Themenblatt

PIC_programmieren

Einfache Codesequenzen und kleine Funktionen

Inhaltsverzeichnis

1 Die Hardware

Will man in Assembler programmieren, ist eine genaue Kenntnis der Hardware und der internen Struktur des Mikrocontrollers notwendig. Neben dem Schaltplan, der zeigt wo welche Signale am Controller angeschlossen sind muss man auch die Registerstruktur und weitere Interna des Mikrocontrollers kennen.

1.1 Der Schaltplan

Der Schaltplan (Schematic) einer einfachen Anwendung ist in Abbildung 1 zu sehen.

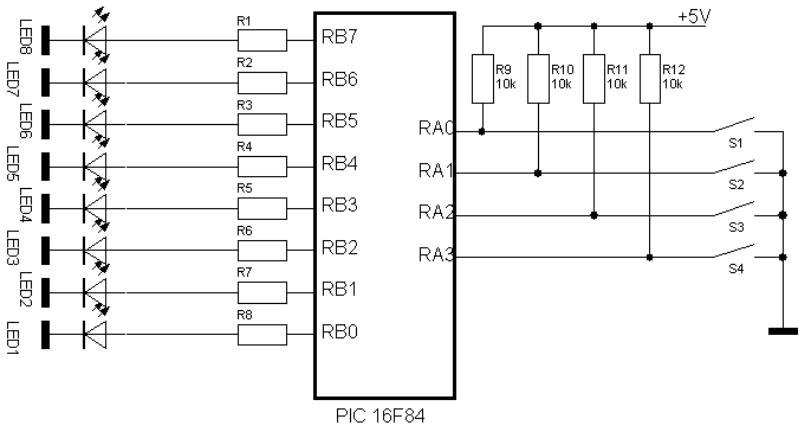


Abbildung 1: Eine einfache PIC Applikation

Die Leuchtdioden LED 1 bis LED 8 sind über strombegrenzende Widerstände am Port RB des PICs angeschlossen. Dieser Port muss später als Ausgang arbeiten, da die Leds eine Spannungsquelle benötigen. Denn arbeitet ein Anschluss (Pin) als Ausgang, fungiert er praktisch als schaltbare Spannungsquelle oder Senke.

Auf der rechten Seite sind an die Leitungen RA 0 bis RA 3 die Schalter S 1 bis S 4 angeschlossen. Wenn einer dieser Schalter geschlossen[1] wird, verbindet er den entsprechenden Anschluss mit Massepotenzial (0 V). Damit der als Eingang arbeitende Anschluss bei offenem Schalter nicht "in der Luft hängt", sind Pullup-Widerstände angeschlossen. Somit erzeugt ein offener Schalter am Eingang einen High-Pegel, ist der Schalter geschlossen einen Low-Pegel. Dies wird als Low-Active Logik bezeichnet. Dies muss bei der Programmentwicklung berücksichtigt werden.

1 Schalter und Taster werden i.d.R. in Ruhestellung gezeichnet, hier also im geöffneten Zustand

Was nicht im Schaltplan eingezeichnet, aber beim realen Projekt in jedem Fall vorhanden sein muss ist die Spannungsversorgung und ggf. der Taktgeber[2] sowie eine Resetlogik.

1.2 Die Registerstruktur des PICs

Im Gegensatz zum Programmieren in einer Hochsprache muss sich bei der Assemblerprogrammierung um fasst jedes Detail selbst kümmern. Während ein Hochsprachencompiler beispielsweise den verwendeten Variablen selbst die passende Speicheradresse zuordnet, ist dies die Aufgabe des Assemblerprogrammierers. Dazu muss er die Registerstruktur und die Adressen der speziellen Funktionsregister kennen.

Adresse	Bezeichnung	Bezeichnung	Adresse
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B	indirect TMR 0 PCL Status FSR Port RA Port RB - EEData EEAdr PCLATH INTCON	indirect Option PCL Status FSR Tris RA Tris RB - EECon1 EECon2 PCLATH PCLATH	0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B
0x0C ... 0x2F	Anwender-bereich	wird auf Bank 1 gespiegelt	0x0C ... 0x2F
0x30 ... 0x7F	nicht implementiert	nicht implementiert	0x30 ... 0x7F
Bank 0		Bank 1	

Die Bank, auf die man aktuell zugreifen kann, wird im Statusregister in dem Bit

2 Falls der Controller keinen internen Oszillator besitzt.

RP 0 festgelegt. Bei PIC Mikrocontroller mit vier statt zwei Bänken ist noch ein RP 1- und IRP-Bit vorhanden. Will man beispielsweise das TRIS RA verändern, muss man per Programm auf Bank 1 umschalten. Die notwendige Sequenz lautet:

```
BSF     status, rp0     ;umschalten auf Bank 1
MOVLW   11110011B       ;RA 0 und RA 1 sind Eingang
MOVWF   5               ;RA 2 und RA 3 sind Ausgang
BCF     status, rp0     ;zurück auf Bank 0
```

Manche Register sind sowohl auf Bank 0 als auch auf Bank 1 zu finden. Die Notwendigkeit ist beim Statusregister offensichtlich, da sonst ein zurückschalten auf Bank 0 nicht mehr möglich wäre.

