1 Festlegung der Taktfrequenz

Für jeden Automaten kann prinzipiell eine eigene, der Aufgabe am besten angepasste, Taktfrequenz gewählt werden. Für viele Anwendungen in Kleingeräten eignen sich Taktfrequenzen im Bereich zwischen 50 Hz und 2 kHz.

1.1 Obere Grenze

Die Taktfrequenz wird dabei nach oben von der Leistungsfähigkeit des µC begrenzt und nach unten von der erforderlichen Reaktionsgeschwindigkeit.
Das Programm muss in der Lage sein, innerhalb eines Taktzyklus der ISR (d.h. zwischen zwei aufeinanderfolgenden Aufrufen der ISR) jeweils einen Zustandsübergang abzuarbeiten. Es ist zwar durchaus möglich, den nächsten Aufruf der ISR dynamisch zu verschieben (z.B. indem der Timer erst nach der Abarbeitung der ISR erneut gestartet wird). Allerdings verliert man damit die gewünschte Regelmäßigkeit der Zustandsübergänge und die Echtzeitfähigkeit des Systems wird zumindest in Frage gestellt.
Damit ein µC problemlos mehrere Automaten betreiben kann, sollten die erforderlichen Berechnungen für einen einzelnen Zustandsübergang einfach bleiben. Im Zweifelsfall ist es besser, für eine Sequenz von Aktionen mehrere Zustände nacheinander zu durchlaufen, als in einem Zustand eine längere Sequenz von Ausgaben zu durchlaufen.

1.2 Untere Grenze

Da der Automat die Eingänge nur mit der Taktfrequenz (der ISR) abtastet, kann eine zu niedrig gewählte Taktfrequenz im Extremfall dazu führen, dass eine Eingabe vollständig übersehen wird. Bei einer Taktfrequenz von 1 Hz muss ein Eingangssignal im ungünstigsten Fall für 1 s gehalten werden, um sicher erkannt zu werden. Das ist für viele Anwendungen zu lang. Aus diesem Grund kann die Taktfrequenz auch nicht beliebig klein gewählt werden. Für Automaten, die auf Eingaben von einem menschlichen Bediener reagieren sollen, dürfte eine untere Grenze von ca. 20 Hz gelten. Ebenso wichtig kann bei Automaten zur Steuerung einer Maschine die maximal zulässige Zeit für eine Sequenz von Ausgaben werden. Die Zahl der dabei zu durchlaufenden Zustände bestimmt dann die untere Grenze der Taktfrequenz.

2 Umsetzung in ein Programm auf einem µC

Wenn einmal das Zustandsübergangsdiagramm und die Ausgabetabelle aufgestellt worden sind, dann ist die Umsetzung in ein Programm (z.B. in C) auf einem µC reine Formsache. Im folgenden Beispiel wird keinerlei Wert auf Optimierungen gelegt sondern das prinzipielle Vorgehen gezeigt. Das Zustandsübergangsdiagramm wird dabei durch ein Programm abgearbeitet, das aus einer ISR aufgerufen wird. Die ISR stellt dabei die Taktquelle dar und ist daher meist an einen Timer gekoppelt. Ein- und Ausgänge können entweder Variablen sein oder Signale an Ports. Variablen werden verwendet, wenn der Automat mit anderen Programmteilen, das können auch weitere Automaten sein, zusammenarbeiten soll. Ein Beispiel für zwei zusammenarbeitende Automaten wäre die Aufteilung in einen Steuerautomaten (Automat 1) und eine Uhr (Automat 2), die vom Automaten 1 für die Zeitmessung verwendet wird.

> Zu dem Kaffeeautomaten gibt es C-Projekt für den Praktikums-µC. Dort ist die nachfolgend beschriebene Umsetzung des Zustandsübergangsdiagramms beispielhaft durchgeführt. Das Projekt ist komplett und kann mit der Praktikumshardware getestet werden.

2.1 Prinzipielle Sequenz

Bei jedem Aufruf der ISR für einen reinen Moore-Automaten wird dieselbe Sequenz durchlaufen:

1. Einlesen der Eingangssignale
2. Wechsel vom aktuellen Zustand in den Folgezustand
3. Ausgabe der zum neuen Zustand passenden Signale

Diese Sequenz kann man so in einer C-Funktion abarbeiten (Abbildung 1, rechts).

