

Digitaltechnik-Praktikum

Sommer-Semester 2024

Projekt: IR-Remote

Team: Sebastian Pasinski, Benedikt Schnörr

Prüfer: Prof. Dr. Francesco Volpe

Inhalt

Planung	3
Aufgabenstellung	3
Anforderungen	3
Anwendungsfalldiagramm	3
Skizze	4
Schaltplan	5
Mikrocontroller	5
Taktgebung	5
Programmier-/Debugging-Schnittstelle	6
Spannungsregulierung	6
Schalter	7
IR-LED	7
Layout	8
Anordnung der Bauteile	8
Routing	8
Platine	9
Software	10
Einsatz von Interrupts	10
Initialisierung der Interrupts im Programm	10
Interrupt-Routine	12
Bitübertragung mit dem RC5-Protokoll	14
Konfiguration des PWM-Moduls	15
Konfiguration von Timer 0	16
RC5-Übertragung im Code	16
Main-Routine	18
Vollständiger Code	22
Quellen	32
Abbildungsquellen	32

Planung

Aufgabenstellung

Die Aufgabenstellung für das Digitaltechnik-Praktikum ist es, eine Infrarot-Fernbedienung zu bauen. Diese soll einen Receiver ansteuern, welcher das RC5-Protokoll unterstützt. Es sollen die wichtigsten Funktionen eines CD-Players mit der Fernbedienung angesprochen werden können. Dazu gehören:

- Play
- Stop
- Pause
- Forward (vorspulen)
- Backward (zurückspulen)

Anforderungen

Die Anforderungen für dieses System sind, dass es auf einer eigenen Platine aufgebaut und batteriebetrieben arbeiten soll. Im Verlauf des Projekts wird zunächst ein Schaltplan entworfen, der die elektrischen Bauteile und den Mikrocontroller enthält, sodass daraufhin eine Platine designt und die Bauteile bestellt werden können. Im zweiten Schritt wird dann der Code entwickelt, mit dem die gedrückte Taste auf der Fernbedienung ausgelesen und das passende Signal als RC5-Code gesendet wird.

Anwendungsfalldiagramm

Die grundlegende Funktionsweise des Systems wird in Abb.1 dargestellt.

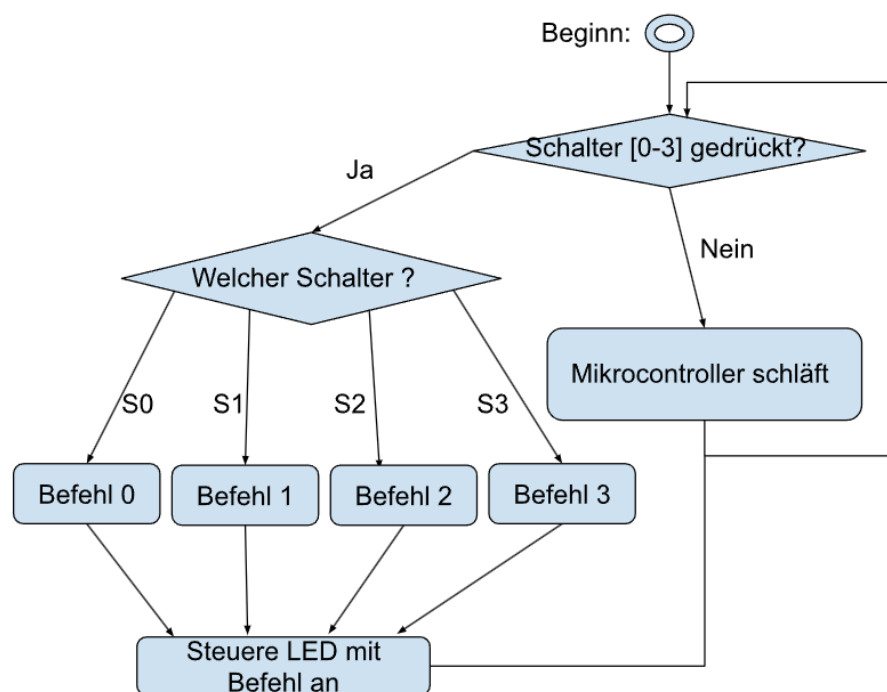


Abb. 1: Anwendungsfalldiagramm der Funktionsweise im System

Skizze

In Abb. 2 ist eine grobe Skizze der Fernbedienung zu sehen, um die Funktionen der einzelnen Taster zuzuordnen.

