– Lösungen –
12. Mikrocontroller

- Elektronik für Informatiker -

von den Grundlagen bis zur Mikrocontroller-Applikation

Manfred Rost Sandro Wefel

23. November 2021

https://doi.org/10.1515/9783110609066

Verlag: De Gruyter Oldenbourg

Anmerkung: Bildnummern und Seitenzahlen beziehen sich auf die 2. Auflage des Buches.

12 Mikrocontroller

12.1 Jeder Rechner besitzt folgende Komponenten:

- eine CPU mit einem oder mehreren Kernen
- einen Halbleiter-Arbeitsspeicher
- einen Langzeitspeicher (Flash, HDD, SDD, etc.) für Programme und Daten
- I/O-Geräte und Schnittstellen
- ein Verbindungssystem zwischen den einzelnen Komponenten für die Datenübertragung und für die Energieversorgung

12.2 Mikroprozessoren für PC-Anwendungen (CPUs) sind für den universellen Einsatz konzipiert und enthalten eine oder mehrere Rechenkerne und den L1-Cache. Moderne Prozessoren umfassen auch den L2-Cache und einige Baugruppen, die für den Einsatz im PC-Umfeld von Interesse sind, z.B. Krypto- oder Grafik-Coprozessoren (GPUs). Die Entwicklung geht zwar in Richtung stromsparender MPUs, vordergründig sind meist Geschwindigkeit, Durchsatz und Funktionalität.

µCs für Steuer- oder Messaufgaben enthalten Spezialhardware für den jeweiligen Einsatzzweck, die in PC-CPUs nicht zu finden sind. Dazu gehören u.a. ADUs, DAUs und Portbausteine an der Schnittstelle zur Außenwelt. Die µCs werden für geringe Stromaufnahme bei kleiner bis zu moderater Rechenleistung für einen Dauereinsatz konzipiert.

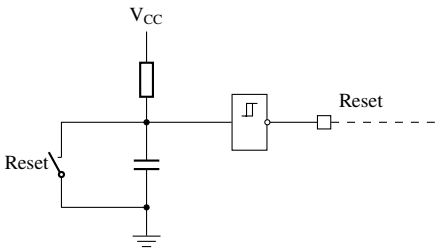
12.3 „Uhrenquarze“ sind auf eine Frequenz von 32 768 Hz konzipiert. Dies entspricht einem Wert von 2^{15} Hz. Durch einen einfachen 2^{15}-Teiler, der durch Binärteiler einfach realisiert werden kann, lässt sich der Sekundentakt ermitteln. Siehe Abb. 8.65.

12.4 Die Halbleiterspeicher des Mikrocontrollers, das umfasst die Register und den RAM, enthalten beim Einschalten einen zufälligen Wert. Die Funktionen des Mikrocontrollers werden über Register gesteuert. Um ein definiertes Verhalten des Controllers zu gewährleisten, müssen die Steuerregister und damit die zugehörigen Baugruppen in einen definierten Ausgangszustand versetzt werden, z.B. indem man alle Registerinhalte auf 0 setzt. Außerdem kann es bei langsam ansteigender Betriebsspannung zu einem undefinierten Verhalten kommen, da bestimmte Baugruppen bereits bei geringer, andere erst bei höhere

Spannung ihren Betrieb aufnehmen. Für die deterministische Programmbearbeitung muss das Rücksetzen der Register und der verzögerte Start der Programmabarbeitung durch eine Hardware-Zusatzschaltung sicher gestellt werden.

Diese Aufgabe übernimmt die Resetschaltung, welche beim Anlegen der Spannung ein Resetsignal erzeugt und über die Resetleitung an die Baugruppen weiterleitet. Das Signal wird bis zum Erreichen der Betriebsspannung und einen gewissen Zeitraum darüber hinaus angelegt. Dieser Zeitraum muss derart bemessen sein, dass alle kombinatorischen und sequentiellen Baugruppen ihren Initialzustand einnehmen können.

