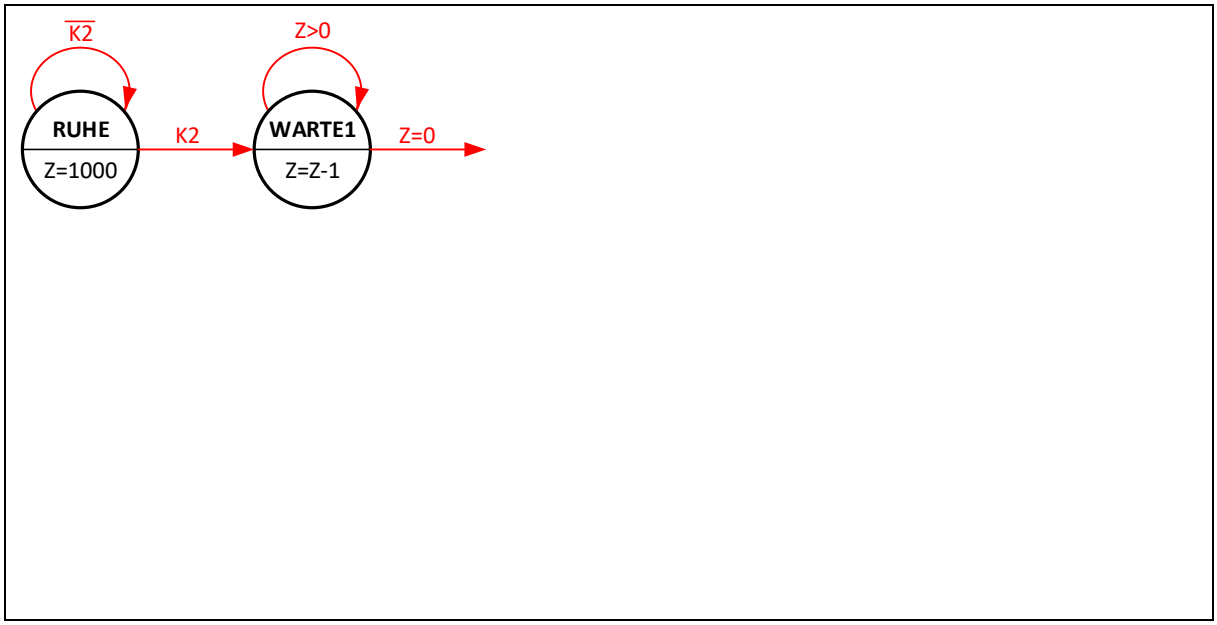

Abbildung 2: Zustandsübergangsdiagramm

[1] - bedeutet „don't care", also egal

2 Programmierung des Automaten

Wechseln Sie in einen neuen, leeren Workspace. Importieren Sie dann aus dem Archiv *v6_vorbereitung.zip* alle drei Projekte *lpc_chip_11uxx_lib*, *mch_aux_lib* und *v6*.

Der Versuch baut auf V4 auf, da die Multiplexanzeige benötigt wird. Damit das Programm übersichtlicher wird, sind die Anzeigefunktionen alle in die Bibliothek *mch_aux_lib* ausgelagert. Es gibt vier Funktionen, die Sie gut verwenden können:

1. *void mch_display_mux(void);*
   Macht die eigentliche Arbeit (Multiplexen), sollte also oft genug aufgerufen werden. Deswegen wird die Funktion auch schon fest in der Timer-ISR aufgerufen.

2. *void mch_display_set(uint8_t *p);*
   Bestimmt, was auf der Anzeige dargestellt wird. Der Parameter *p* ist die Adresse eines Feldes mit 4 Mustern. Jedes Muster benötigt einen 8 Bit Integer.

3. *void mch_display_show(bool s);*
   Bestimmt mittels des Parameters *s*, ob die Anzeige an (*s=true)* oder aus (*s=false*) ist.

4. *void mch_7seg_print_int(uint32_t x, uint8_t* buf, uint8_t n, uint8_t base, bool lz);*

   Das ist eine Art *printf("%d",x)*, nur eben für diese Anzeige.
   x: Integer, der angezeigt werden soll
   buf: Zeiger auf das Feld, in dem das passende Muster (siehe *mch_display_set()*) abgelegt wird.
   n: Anzahl der Stellen, die gedruckt werden. Darf hier maximal 4 sein, da die Anzeige nur 4 Stellen hat. Das Feld, in das gedruckt wird, muss natürlich auch mindestens n Elemente haben.
   base: Zahlenbasis, z.B. 10 für Dezimaldarstellung
   lz: Bestimmt, ob führende Nullen gedruckt werden (*lz=true*) oder nicht (*lz=false*)

Beispiel:

```
// Feld für die Anzeigedaten anlegen
uin8t_t anzeigedaten[4];

// x in das Feld drucken (4 Stellen, dezimal, führende Nullen anzeigen)
mch_7seg_print_int(x, anzeigedaten, 4, 10, true);

// Der Anzeige sagen, wo die Muster stehen
mch_display_set(anzeigedaten);

// Die Anzeige einschalten
mch_display_show(true);
```

In *v6.h* ist schon ein Aufzählungstyp für die Zustände angefangen, da können Sie Ihre weiteren Zustände hinzufügen.

Versuch 6 – Automat

Der gesamte Automat soll in der Funktion vo*id v6_reaktionstester(void)* programmiert werden. Auch diese Funktion existiert schon, sie wird ebenfalls in der Timer-ISR jede Millisekunde einmal aufgerufen.
An einer Stelle benötigen Sie eine Zufallszahl (hier im Bereich [0; 2000]. Echte Zufallszahlen sind schwer zu erzeugen. Für viele Anwendungsfälle genügt es aber, wenn der Mensch keine Regelmäßigkeiten erkennen kann. Hier steht Ihnen die Funktion *SysTick_Read()* zur Verfügung. Sie liefert den Zählerstand des Systick-Timers. Da der Timer sehr schnell läuft (48 MHz), kann man davon ausgehen, dass der Zählerstand bei Abfrage in unregelmäßigen Abständen zumindest in den unteren (niederwertigen) Stellen eine „ausreichend zufällige" Zahl[2] liefert.
Für das Intervall [0; 2000] kann man recht gut und schnell einfach die unteren 11 Bits nehmen, da 2^{11}=2048. Man bekommt dann Zahlen im Bereich [0; 2047] und das ist für die zufällige Wartezeit gut genug. In C sieht das dann so aus: *zufallszeit=SysTick_Read() & 0x7ff;*
(Das ist nichts anderes als das Kochrezept 2 – zwinge manche Stellen auf 0 und lasse andere unverändert.)

Programmieren Sie jetzt den Automaten, den Sie entworfen haben, in dieser Funktion. Gehen Sie am besten nach Schema F aus der Vorlesung vor – Eleganz kann später kommen!

Lernziele

- Automatenentwurf
- Umsetzung in einer ISR

Material zur Vorbereitung

- Projektarchiv v6_vorbereitung.zip für MCUXpresso

Material zur Durchführung

- Basisplatine, Piggyback weiß, Flachbandkabel, LED-Display, Netzteil
- Entwicklungswerkzeug (hier MCUXpresso Version 10.0.0)

[2] Besser wäre ein unabhängiger Timer, da der Systick-Timer auch die ISR selbst aufruft. Das könnte dem Zufall ein Ende bereiten, ist hier aber hinnehmbar.