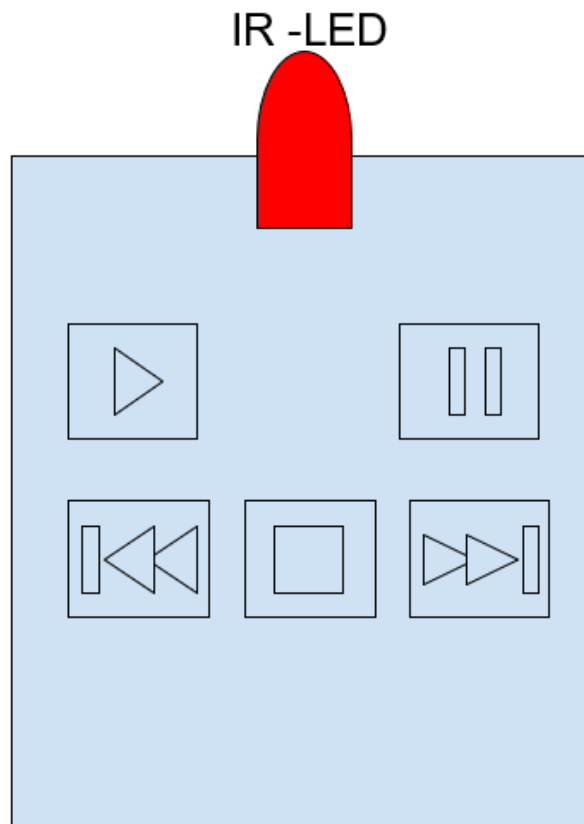


Abb. 2: Skizze der Fernbedienung

Schaltplan

Mikrocontroller

Der in der Schaltung verwendete Mikrocontroller ist ein PIC18F4525, an ihn sind neben Schaltern und Infrarot-LED noch weitere Schaltkreise angeschlossen. In Eagle wird zur Darstellung ein PIC18F4520 verwendet, der aber die gleiche Pin-Belegung und die gleichen Abmessungen wie der PIC18F4525 besitzt und daher als Alternative im Schaltplan benutzt werden kann.