Eine mögliche Hardware besteht aus einem RC-Glied in Verbindung mit einem Schwellwertschalter:



12.5 Anhand der Abbildung oder aus der Beschreibung im Users's Guide entnimmt man folgende Konfiguration:

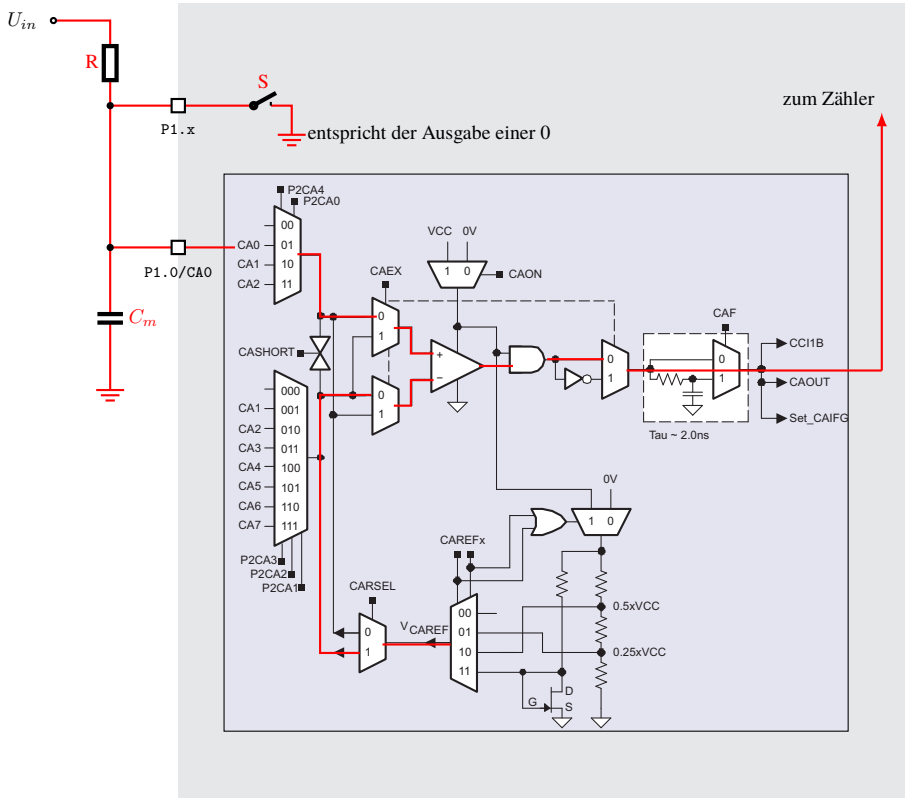
- `CONSEQ=10` *Repeat-single-channel*; der ausgewählte Eingang wird wiederholt abgetastet und über
- `CSTARTADD=0` wird Register `ADC12MCTL0` für die Kontrolle der Wandlung ausgewählt; die folgenden Werte für `SREFx` und `INCHx` müssen in diesem Register konfiguriert werden; die gemessenen Werte werden im Speicher `ADC12MEM` immer an Position 0 abgelegt und können von dort ausgelesen werden
- `ADC12SSEL=01` wählt ACLK als Taktquelle aus (`ADC12SSEL_1`)
- `ADC12DIV=8` stellt den Teiler des SA-Registertakts auf 8 (`ADC12DIV_7`)
- `REFON=1` aktiviert die Referenzspannungsquelle und
- `REF2_5V=0` setzt diese auf 1,5 V
- `REFON=1` aktiviert die Referenzspannungsquelle V_{REF+} und
- `SREF0=1, SREF1=0, SREF2=0` wählt als Referenzspannung $V_{R+} = V_{REF+}$ und für $V_{R-} = V_{SS}$ (Masse) aus (`SREF_1`)
- `INCH=0010` wählt `A2` für die Signalquelle (`INCH_2`)
- `SHS=10` wählt `TB0` für das *Sample-and-Hold*-Signal (`SHS_2`)

- ISSH=0 es wird die steigende Flanke genutzt
- die *Sample-and-Hold* Zeit soll, beginnend mit der Flanke über die H-Periode von TB0 bestimmt werden; über SHP = 0 wird TB0 direkt als Zeitquelle durchgestellt
- MSC=0 die nächste Abtastung erfolgt mit der nächsten Flanke von SHI, entsprichtTB0