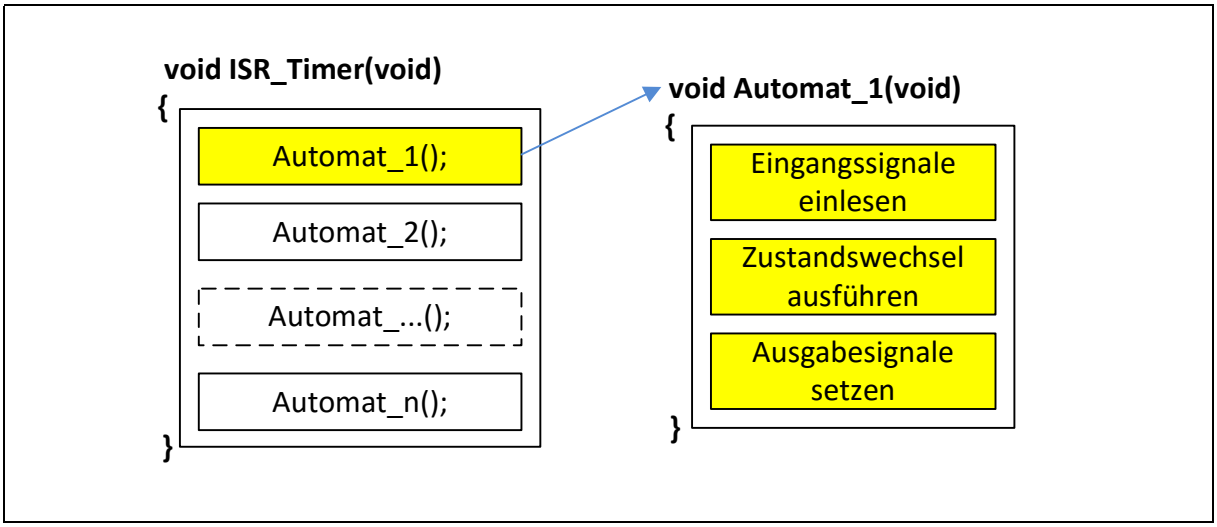

Abbildung 1: Automatenfunktionen in einer ISR

Hat man mehrere Automaten, dann benötigt man in der Regel nur eine ISR, die dann ihrerseits nacheinander die C-Funktionen für die einzelnen Automaten aufruft (Abbildung 1, links). Werden für die Automaten sehr unterschiedliche Takte benötigt (z.B. für Automat 1 50 Hz, für Automat 2 nur 2 Hz), dann stellt man den Timer auf 50 Hz und ruft in der ISR den Automaten eben nur jedes 25te Mal auf (Listing 1).

```
void RIT_IRQHandler(void)
{
  static uint8_t prescaler=0;

  // Flag löschen
  Chip_RIT_ClearInt(LPC_RITIMER);

  // Aufruf des Uhrenautomaten alle 0,5 Sekunden
  prescaler++;
  if (prescaler >= 25)
  {
    prescaler = 0;
    automat_uhr();
 }

  // Aufruf des Kaffeeautomaten alle 20ms
  automat_kaffee();
}
```

Listing 1: Automaten mit unterschiedlichen Takten (ISR des Kaffeeautomaten)

2.2 Einlesen der Eingangssignale

Zu Beginn werden alle Eingangssignale eingelesen. Ein prinzipielles Problem bei einem µC-System ist, dass Signale, die an verschiedenen Ports anliegen, nacheinander eingelesen werden müssen. Im ungünstigsten Fall könnten sich also Signale genau zwischen zwei Leseoperationen ändern und so eine Bedingung ergeben, die weder der „alten“ noch der „neuen“ Bedingung entsprechen. Daher sollte man zusammengehörige Signale am besten auf einen Port legen (dann können alle Signale zum selben Zeitpunkt gelesen werden).
In einer sehr ausführlichen Variante kann man die einzelnen Eingangssignale durch einzelne Variablen nachbilden. Das hat den Vorteil, dass man dann die Bedingungen direkt aus dem ZÜ-Diagramm in C übernehmen kann. Ist eine Variable 0, dann hat sie in C den logischen Wert falsch, alle Werte ungleich Null entsprechen dem logischen Wert wahr.

Das Einlesen der Signale zu Beginn könnte also bei dem Praktikums-µC wie folgt aussehen:

```
      // Eingangssignale Taste t und Wassersensor s
      bool t, s;

      // Einlesen der Eingangssignale
      t=!Chip_GPIO_GetPinState(LPC_GPIO_PORT, 0, 2);
      s=!Chip_GPIO_GetPinState(LPC_GPIO_PORT, 3, 0);
```

Listing 2: Einlesen der Signale (Auszug aus der Automatenfunktion)

Signale können auch intern erzeugt und verwendet werden. Ein zweiter Automat (siehe Kapitel 0 könnte beispielsweise die Zeit messen und in einer globalen Variable die Information bereitstellen, ob seit dem letzten Nullsetzen bereits 2 Stunden oder mehr vergangen sind

2.3 Wechsel vom aktuellen Zustand in den Folgezustand

In der Programmiersprache C bietet sich für die Erfassung der Zustände der Datentyp *enum* an.

```
// Typdefinition Zustandsvariable
typedef enum {RUHE_OW, RUHE_MW, DURCHLAUF, WARMHALTEN, START} COFFEE_STATES_T;
```

Listing 3: Enumerationstyp für die Zustandsvariable

Neu gegenüber dem Zustandsübergangsdiagramm ist der Zustand *START*. Damit kann unmittelbar beim Einschalten ein gesonderter Startzustand eingestellt werden. Bei einem µC-System könnten dort Einstellungen oder Prüfungen vorgenommen werden, die nur einmal erforderlich sind.