1.3 Register 0 für die indirekte Adressierung

Die indirekte Adressierung erlaubt bei bestimmten Programmstrukturen ein sehr effizientes und kompaktes Programmieren. Während bei der direkten Adressierung die verwendete Speicheradresse als absoluter Wert angegeben wird, zeigt bei der indirekten Adressierung ein Zeiger auf die verwendete Speicheradresse. Dieser Zeiger steht in einem Register und ist somit während des Programmablaufs veränderbar.

Beim PIC Mikrocontroller heißt dieses Zeigerregister FSR (File Select Register) und hat die RAM-Adresse 4 (sowohl auf Bank 0, als auch auf Bank 1). Es umfasst 8 Bit und erlaubt auf diese Weise neben den Adressen auf Bank 0 auch die auf Bank 1 direkt zu adressieren.

```
MOVLW   10H     ;Zeiger auf Adresse 0x10 setzen
MOVWF   fsr
MOVLW   0AAH    ;dieser Wert wird an Adresse 10
MOVWF   indirect;abgelegt.
INCF    indirect;in 0x10 steht jetzt 0xAB
```

Die indirekte Adressierung kann auf alle Befehle, die auf eine RAM-Adresse zugreifen, verwendet werden.

1.4 TMR0 Register

Das Timerregister TMR0 kann zum einen durch äußere Impulse hochgezählt

werden. Dazu wird durch eine entsprechende Einstellung im Optionregister der Anschluss RA 4 mit dem TMR0 verbunden. Dabei kann zusätzlich auch noch ein Vorteiler[3] verwendet werden, so dass nicht jeder Impuls am Eingang gezählt wird.

Das TMR0-Register lässt sich aber auch durch den internen Befehlstakt inkrementieren.

1.5 PCL

Das PCL-Register ist der untere Teil des Programmzählers. Nur diese unteren 8 Bit sind durch arithmetische bzw. logische Befehle änderbar. Obwohl das PCL ein Teil des Programmzählers ist, wird es wie ein normales 8-Bit Register angesprochen und es verhält sich auch dementsprechend. Ein 8 Bit Überlauf wird nicht auf die nächst höheren Bits des Programmzählers übertragen, sondern kommt wie bei allen übrigen Registern ins Carryflag.

Durch die Addition einer Zahl zum PCL lässt sich somit ein berechneter Sprung ausführen. Allerdings ist der Bereich, den man auf diese Weise abdecken kann, nur 256 Adressen groß. Diese Blöcke des Adressraums beginnen immer bei den Adressen 0x0000, 0x0100, 0x0200, 0x0300 etc.

Das Arbeiten mit Konstantentabellen wird mit diesem Register erst richtig möglich. Wie solche Tabellen aufgebaut werden ist in Kapitel 7 beschrieben.

1.6 Status

Im Statusregister sind neben dem Zeroflag, Carryflag und Digitcarryflag weitere, zur Steuerung des PICs sinnvolle Umschalt- und Zustandbits enthalten. Welche Bits durch welche Befehle beeinflusst werden, ist aus der Befehlsübersicht auf Seite 10 zu sehen. Ein Zustandbit bleibt solange unverändert, bis wieder ein Befehl ausgeführt wird, der das entsprechende Flag beeinflusst. Dabei wird es gesetzt oder zurückgesetzt, je nach Ergebnis der Operation.

1.6.1 Carryflag: Bit 0

Dieses Zustandbit zeigt an, ob ein Zahlenbreichsüberlauf erfolgt ist. Dabei ist es gleichgültig, ob dieser Überlauf an oberen Ende, also bei 255 oder am unteren Ende bei 0 erfolgt. Das Übertragsbit (Carry) zeigt an, dass das Ergebnis eigent-

3 Der Vorteiler kann aber auch dem Watchdog zugeordnet werden, aber nur entweder oder.

lich falsch ist, wenn nicht dieses zusätzliche Bit als Bereichserweiterung betrachtet wird.

Beispiel: 255 + 1 = 0, C = 1 ;diese 1 entspricht dem Wert 256
Vorsicht ist geboten, wenn das Carry nach einer Subtraktion interpretiert werden soll. Mehr dazu unter 5.3 .

1.6.2 DC-Flag: Bit 1

Dieses Flag wird auch als Halfcarry bezeichnet. Es zeigt den Überlauf der unteren 4 Bits an. Da bei einer 8-Bit Operation dieser Überlauf nahtlos in die oberen 4 Bits einfließt hat er eher eine untergeordnete Bedeutung. Lediglich wenn man Routinen für die BCD-Arithmetik[4] implementiert, wird dieses Bit wichtig.

1.6.3 Zero-Flag: Bit 2

Das Zerobit wird gesetzt, wenn das Ergebnis einer Operation 0 ist. Interessant ist beim PIC Mikrocontroller, dass auch beim MOVF-Befehl das Zeroflag beeinflusst wird.

1.6.4 $\overline{\text{PD}}$-Flag: Bit 3

Dieses Power-Down-Bit wird im Zusammenhang mit dem Einschalten, dem CLRWDT- und dem SLEEP-Befehl verändert.

gesetzt auf 0	bei der Ausführung des SLEEP-Befehls
gesetzt auf 1	beim Einschalten des PICs und bei der Ausführung des CLRWDT-Befehls.