12.6 Die Wandlung mit Komparator erfolgt entweder nach dem Prinzip des Wägeverfahrens (Abb. 11.17 in Kapitel 11.4.2) oder mit einem zusätzlichen Timer nach dem Zählverfahren (Abb. 11.18 und Abb. 11.19 in Kapitel 11.4.3). Für das Wägeverfahren wird ein DAC benötigt, für das Zählverfahren ein RC-Glied z.B. der Integrator und ein Timer, der als Zähler (*Counter*) betrieben wird. Der Integrator selber benötigt einen Kondensator, welcher üblicherweise extern an einem (analogen) MSP430 I/O-Port mit Verbindung zum internen Komparator angeschlossen werden muss.

Betrachten wir ein Modell der Familie MSP430F20x1 (siehe MSP430x2xx Family Users's Guide). Dieser Controller besitzt ein Comparator_+- und ein Watchdog-Modul sowie Timer_A2, jedoch keinen eigenen ADC bzw. DAC, den man für das Wägeverfahren nutzen könnte. Somit bleibt nur die Wandlung per Zählverfahren. Nach Abb. 11.18 wird neben dem Komparator noch ein weiterer OPV für den Integrator benötigt. Durch ein externes RC-Glied, dessen Ladung über den Controller gesteuert wird, können wir den zu messenden Wert über Ladekurve eines Kondensators durch Zeitmessung mit Hilfe des Komparators ohne den Integrator bestimmen.

Einen möglichen Aufbau skizziert die folgende Grafik:



Der Schalter S wird durch den Port `P1.x` realisiert, der entweder als Eingang (hochohmig ≡ Schalter geöffnet) oder als Ausgang (niederohmig ≡ Schalter geschlossen) angesteuert wird. Bei geschlossenem Schalter liegt der Ausgang auf dem Masse-Potential (üblicherweise durch Ausgabe der 0). Wenn der Schalter geschlossen ist, wird der Kondensator sofort entladen. und bei geöffnetem Schalter über den Widerstand mit der zu messenden Spannung geladen. Der Widerstand R und der Kondensator C_m bestimmten eine Ladekurve, bei der ein bestimmter Schwellwert zeitlich abhängig von U_{in} erreicht wird.

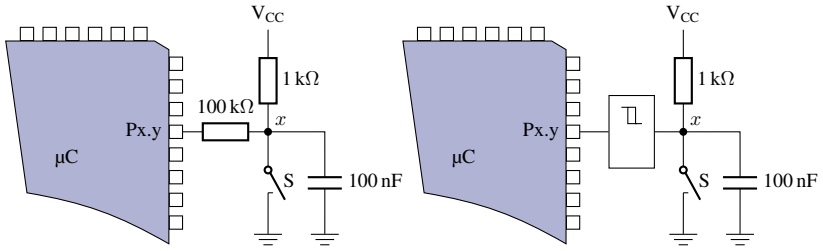
Der in der Grafik blau hinterlegte `Comparator_+` wird so konfiguriert, dass der nichtinvertierenden Eingang des OPV über Port `P1.0` mit dem Kondensator verbunden ist. Als Vergleichsspannung wird über die interne Referenzspannungsquelle ein Wert (das kann z.B. der Wert 0,5·V_{CC} sein) an den invertierenden Eingang angelegt. Der Komparator schaltet bei Erreichen der Referenzspannung den Ausgang um. Dies bedingt, dass U_{in} höher als die Referenzspannung ist. Die Umschaltung wird zur Steuerung des Timers genutzt.

Die Messung läuft wie folgt ab. Zuerst wird der Schalter geschlossen und damit der Kondensator entladen. Nach vollständiger Entladung werden zeitgleich der Schalter geöffnet und der Zähler gestartet. Damit beginnt die Ladung des Kondensators. Bei Erreichen des

Schwellwertes wird der Zähler gestoppt. Der Zählerwert entspricht der verstrichenen Zeit bis zum Erreichen des Schwellwertes. Diese Zeit ist bei festem R und C_m nur abhängig von U_{in}. Aus der Ladekurve kann jetzt die Eingangsspannung errechnet werden.