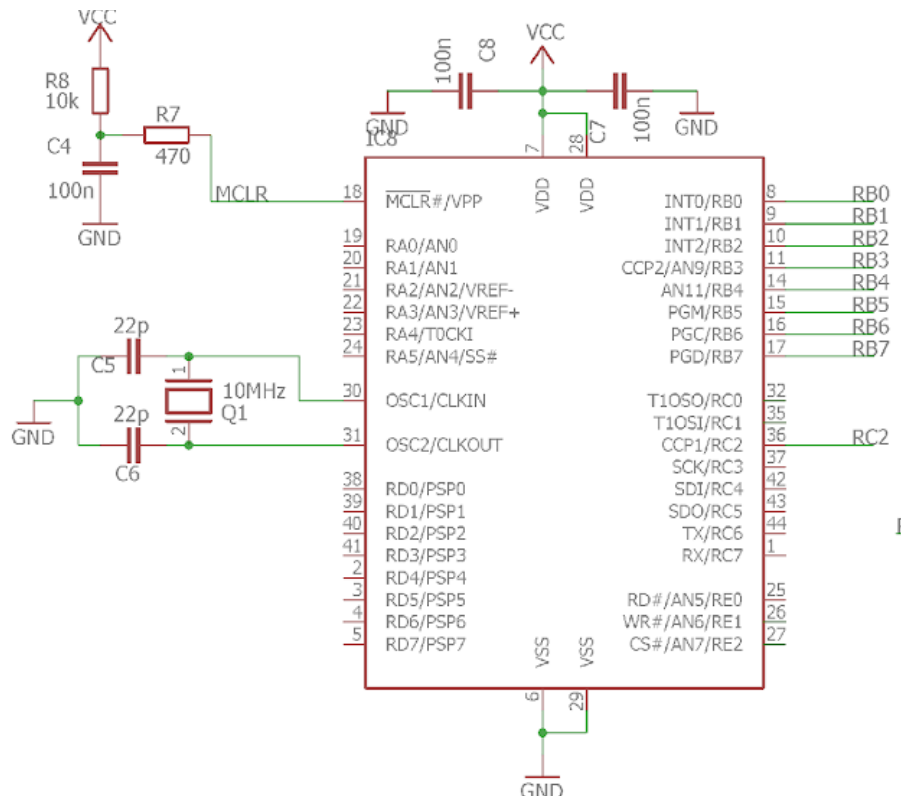


Abb. 3: Grundsaltung des PIC18F4525 mit Quarz, Reset und I/O

Taktgebung

Der Quarz dient als externer Oszillator für den Mikrocontroller, der die Taktfrequenz vorgibt. Das Datenblatt des PIC18F4525 empfiehlt für die Oszillator-Konfiguration "High Speed Crystal/Resonators" mit einer Frequenz von 10 MHz die Verwendung von Kondensatoren mit einer Kapazität von 15 pF. In der vorgegebenen Schaltung wurde eine höhere Kapazität verwendet, dies verbessert die Stabilität der Schaltung, kann aber zur Vergrößerung der Startzeit führen [1].

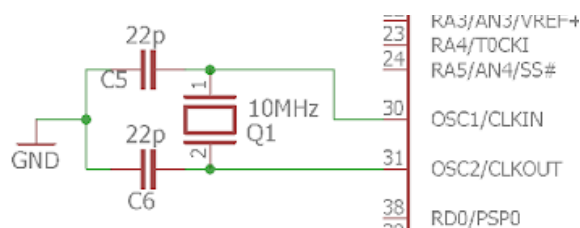


Abb. 4: Quarzoszillator mit Kapazitäten

Programmier-/Debugging-Schnittstelle

Die in Abbildung 5 dargestellten Schaltungsblöcke ermöglichen das Programmieren und Debuggen des PIC [1].

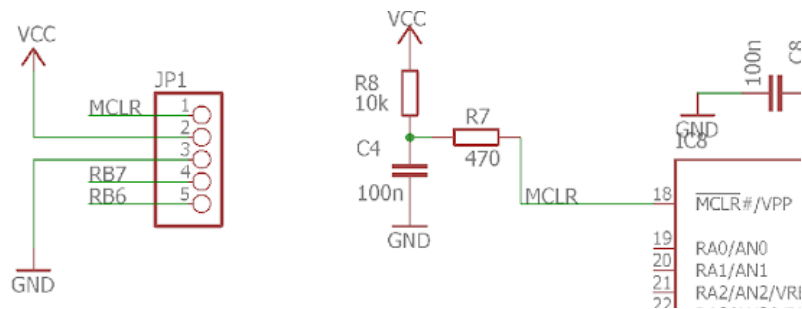


Abb. 5: Schnittstelle des Mikrocontrollers zum Debuggen/Programmieren

Spannungsregulierung

Da die Schaltung laut Anforderungen mit einer 9 V-Batterie betrieben wird, muss ein Spannungsregulierer eingesetzt werden, um die Spannung auf die Betriebsspannung des PIC von 5 V zu reduzieren. In der fertigen Schaltung wird dazu der ON Semiconductor NCV1117ST50 genutzt. Im Schaltplan wird allerdings ein AP1117E50G eingesetzt, der die gleiche Pin-Belegung und Funktionsweise besitzt. Die Kapazitäten an Ein- und Ausgang sorgen dafür, dass der Regelkreis im Chip des Spannungsregulierers nicht beginnt zu schwingen. Diese Fähigkeit ist wichtig, um eine stabile Ausgangsspannung bereitzustellen.