1.6.5 $\overline{\text{TO}}$-Flag: Bit 4

Bit 4 im Statusregister ist das Time-Out-Bit. Es wird im Zusammenhang mit dem Einschalten, einem CLRWDT-, einem SLEEP-Befehl oder durch das Ablaufen der Watchdogzeit beeinflusst.

gesetzt auf 0	wenn der Watchdog abgelaufen ist
gesetzt auf 1	beim Einschalten des PICs und bei der Ausführung der Befehle SLEEP und CLRWDT

1.6.6 RP0 und RP1: Bit 5 und 6

Mit diesen beiden Bits lassen sich die Registerbänke umschalten. Sind nur 2

4 BCD = Binary Coded Digit. Bei einer BCD-Zahl sind in einem Byte 2 Zahlen codiert. Jedes Halbbyte zählt dann von 0 bis 9.

dieser Bänke vorhanden, reicht das RP0-Bit. Ist es zurückgesetzt, greift man auf die Register der Bank 0 zu, bei gesetztem RP0 auf die Register er Bank 1.

RP1	RP0	
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

1.6.7 IRP: Bit 7

Dieses Bit wird nur bei PICs mit mehr als 2 Registerbänken benötigt. Will man eine Adresse auf Bank 3 indirekt adressieren, reicht die 8-Bit-Adresse im FSR-Register nicht aus. Deshalb muss das IRP zusätzlich gesetzt werden.

indirekter Zugriff auf Bank 0 und 1	IRP = 0
indirekter Zugriff auf Bank 2 und 3	IRP = 1

1.7 FSR

Dieses Register arbeitet mit dem INDIRECT-Register Hand in Hand. Die ausführliche Beschreibung ist unter 1.3 zu finden.

2 Assemblerprogrammierung

2.1 Der Befehlssatz

Betrachtet man den Befehlssatz des PIC Microcontrollers fällt die kleine Anzahl von Befehlen auf. Man rechnet diesen Mikrocontroller zu den sogenannten RISC Prozessoren. RISC steht für Reduced Instruction Set Computer. Die Befehle werden sehr schnell, meist in nur einem Befehlstakt abgearbeitet. Dafür müssen komplexere Befehle durch eine Befehlssequenz nachgebildet werden.

In der Tabelle 1 sind alle Befehle aufgelistet. Die exakte Beschreibung eines jeden einzelnen Befehls befindet sich im Anhang.

Der Buchstabe f bedeutet Fileregister und ist eine Adresse des Datenspeichers. Sie ist maximal 7 Bit breit und kann somit die Werte 0 bis 127 bzw. 0x00 bis 0x7F annehmen. Wird eine Fileregisteradresse angegeben, wird immer deren Inhalt verarbeitet bzw. verknüpft.

Das sogenannte Destinationbit d gibt an, wohin das Ergebnis der Verknüpfung abgelegt werden soll. Ist d = 0, kommt das Ergebnis ins W-Register, bei d = 1 ins Fileregister.

Bei den Bitbefehlen wird das zu verwendende Bit b im Befehl angegeben. Es kann Werte von 0 bis 7 annehmen.

Die Literale werden in der Übersicht durch den Buchstaben k gekennzeichnet. Microchip, der Hersteller des PIC Mikrocontrollers, bezeichnet Konstante als Literale. Sie können entweder 8 Bit lang sein, wenn es um Arithmetikbefehle oder logische Verknüpfungen handelt. Im Fall von Sprungbefehlen (GOTO und CALL) sind sie 11 Bit lang.

Manche Befehle beeinflussen die Flags im Statusregister. Welche dies sind, ist in der letzten Spalte aufgeführt. Ein Carryflag (C) zeigt den Über- oder Unterlauf einer 8-Bit Rechenoperation an. Hier ist aber auf die seltsame Eigenheit des PICs zu achten. Mehr dazu unter 5.3.

Das Digitcarry-Flag (DC) zeigt den Überlauf vom unteren auf das obere Halbbyte an.

Das Zeroflag (Z) wird gesetzt, wenn das Ergebnis der Operation 0 ergibt. Allerdings verändert auch der Befehl MOVF f,d dieses Flag. Mehr dazu unter Kapitel 4.