Dieser Aufbau hat einige Nachteile gegenüber dem Verfahren mit dem Integrator aus Abb. 11.18, bei dem die Ladekurve nicht ausgewertet wird und insbesondere gegenüber dem Zweiflanken-Verfahren (Abb. 11.19), wo die Werte von R und C_m nicht in die Berechnung einfließen. Allerdings erspart man sich den zweiten OPV, für den kein Modul im MSP430F20x1 vorhanden ist.

12.7 Die Entprellung (*engl. debounce*) kann durch einen vorgeschalteten Tiefpassfilter erfolgen, der sich durch ein RC-Glied entsprechend der folgenden Grafik aufbauen lässt.



Die Ladekurve des Kondensators führt beim Öffnen des Tasters zu einem zeitlich verzögerten Ansteigen der Spannung am Eingang des µC. Zur Verbesserung der Flankensteilheit kann ein Schmitt-Trigger Gatter hinzu geschaltet werden. In der Portschaltung des MSP430 ist ein Schmitt-Trigger enthalten (Abb. 12.21). Man benötigt somit keine zusätzliche Hardware, abgesehen von einem Kondensator.

Alternativ kann ein RS-FF (Abb. 8.20) verwendet werden, das über einen Umschalter am Set- und Reset-Eingang wechselseitig auf Masse geschaltet wird. Über entsprechende Pull-Up-Widerstände werden während des Umschaltens und somit während des Prellens die Eingänge auf einen festen Pegel gehalten. Der Hardwareaufwand ist allerdings beträchtlich, da man pro Taste ein FF, einen Umschalter statt eines Tasters und zwei Widerstände benötigt.

Mit einem Mikrocontroller kann die Entprellung auch in Software erfolgen, vorausgesetzt die erforderlichen Module, z.B. Timer und entsprechender Programmspeicherplatz sind vorhanden.

Für die Softwareentprellung existieren mehrere Verfahren. Man kann z.B. zur Abfrage des Tastenzustands den Tastenport so oft nacheinander auslesen, bis der Wert für eine bestimmte Anzahl von Abfragen stabil ist. Dabei handelt es sich um einen Polling-Betrieb, was die CPU von anderen Aufgaben abhält.

Alternativ kann man den Zähler auch in einen Timer-Interrupt einbauen, der ebenfalls die Portabfrage integriert und den Zustand in einer globale Variable ablegt, welche vom Programm jederzeit abgefragt werden kann.

Eine andere simple Variante sperrt bei Erkennen eines Tastendrucks im Interrupt den Port für einen gewissen Zeitraum und wertet den folgenden Zustand aus. Die Zeit hängt vom Aufbau ab und muss durch Probieren ermittelt werden.

Darüber hinaus existieren weitere Varianten und wir möchten an dieser Stelle auf die Literatur verweisen[1].

Wir zeigen hier die einfach zu verstehende Lösung mit einem Port-Interrupts, welcher den Tastendruck detektiert und dabei weitere Port-Interrupts verbietet. Außerdem wird der Watchdog Timer aktiviert, der interruptgesteuert nach einer gewissen Zeit den Port-Interrupt wieder frei gibt.

```
#include "io430.h"
#define BUTTON1 BIT0
#define BUTTON2 BIT1
//
// Function Declarations
//
void initPortPins(void);

void main(void)
{
  // falls nicht der Watchdog-Timer genutzt wird, ist folgende Zeile
  // erforderlich
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  // Einstellungen Oszillator, äLastkapazitt üfr Quarz aktivieren
  FLL_CTL0 |= XCAP18PF;

  // Ports initialisieren
  initPortPins();

  // die folgende Schleife wird bei einem Timer-Interrupt genau einmal
  // durchlaufen -
  for(;;)
  {
    // gehe sofort wieder in LPM3 und
    // warten auf naechsten Interrupt
    __bis_SR_register(LPM3_bits + GIE);

    // hier koennen weitere Aktionen erfolgen
  }

}

//
// LED und Button Port Pins + Interrupt
```