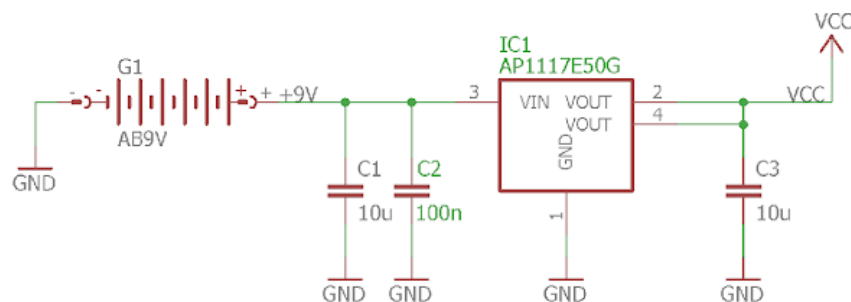


Abb. 6: Batterie mit Spannungsregulierer und Kapazitäten

Schalter

Die Schalter, welche an verschiedenen Pins von Port B des Mikrocontrollers angeschlossen sind, sorgen dafür, dass im ausgeschalteten Zustand die Versorgungsspannung von 5 V am jeweiligen Pin anliegt. Im angeschalteten Zustand werden die Pins auf Masse gezogen.

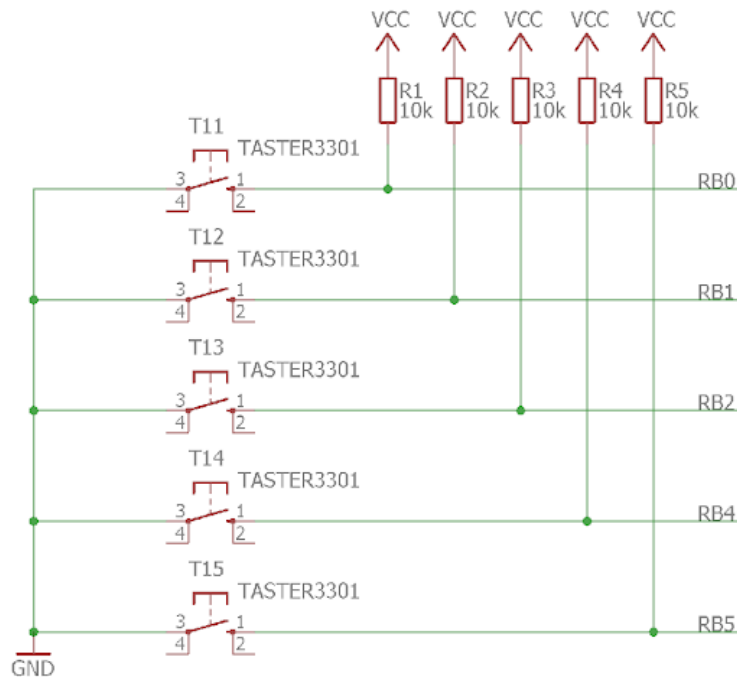


Abb. 7: Verschaltung der Taster als Active-Low-Elemente

IR-LED

Für das Aussenden der Lichtsignale wird eine 5 mm Vishay Infrarot-LED verwendet. Der Vorwiderstand berechnet sich mit dem Spannungsabfall an der LED und dem gewünschten Strom in diesem Zweig. Laut Datenblatt liegt die Nennspannung bei 1,35 V, der Nennstrom kann bis zu 100 mA betragen [5]. Mit dem Ohm'schen Gesetz ergibt sich ein Widerstand von 36,5 Ω . Mit dieser Berechnung wird ein Widerstand von 47 Ω gewählt, damit der Strom durch die LED mit Sicherheit nicht zu groß wird.

Die LED wird an Pin RC1 angeschlossen, da an diesem Pin mit wenig Aufwand eine modulierte Rechteckspannung mit dem PWM-Modul des PIC ausgegeben werden kann. Diese wird für das RC5-Protokoll benötigt.

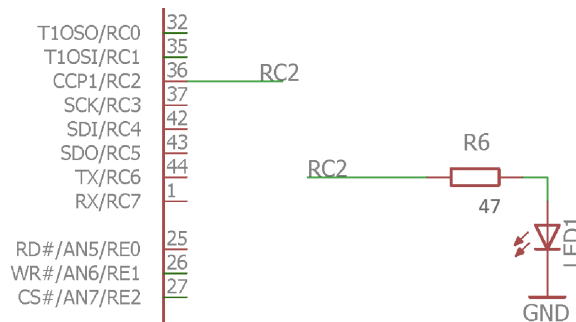


Abb. 8: Verschaltung der IR-LED mit Vorwiderstand

Layout

In Abb. 9 ist das finale Layout zu sehen, mit dem auch die reale Platine erstellt wurde.