Mnemonic		Beschreibung	Befehls-takte	Flag
Byteorientierte Befehle				
ADDWF	f,d	Addiert W und Inhalt von f	1	C, DC, Z
ANDWF	f,d	UND-Verknüpfung von W und Inhalt von f	1	Z
CLRF	f	löscht den Inhalt von f	1	Z
CLRW		löscht das W-Register	1	Z
COMF	f,d	Bildet Komplement vom Inhalt von f	1	Z
DECF	f,d	Dekrementiert den Inhalt von f	1	Z
DECFSZ	f,d	Dek. f und überspringt bei 0 den nächsten Befehl	1 (2)	
INCF	f,d	Inkrementiert den Inhalt von f	1	Z
INCFSZ	f,d	Ink. f und überspringt bei 0 den nächsten Befehlen	1 (2)	
MOVF	f,d	Holt den Wert aus Adresse f	1	Z
MOVWF	f	Schreibt Inhalt von W nach Adresse f	1	
NOP		Leerbefehl	1	
RLF	f,d	Rotiert den Inhalt von f nach links durchs Carry	1	C
RRF	f,d	Rotiert den Inhalt von f nach rechts durchs Carry	1	C
SUBWF	f,d	Subtrahiert W vom Inhalt der Adresse f	1	C, DC, Z
SWAPF	f,d	Vertauscht die beiden Halbbytes in Adresse f	1	
XORWF	f,d	EXCLUSIV-ODER von W und f	1	Z
Bitorientierte Befehle				
BCF	f,b	Löscht Bit b in Adresse f	1	
BSF	f,b	Setzt Bit b in Adresse f	1	
BTFSC	f,b	Testet Bit b an Adresse f und springt, wenn es 0 ist	1 (2)	
BTFSS	f,b	Testet Bit b an Adresse f und springt wenn es 1 ist	1 (2)	
Literal und Steuerbefehle				
ADDLW	k	Addiert Literal k zum W-Register	1	C, DC, Z
ANDLW	k	Literal k wird UND verknüpft mit dem W-Register	1	Z
CALL	k	Unterprogrammaufruf an Adresse k	2	
CLRWDT		löscht den Watchdog	1	$\overline{TO}$, $\overline{PD}$
GOTO	k	Sprung zur Adresse k	2	
IORLW	k	Literal wird mit dem W-Register ODER verknüpft	1	$\overline{Z}$
MOVLW	k	Lädt das Literal ins W-Registers	1	
RETFIE		Rückkehr aus der Interruptroutine, setzte GIE	2	
RETLW	k	Rückkehr aus einem Unterprogramm, lädt k in W	2	
RETURN		Rückkehr aus einem Unterprogramm	2	
SLEEP		Geht in den Stand By Modus	1	$\overline{TO}$, $\overline{PD}$
SUBLW	k	Subtrahiert W vom Literal k	1	C, DC, Z
XORLW	k	EXCLUSIV ODER Verknüpfung von W und k	1	Z

Tabelle 1: Befehlsübersicht

In der zweitletzten Spalte werden die benötigten Befehlstakte angegeben. Üblicherweise werden die Befehle in einem Befehlstakt (= 4 Quarztakte) abgearbeitet. Bei manchen muss aber der in der Pipeline liegende Befehl gelöscht werden, was einen zusätzlichen Befehlstakt bedingt.

3 Arithmetische und logische Verknüpfungen

Bei arithmetischen und logischen Verknüpfungen sind immer zwei Argumente notwendig. Diese können entweder im Speicher stehen und sind somit während des Programmablaufs veränderbar oder einer der beiden ist ein Literal (Konstante). Um die Verknüpfung durchzuführen muss ein Argument im W-Register stehen, das andere in einer Speicherstelle oder als Literal im Befehlscode.

Erfolgt die Verknüpfung mit dem Inhalt einer Speicherstelle, kann man mittels des sogenannten Destinationbits (d) bestimmen, ob das Ergebnis der Verknüpfung in das W-Register (d = 0) oder in die Speicherstelle aus der das Argument kam (d=1) zurückgeschrieben werden soll.

4 Überprüfen einer Speicherstelle auf Null

Lade- bzw. Move-Befehle beeinflussen üblicherweise keine Flags. Eine Ausnahme bildet der MOVF-Befehl des PIC. Er beeinflusst das Zeroflag im Statusregister.

```
MOVF    adr[5]      ;Der Inhalt von adr wird gelesen und
                    ;in adr zurück geschrieben.

MOVF    adr, W[6]   ;Der Inhalt von adr wird gelesen und
                    ;ins W-Register geschrieben
```

Wird dieser Befehl ausgeführt und der Inhalt von adr Null, dann wird das Zeroflag im Statusregister (Bit 2) gesetzt. Ist der Inhalt nicht Null, wird das Zeroflag auf 0 gesetzt.

Damit lässt sich der Inhalt einer Speicherzelle auf Null prüfen, ohne dass ein anderes Register verändert wird (außer dem Statusregister).

5 Inhalt einer Speicherstelle auf einen bestimmten Wert prüfen

Will man auf Gleichheit prüfen, ist eine XOR-Befehl die erste Wahl. Denn dieser verändert im Statusregister nur das Zeroflag.

5 Gleichbedeutend mit MOVF adr,1
6 Gleichbedeutend mit MOVF adr,0

5.1 Vergleich zweier Speicherstellen

```
MOVF    adr1, W     ;ein Argument ins W-Register holen
XORWF   adr2, W     ;XOR verknüpfen und Ergebnis in
                    ;W-Register, so bleiben adr1 und
                    ;adr2 unverändert.
BTFSC   status, Zflag
GOTO    sindGleich
GOTO    sindUngleich
```

5.2 Vergleich einer Speicherstelle mit einem Literal

```
MOVLW   literal     ;das Literal kommt ins W-Register
XORWF   adr         ;das Ergebnis der XOR-Verknüp
                    ;fung kommt nach adr
BTFSC   status, Zflag
GOTO    sindGleich
GOTO    sindUngleich
```

oder:

```
MOVLW   literal     ;das Literal kommt ins W-Register
XORWF   adr,W       ;das Ergebnis der XOR-Verknüp
                    ;fung kommt ins W-Register
BTFSC   status, Zflag
GOTO    sindGleich
GOTO    sindUngleich
```

5.3 Prüfen, ob der eine Wert größer als der andere ist

In diesem Fall muss man mit dem Subtraktionsbefehl arbeiten. Allerdings hat dieser beim PIC so seine „Eigenheiten“. Das Carryflag wird anderst gesetzt, als es dem üblichen Verständnis entspricht. Wird das Ergebnis größer oder gleich Null, wird das Carryflag gesetzt, sonst zurückgesetzt.