[1] Jack G. Ganssle, A Guide to Debouncing http://www.eng.utah.edu/ cs5780/debouncing.pdf

```
36 //
   void initPortPins(void)
38 {
     // LED Ports
40   P2DIR = PIN2+PIN1;                          // P2.2,1 ist Ausgang (=1)
     P5DIR = PIN1;                               // P5.1 ist Ausgang (=1)
42   P2OUT = PIN1;                               // P2.1 ist an (=1)

44   // Eingabe Ports              // Initialisierung (nicht notwendig)
     P1DIR &= ~PIN0;
46   P1DIR &= ~PIN1;

48   // Interrupt auf P1.0 + P1.1
     P1IE |= (BUTTON1 + BUTTON2);          // P1.0, P1.1 Interrupt aktiviert
50   P1IFG &= ~(BUTTON1 + BUTTON2);        // P1.0, P1.1 IFG cleared
   }
52


54 // Port1 Interrupt Service Routine
   #pragma vector=PORT1_VECTOR
56 __interrupt void Port_1(void)
   {
58   // Taste verarbeiten
       if(P1IFG & BUTTON1) // Interrupt von BUTTON1 (P1.0)?
60     {
         P2OUT ^= PIN2+PIN1;                  // Toggle P2.2,1
62     } else {            // sonst ist er von BUTTON2
         P5OUT ^= PIN1;                       // Toggle P5.1
64     }

66   // vorerst kein Button Interrupt erlauben (debounce)
     P1IE &= ~(BUTTON1 + BUTTON2);
68   // WDT vorbereiten
     WDTCTL = WDT_ADLY_250;  // Timer-Intervall = fACLK/8192, ca. 250ms
70   // WDT Interrupt Flag loeschen, ein neuer WDT-IRQ kann ausgeloest
         werden
     IFG1 &= ~WDTIFG;
72   // WDT erlauben, Ausloesung in ca. 250ms
     IE1 |= WDTIE;
74
     // ggf. CPU kurz aufwecken (clear LPM3 Bits)
76   __bic_SR_register_on_exit(LPM3_bits);
    }
78
   // Watchdog Timer interrupt service routine
80 // Verlassen des LPM3-Modus, Fortsetzung der CPU
   #pragma vector=WDT_VECTOR
82 __interrupt void watchdog_timer(void)
   {
84   // WDT IRQ verbieten
     IE1 &= ~WDTIE;
86   // WDT Flag loeschen
     IFG1 &= ~WDTIFG;
88   // WDT stoppen
     WDTCTL = WDTPW + WDTHOLD;
90   // Button Flags ölschen, so dass IRQ erlaubt werden kann
```

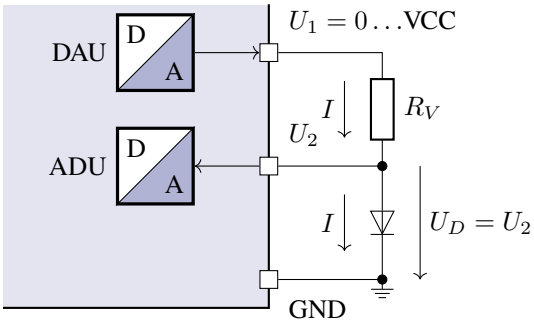
```
    P1IFG &= ~(BUTTON1 + BUTTON2);
92  // Button Interrupt erlauben
    P1IE |= (BUTTON1 + BUTTON2);
94
    // ggf. CPU kurz aufwecken (clear LPM3 Bits)
96  __bic_SR_register_on_exit(LPM3_bits);
}
```

12.8 Das Maximum an Tasten bei gleichzeitiger Minimierung der Zeilen und Spalten (entspricht der Anzahl der Anschlüsse) erreicht man bei einer quadratischen Matrix. Das entspricht der Frage, welches Rechteck den größten Flächeninhalt bei vorgegebenen Umfang hat, was bekanntlich ein Quadrat ist. Das Ziel für die Matrix ist folglich der Aufbau eines Quadrates, insofern dies mit der gegebenen Anzahl an Anschlüssen möglich ist.