Hier bietet sich in C die Anweisung *switch/case* an. Damit kann man sofort in den derzeit aktuellen Zustand *az* springen und dann dort die Bedingungen auswerten. In den meisten Zuständen kann auch verblieben werden. Das wird durch die vorangehende Zuweisung *fz=az;* ausgedrückt.

```
// Bestimmen des Folgezustands
fz=az;
switch (az)
{
      case START:
                        if (s)      fz=RUHE_MW;
                        else        fz=RUHE_OW;
                        break;

      case RUHE_OW:
                        if (s)      fz=RUHE_MW;
                        break;

      case RUHE_MW:
                        if (!s)     fz=RUHE_OW;
                        else
                        {
                          if (t)    fz=DURCHLAUF;
                        }
                        break;
```

Listing 4: Auszug aus der Bestimmung des Folgezustands

Interessant ist hier die Möglichkeit, im *default*-Fall einen schwerwiegenden Fehler erkennen und ggf. abfangen zu können. Durch eine Störung oder einen Absturz des Programms könnte es vorkommen, dass der Automat in einen Zustand gerät, der gar nicht vorgesehen ist. Alle diese Fälle führen in den *default*-Zweig. Dort könnte eine sichere Konfiguration der Ausgänge folgen, eine Fehlermeldung ausgegeben werden und der µC neu gestartet werden. Im Beispiel geschieht hier nichts davon, der Automat würde endlos im Fehlerzustand bleiben.

2.4 Ausgabe der zum neuen Zustand passenden Signale

Nur wenn sich der Zustand geändert hat, ist eine Anpassung der Ausgangssignale nötig. Dieser Teil ist hier nur aus systematischen Gründen explizit getrennt geschrieben, er könnte natürlich auch in die vorangegangene Zustandsauswertung integriert werden (Listing 5).

```
// Falls Zustand unverändert -> fertig
if (fz==az) return;

// Sonst: In den Folgezustand gehen und Ausgänge passend setzen
az=fz;
switch (az)
{
  case RUHE_OW:
Chip_GPIO_SetPinState(LPC_GPIO_PORT, 5,  2, FALSE); // Kannenheizung aus
Chip_GPIO_SetPinState(LPC_GPIO_PORT, 1, 12, FALSE); // Wasserheizung aus
Chip_GPIO_SetPinState(LPC_GPIO_PORT, 5,  4, FALSE); // Anzeige(0) aus
Chip_GPIO_SetPinState(LPC_GPIO_PORT, 5,  3, FALSE); // Anzeige(1) aus
  break;
```

Listing 5: Auszug aus dem Setzen der Ausgabesignale

2.5 Uhrenautomat

Für das Beispiel Kaffeemaschine wird noch ein zweiter Automat, die Uhr benötigt (Listing 6)

```
void automat_uhr(void)
{
  static uint8_t prescaler=0;
  static uint32_t time=0;

  // LED zur Kontrolle umschalten (ergibt eine Blinkfrequenz von  1 Hz)
  Chip_GPIO_SetPinToggle(LPC_GPIO_PORT, 0, 8);

  // Nullstellen der Uhr prüfen
  if (kaffee_uhr.reset)
  {
    time=0;
    kaffee_uhr.timeout=FALSE;
    return;
  }

  // Uhr zählt Sekunden, also nur jedes zweite Mal bei Takt 2 Hz
  if (prescaler < 1)
  {
    prescaler++;
    return;
  }
  prescaler=0;

  // weniger TIMEOUT vergangen: nur Zeit aktualisieren
  if (time < COFFEE_TIMEOUT)
  {
    time++;
    return;
  }

  // 2h seit Nullstellen -> Ausgangsignal setzen, Zeit bleibt stehen
  kaffee_uhr.timeout=TRUE;
}
```

Listing 6: Uhrenautomat

Die Ein-/Ausgabesignale des Uhrenautomaten sind nur globale Variablen (Listing 7). Diese Signale erscheinen also nicht nach außen.

```
// Struktur für die Signale der Uhr
typedef struct
{
  bool      reset;      // Signal NULL (Eingang)
  bool      timeout;    // Signal 2h (Ausgang)
} COFFEE_CLOCK_T;

// Signale des Uhrenautomaten (globale Variable)
volatile COFFEE_CLOCK_T kaffee_uhr= { TRUE, FALSE};
```

Listing 7: Interne Signale durch Variablen (hier: Uhrensignale)