Dieses seltsame Verhalten ist auf die Art und Weise wie der PIC Mikrocontrol-

ler die Subtraktion durchführt und auf einen Maskenfehler zurückzuführen[7]. Die Subtraktion wird durch die Addition des Zweierkomplements realisiert.

Als Beispiel dient die Aufgabe 10 – 15 = – 5. In Hexdarstellung bedeutet die 10 ein 0x00001010, eine 15 ein 0x00001111. Das Zweierkomplement davon ist 0x11110001.

```
                0x00001010
        +       0x11110001
-------------------------------------
                0x11111011          = 251
```

Dieser Binärwert entspricht dem Ergebnis – 5. Nur erfolgte bei der obigen Berechnung an keiner Stelle ein Überlauf, so dass das Carryflag nicht gesetzt ist. Und hier kommt der Maskenfehler ins Spiel. Normalerweise wird das Carry- und Digitcarry nach einer Subtraktion invertiert. Das wurde hier schlicht vergessen.

Was passiert aber, wenn der Subtrahend kleiner als der Minuend, das Ergebnis somit positiv ist? Das zeigt die folgende Aufgabe: 15 – 10 = 5.

```
                0x00001111
        +       0x11110110          Zweierkomplement
Übertrag      1   11111100
-------------------------------------
                0x00000101
```

Hier wird ein Ergebnisüberlauf erzeugt, der ins Carryflag geschrieben wird.

Als letztes interessiert uns noch, was bei der Berechnung von 15 – 15 = 0 passiert.

```
                0x00001111
        +       0x11110001
Übertrag      1   11111110
-------------------------------------
                0x00000000
```

Das Ergebnis ist wie erwartet eine 0, aber auch hier wird das Carryflag auf 1 gesetzt.

Das bedeutet: Ist der Subtrahend kleiner oder gleich dem Minuend, wird das

7 Dieser Maskenfehler wird heute nicht als Fehler, sondern als Feature „verkauft“.

Carryflag gesetzt. Ist hingegen der Subtrahend größer als der Minuend, wird das Carryflag gelöscht.

Wichtig: Der Inhalt des W-Registers wird vom Literal oder dem Inhalt der Speicherstelle abgezogen!

Literal – W bzw. Registerinhalt – W

5.4 Vergleich zweier Speicherstellen auf größer / kleiner

```
MOVF    adr1, W     ;ein Argument ins W-Register holen
SUBWF   adr2, W     ;subtrahiere W vom Inhalt von adr2
                    ;und schreib Ergebnis ins ;W-Regis
                    ;ter, so bleiben adr1 und adr2 unver
                    ;ändert.
BTFSC   status, Zflag
GOTO    sindGleich  ;Wert in adr1 ist gleich dem Wert in
                    ;adr2
BTFSC   status,CFlag
GOTO    kleiner     ;Wert in adr1 ist kleiner als der in
                    ;adr2
GOTO    groesser    ;Wert in adr1 ist größer als der in
                    ;adr2
```

5.5 Ist Inhalt der Speicherstelle größer / kleiner als Literal

```
MOVLW   literal     ;das Literal kommt ins W-Register
SUBWF   adr         ;das Ergebnis der XOR-Verknüp
                    ;fung kommt nach adr
BTFSC   status, Zflag
GOTO    sindGleich  ;Wert in adr ist gleich dem Literal
BTFSC   status,CFlag
GOTO    kleiner     ;Literal ist kleiner als Wert in adr
GOTO    groesser    ;Literal ist größer als Wert in adr
```

oder:

```
MOVLW   literal     ;das Literal kommt ins W-Register
SUBWF   adr,W       ;das Ergebnis der XOR-Verknüp
```

```
                        ;fung kommt ins W-Register
BTFSC   status, Zflag
GOTO    sindGleich      ;Wert in adr ist gleich dem Literal
BTFSC   status,CFlag
GOTO    kleiner         ;Literal ist kleiner als Wert in adr
GOTO    groesser        ;Literal ist größer als Wert in adr
```

6 Flankenerkennung mittels Polling

Eine Flankenerkennung wird bei Mikrocontroller-Programmen sehr oft benötigt. Flankenerzeugende Bauteile sind z.B. Taster, Lichtschranken, Reedkontakte, Encoder etc. Je nach Anwendung werden unterschiedliche Anforderungen an eine Flankenerkennung gestellt.

Eine der wichtigsten Unterscheidungen ist, ob es sich um einen blockierende oder nicht blockierende Funktion handelt. Bei einer blockierenden Funktion verweilt das Programm so lange in ihr, bis die entsprechende Flanke am Eingang erscheint. Es können keinerlei weitere Abläufe parallel zu dieser Funktion vorgenommen werden. Dieses Verhalten ist mit modalen Meldefenstern in Desktopanwendungen vergleichbar. Auch dort muss diese Meldung zuerst quittiert werden, bevor ein anderes Fenster fokusiert werden kann.