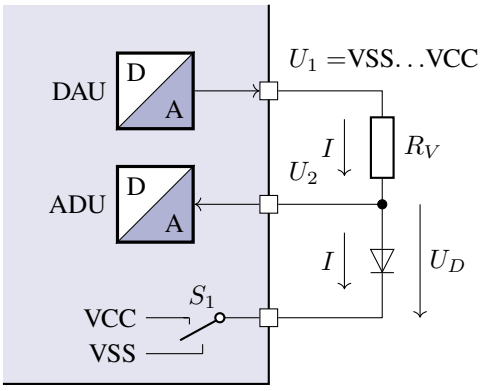
Vier Tasten erfordern ein 2x2 Quadrat und damit 4 Anschlüsse, die auch bei einer Einzelansteuerung der Tasten erforderlich sind. Mit einem 2x3-Rechteck erreicht man bereits 6 Tasten mit 5 Anschlüssen und somit einen Vorteil gegenüber der Einzelansteuerung. Mit einem zusätzlichen Anschluss wird es ein Quadrat (3x3) mit 9 Tasten. Damit vergrößert sich bei jeder weiteren Leitung die Anzahl der Tasten, die bei einer Matrix gegenüber der Einzelansteuerung erreicht werden, beginnend ab 6 Tasten.

Die Vorteile der Matrix sind der kompakte Aufbau und die vergleichsweise geringe Anzahl benötigter Anschlüsse. Die Nachteile sind die erforderliche Auswertung in Software und die Verzögerungen, die sich zwangsläufig ergeben, da nicht alle Tasten zeitgleich abgefragt werden und somit bei einem Tastendruck nicht sofort ein Interrupt ausgelöst werden kann.

12.9 Für die Aufnahme der Diodenkennlinie benötigen wir eine Möglichkeit zur Messung des Stromes, der über die Diode fließt, in Abhängigkeit von der anliegenden Spannung. Die Spannung kann über den Mikrocontroller mittels eines DA-Wandlers vorgegeben werden. Der Strom lässt sich nicht direkt messen. Wir benötigen eine Zusatzschaltung, bei der wir den Stromfluss über die Spannung bestimmen können, welche wir mittels eines AD-Wandlers vom Mikrocontroller auslesen können. Dafür benötigen wir einen Widerstand mit bekanntem Wert, so dass wir aus dem Spannungsabfall über diesen Widerstand den Strom ermitteln können. Daraus ergibt sich folgender Aufbau:



Zusätzlich benötigen wir eine Schaltung, welche die Diode in Durchlass- und in Sperrrichtung betreibt. Das können wir mit einem Mikrocontroller ohne Zusatzschaltung dadurch erreichen, dass wir als Bezugspotential nicht die Masse (GND), sondern einen Pin des Mikrocontrollers verwenden. Diesen können wir zur Laufzeit mit Masse (VSS)[1] oder Betriebsspannung (VCC) verbinden. Den Versuchsaufbau zeigt die folgende Schaltung:



Wir gehen von einem hochohmigen Eingang des ADU aus. Somit entspricht der Strom, der über die Diode fließt dem Strom, der auch über R_V fließt. Damit gilt für jeden Messpunkt i, der Strom $I(i) = \frac{U_1(i) - U_2(i)}{R_V}$.

Programmablauf:

- Wir starten in Sperrrichtung und schalten dazu S_1 auf VCC und U_1 auf VSS (0 V). Nun lassen wir U_1 schrittweise gegen VCC wachsen und messen dabei die Spannung U_2.

  Die Spannung über die Diode im i-ten Schritt ist $U_D(i) = \text{VCC} - U_2(i)$. Die Werte $U_D(i)$ und $I(i)$ legen wir für jeden Schritt i im Hauptspeicher ab.

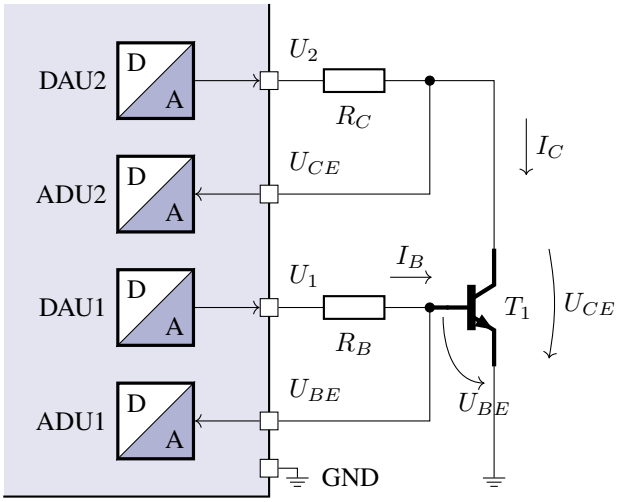
[1] Wir bezeichnen die Masse entsprechend der Notation von Texas Instruments als DVSS oder vereinfacht VSS.