Im Gegensatz dazu kann man mit nicht blockierende Funktionen, eine Quasiparallelität der Programme programmieren. Nachteil bei dieser Methode ist die verlängerte Zykluszeit, die eine Mindestdauer des Impules, dessen Flanke erkannt werden soll, voraussetzt. Auch eine Zunahme der Latenzzeit ist wahrscheinlich. Das ist die Zeit, die zwischen dem Auftreten der Flanke und dem Erkennen durch das Programm, vergeht. Wird diese Latenzzeit intolerabel, muss auf die Hilfe von Interrupts zurückgeriffen werden.

Das Prinzip der Flankenerkennung mittels Polling ist der Vergleich zwischen dem aktuellen Eingangspegel und einem in der Vergangenheit liegenden Wert. Sind beide gleich, gab es keinen Flankenwechsel. Dieser liegt nur dann vor, wenn der neue Pegel sich vom alten unterscheidet. Die Abbildung 2 verdeutlicht diesen Sachverhalt.

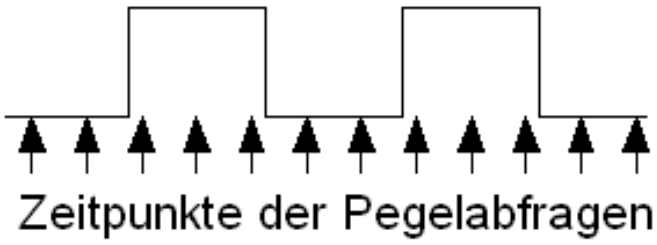
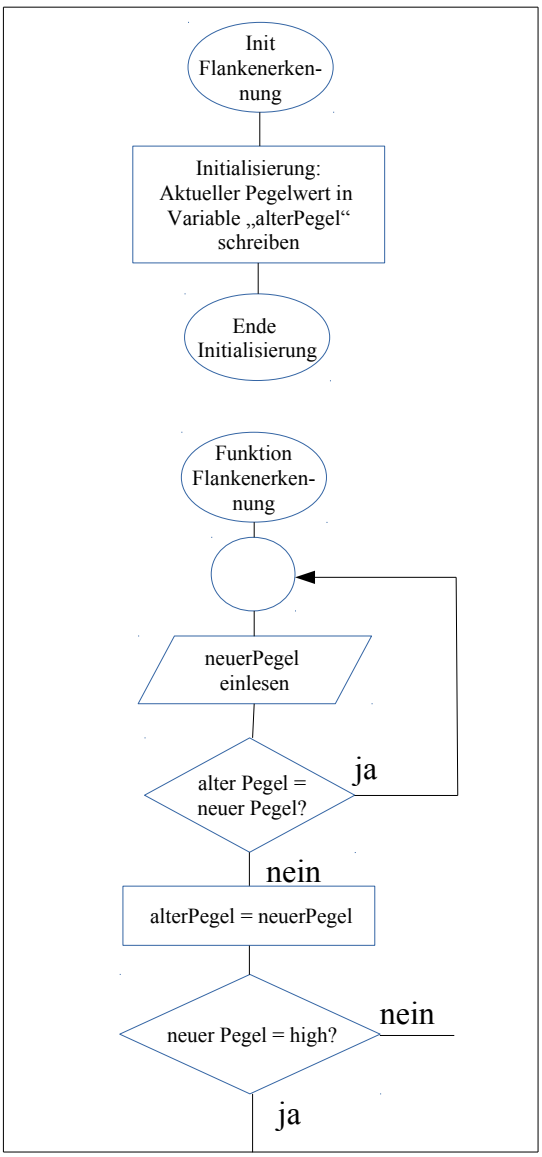


Abbildung 2: Abtastzeitpunkte bei Flankenerkennung

Blockierende Funktion:



Bei der Initialisierung wird der aktuelle Pegel in einer Variablen gespeichert. Dieser wird dann mit dem neu eingelesenen Pegel verglichen um so einen Pegelwechsel (= Flanke) zu erkennen. Diese Initialisierung erfolgt i.d.R. beim Programmstart.

Einsprungpunkt (Ziellabel), falls es keinen Pegelwechsel gegeben hat.

Aktueller Pegel wird eingelesen

und mit altem Pegel verglichen. Sind beide gleich, fand keine Pegelwechsel statt, ansonsten wurde eine Flanke (welche ist noch unklar) gefunden.
Den neuen Pegel muss man sich merken.

Ist der neue Pegel ein High, wurde eine Low-High Flanke gefunden (Ausgang ja).
Ist aber der neue Pegel Low, handelt es sich um eine High-Low Flanke (Ausgang nein)

Man erkennt, dass die Funktion nur dann verlassen wird, wenn eine Flanke gefunden wurde. Somit wartet das Programm u.U. ewig, falls keine Flanke kommt oder der Sensor bzw. der Eingang defekt ist.