- Nach Erreichen des Wertes VCC am Ausgang des ADU setzen wir die Messung in Durchlassrichtung fort. Dazu schalten wir sowohl S_1 als auch U_1 auf VSS. Anschließend lassen wir U_1 schrittweise abermals gegen VCC wachsen, wobei wir den Index i weiter hoch zählen. In jedem Schritt wird die Spannung U_2 gemessen.

  Im i-ten Schritt ist $U_D(i) = U_2(i) - \text{VSS} = U_2(i)$. Die Werte $U_D(i)$ und $I(i)$ legen wir ebenfalls im Hauptspeicher ab.

- Nach der Messung tragen wir die Werte aus dem Hauptspeicher in ein U/I-Diagramm ein und erhalten die Kennlinie der Diode.

12.10 Für die Aufnahme der Kennlinienfeldes eine Bipolartransistors benötigen wir zwei gesteuerte Spannungsquellen, für die wir zwei DA-Wandler des Mikrocontrollers verwenden. Die Messung der Spannungen U_{BE} und U_{BE} erfolgt über zwei AD-Wandler. Die Ströme I_B und I_C bestimmen wir über den Spannungsabfall, der über in Reihe zu den Spannungsquellen geschalteten Festwiderständen abfällt. Den Aufbau für eine Messung in Emitterschaltung zeigt die folgende Grafik:



Es gilt:

$$I_C = \frac{U_2 - U_{CE}}{R_C} \qquad I_B = \frac{U_1 - U_{BE}}{R_B}$$

Wir gehen dabei jeweils von einem hochohmigen Eingang der ADUs aus. Die einzelnen Quadranten des Kennlinienfeldes werde wie folgt bestimmt. Dabei läuft die Messung und Ablage der Werte im Hauptspeicher schrittweise, analog zur vorhergehenden Aufgabe.

- Ausgangskennlinienfeld $I_C = f(U_{CE})$ mit I_B als Parameter: Wir stellen eine Spannung U_1 ein und bestimmen I_B. Dann lassen wir die Spannung U_2 den Bereich VSS

bis VCC durchlaufen und bestimmen dabei U_{CE} und I_C. Dabei muss in jeder Messung U_1 nachgeregelt werden, so dass I_B konstant bleibt. Diesen Vorgang wiederholen wir für verschiedene I_B.

- Übertragungskennlinie $I_C = f(I_B)$ für ein festes U_{CE}: Wir stellen die Spannung U_2 für einen bestimmen Wert U_{CE} ein, z.B. 5 V. Anschließend lassen wir die Spannung U_1 den Bereich durchlaufen, wobei in jedem Schritt U_2 derart angepasst wird, dass U_{CE} stabil bleibt. Wir bestimmen aus U_2 und U_{CE} den Wert I_C und aus U_1 und U_{BE} den Wert I_B.
- Eingangskennlinie $U_{BE} = f(I_B)$ für ein U_{CE}: Diese Kennlinie können wir bei der Bestimmung der Übertragungskennlinie für ein bestimmtes U_{CE} (z.B.5 V) parallel ermitteln, da das gesuchte U_{BE} direkt als U_{BE} gemessen wird und somit zu dem rechnerisch aus U_1 und U_{BE} bestimmten Wert I_B abgelegt wird.
- Rückwirkungskennlinienfeld $U_{BE} = f(U_{CE})$ mit I_B als Parameter: Die Kennlinie bestimmen wir zu jedem I_B zusammen mit der Ausgangskennlinie. Wenn bei Messung der Ausgangskennlinie jeweils U_1 nachgeregelt wird, notieren wir den gemessenen Wert U_B, welcher dem gesuchten Wert U_{BE} zum jeweiligen Wert U_{CE} entspricht .

Aus den gespeicherten Werten können wir das Kennlinienfeld nach Abb. 4.25 zusammen setzen.