PIC-Assemblerprogramm

```
;INIT_Flankenerkennung
Init_Flanke
        BSF     status, rp0     ;wegen TRIS auf Bank 1 umschalten
        BSF     trisx,y         ;x = a oder b, y = 0 bis 7 (Impulseingang)
        BCF     status, rp0     ;zurück auf Bank 0

        MOVF    portx, W        ;gesamter Port lesen
        ANDLW   ????????b       ;Maskenaufbau, je nach Eingangspin
        MOVWF   alterPegel      ;Pegel merken
        …
        …


;Funktion_Flankenerkennung
warte_Flanke
        MOVF    portx, W        ;aktueller Pegel einlesen
        ANDLW   ????????b       ;unwichtige Bits löschen (Maske wie oben)
        XORWF   alterPegel, W   ;die beiden Pegel vergl., Ergebnis in W
        BTFSC   status, zflag   ;beide gleich, Ergebnis = 0, Zflag = 1
        GOTO    warte_Flanke    ;keine Flanke, weiter warten
        XORWF   alterPegel      ;neuen Pegel merken[8]
        BTFSS   alterPegel, y   ;prüfe, welche Flanke es ist. Dazu das ent-
        GOTO    hl_flanke       ;sprechende Bit in der Variable „alterPegel“
        GOTO    lh_Flanke       ;prüfen.
        …
        …
```

Das Symbol trisx steht entweder für trisa oder trisb. Das sind die beiden Datenrichtungsregister auf Bank 1 Adresse 5 bzw. 6. Der Wert für y ist das Dezimaläquivalent der Maske ????????b. z.B. Maske = 00001000 => y = 3.

Für die Maske ????????b muss das entsprechende Bitmuster eingesetzt werden, um nur das gültige Eingangssignal übrig zu lassen. Der Rest wird auf 0 gesetzt. Z.B. ist der Impulseingang auf RA, 3 muss die Maske 00001000b sein.

8 Diese Vorgehensweise ist anfangs etwas undurchsichtig. Aber: wenn eine Flanke gefunden wurde, ist durch die XOR-Verknüpfung der Wert im W-Register genau gleich wie die Maske. Erfolgt nun mit diesem Wert eine XOR-Verknüpfung mit der Variable „alterPegel“, wird an genau der passenden Stelle das Bit invertiert und damit auf den neuen Pegel gesetzt.

7 Aufbau von Konstantentabellen

Die Harvard-Architektur verhindert per se den Zugriff auf den ROM-Speicher, o üblicherweise Tabellen mit Konstanten abgelegt werden. Diese Tabelle dienen u.a. zur Initialisierung, Linearisierung von Kennlinien oder auch als Hilfstabellen für mathematische Berechnungen.

In unseren Fall wird eine Tabelle zur Umsetzung von Binärwerten in ein Bitmuster für eine 7-Segmentanzeige entwickelt. Es gibt keinen mathematischen Zusammenhang um aus binären Werten das Muster zu berechnen. Somit benötigt man eine Zuweisungstabelle.

```
            MOVLW   5           ;diese Zahl soll auf dem Display
                                ;angezeigt werden
            CALL    convert
            MOVWF   port
            …
            …
            …
convert
            ANDLW   15          ;damit die Tabelle nicht übersprun
                                ;gen werden kann
            ADDWF   pcl         ;Inhalt von W auf PCL addieren
            RETLW   00111111B   ;Muster für die 0
            RETLW   00000110B   ;für 1
            RETLW   01011011B   ;für 2
            RETLW   01001111B   ;3
            RETLW   01100110B   ;4
            RETLW   01101101B   ;5
            RETLW   01111101B   ;6
            RETLW   00000111B   ;7
            RETLW   01111111B   ;8
            RETLW   01101111B   ;9
            RETLW   01110111B   ;A
            RETLW   01111100B   ;b
            RETLW   00001111B   ;C
            RETLW   01011110B   ;d
            RETLW   01111001B   ;E
            RETLW   01110001B   ;F
```

8 Längenmessung mittels Impulsrad

```
;Aufgabe Nr 8 aus µP-Aufgabensammlung
;(c) Stefan Lehmann

;Deklaration des Prozessortyps
                        device 16F84
;Deklaration diverser Symbolnamen
indirekt    EQU     0           ;Register indirect
pcl         EQU     2
status      EQU     3           ;Statusregister an Adr. 3
fsr         EQU     4           ;Zeigerregister für ind. Adressierung
porta       EQU     5           ;Adresse 5 auf Bank 0
portb       EQU     6
trisa       EQU     5           ;Adresse 5 auf Bank 1
trisb       EQU     6

cflag       EQU     0           ;Carryflag
zflag       EQU     2           ;Zeroflag
rp0         EQU     5           ;Bankumschaltbit
signal      EQU     0           ;Eingangssignal an RA,0

counter0    EQU     10H         ;LSB des 4-stelligen Zählers
counter1    EQU     11H
counter2    EQU     12H
counter3    EQU     13H

ii          EQU     0CH         ;Zählvariable
alterPegel  EQU     0DH         ;für Flankenerkennung
Digit       EQU     0EH         ;Hilfregister Digitanwahl

signalmaske EQU     00000001B
digitmaske  EQU     11110001B   ;je nach Ansteuerung der Digits

signaleingang EQU   0           ;Bit 0 von Port RA

cold:
            GOTO    start       ;Unterprogramme überspringen

;Unterprogramm FLANKE
```

```
;UP liest nur den Port ein, maskiert das richtige Bit und macht eine
;XOR-Verknüpfung mit dem alten Pegel
flanke
            MOVF    porta,W     ;Port RA nach W lesen
            ANDLW   signalmaske ;nur RA0 ist gültig
            XORWF   alterPegel, W ;mit vorherigen Pegel vergleichen
            RETURN              ;bei ZFlag = 1, kein Pegelwechsel

;Unterprogramm INIT_FSR
;setzt die notwendigen Variablen auf deren Startwert. Das Setzen von
;DIGIT ist eigentlich nur bei der Anzeigeroutine notwendig, schadet aber
;beim anderen Aufruf nicht.
init_fsr
            CLRF    digit       ;nur für Anzeigeroutine wichtig
            MOVLW   4           ;Anzahl der vorhandenen Stellen
            MOVWF   ii                      ;ii = Laufvariable
            MOVLW   counter0    ;Startadresse in W laden
            MOVWF   FSR         ;und ins Zeigerregister laen
            RETURN

;DISP gibt den Inhalt von COUNTER auf der 7-Segmentanzeige aus
disp
            CALL    init_fsr
            DECF    ii          ;niederwert. Stelle nicht anzeigen
            INCF    fsr         ;deshalb auch Zeiger um eins weiter
disp1
            MOVF    indirekt, W ;Wert holen
            CALL    convert     ;in Bitmuster für 7-Seg. umsetzen
            MOVWF   portb       ;Segmentinf anlegen
            CALL    digitimpuls
            MOVLW   3           ;Zeiger in 3er Schritten erhöhen
            ADDWF   digit
            INCF    fsr
            DECFSZ  ii          ;hier max. 3 x
            GOTO    disp1
            RETURN              ;danach fertig

;DIGITIMPULS erzeugt je nach anzuzeigender Stelle einen Impuls an RA1,
RA2 oder RA3
digitimpuls
```

```
        MOVF    digit,W     ;Digitinfo nach W lesen
        ADDWF   pcl         ;an die entsprech. Position springen
        BCF     porta,1     ;impuls an RA,1 erzeugen
        BSF     porta,1     ;Impuls wieder egnehmen
        RETURN
        BCF     porta,2     ;Impuls an RA,2
        BSF     porta,2
        RETURN
        BCF     porta,3     ;impuls an RA,3
        BSF     porta,3
        RETURN

;CONVERT wandelt den Wert in W in das passende Bitmuster einer 7-Segmentanzeige um
convert
        ANDLW   15          ;max. 16 Tabelleinträge
        ADDWF   pcl
        RETLW   00111111B   ;Muster für die 0
        RETLW   00000110B   ;für 1
        RETLW   01011011B   ;für 2
        RETLW   01001111B   ;3
        RETLW   01100110B
        RETLW   01101101B   ;5
        RETLW   01111101B   ;6
        RETLW   00000111B   ;7
        RETLW   01111111B   ;8
        RETLW   01101111B   ;9
        RETLW   01110111B   ;A
        RETLW   01111100B   ;b
        RETLW   00001111B   ;C
        RETLW   01011110B   ;d
        RETLW   01111001B   ;E
        RETLW   01110001B   ;F


start
;Initialisierung der Ports und Variablen
init:                       ;man darf einer Adresse 2 Namen geben
        BSF     status,rp0  ;auf Bank 1 umschalten
        MOVLW   11110001B   ;RA1 bis RA3 Ausgänge für Digit
```

```
                MOVWF   trisa           ;RA0 bleibt Eingang
                CLRF    trisb           ;Port RB komplett auf Ausgang
                BCF     status,rp0      ;zurück auf Bank 0
                MOVLW   15              ;Digitselekt auf 1 setzen
                MOVWF   porta
reset
                CLRF    counter0        ;Zähler auf 0 stellen
                CLRF    counter1
                CLRF    counter2
                CLRF    counter3

;Hauptschleife. Hier wird zuerst der Zählerstand angezeigt und
;dann auf die Flanke gewartet
display
                CALL    disp
main
                CALL    flanke          ;bei ZFlag = 1 ist Flanke da
                BTFSC   status, zflag   ;
                GOTO    main            ;keine Flanke, weiter warten
flanke_da
                XORWF   alterPegel      ;neue Flanke merken
                BTFSS   alterPegel,signaleingang    ;richtige Flanke?
                GOTO    main            ;war falsche Flanke
flanke_ok
                CALL    init_fsr        ;FSR für ind. Adressierung setzen
                INCF    indirekt        ;niedrigste Stelle um 2 hochzählen
counting
                INCF    indirekt        ;
                MOVF    indirekt,W      ;hochgezählter Wert in W
                XORLW   10              ;ist der Wert 10 erreicht?
                BTFSS   status, zflag
                GOTO    display         ;zurück und Wert anzeigen
overflow
                CLRF    indirekt        ;aktuelle Stelle löschen
                INCF    fsr             ;Zeiger um einen Schritt weiter
                DECFSZ  ii              ;max. 4 Durchläufe
                GOTO    counting        ;nächste Stelle bearbeiten
                GOTO    display         ;zurück und Wert anzeigen

                END
```