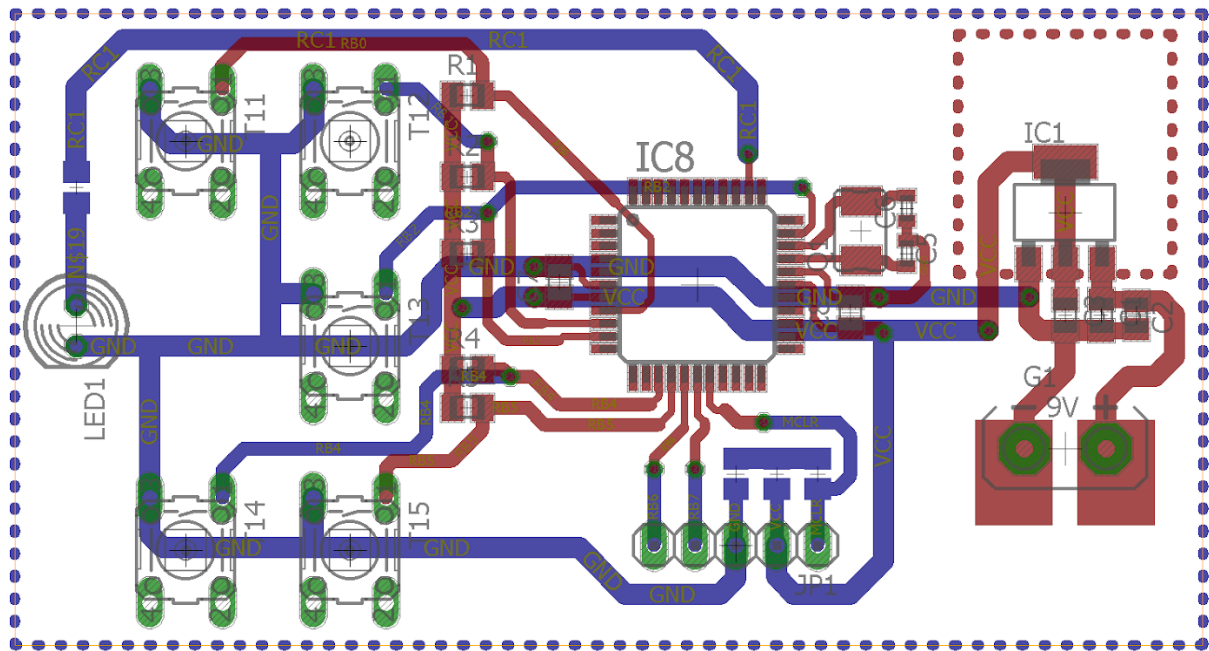


Abb. 9: PCB-Layout in Eagle mit Platzierung und Routing

Anordnung der Bauteile

Beim Layout werden zuerst funktionale Gruppen mit Bauteilen, die direkt miteinander verbunden sind, gebildet. Diese werden dann auf der Platine angeordnet. Die funktionale Gruppe um den Quarz sollte möglichst nah an den angeschlossenen Eingängen des Mikrocontrollers liegen, da der Quarz sehr anfällig gegenüber Störsignalen ist.

Die Kondensatoren C7 und C8 liegen ebenfalls möglichst nah an den verbundenen Pins zur Stromversorgung, um Schwankungen in der Versorgungsspannung durch Störungen möglichst schnell auszugleichen.

Zusätzlich ist bei der Spannungsregulierung eine Kühlfläche vorgesehen, um die entstehende Wärme möglichst gut zu verteilen und den Chip dadurch zu kühlen.

Die Taster werden so angeordnet wie das bei handelsüblichen Fernbedienungen der Fall ist und in Abb.2 skizziert wurde. Auch die Anordnung der LED am vorderen und die der Batterie am hinteren Ende der Fernbedienung wird aus bekannten Fernbedienungen übernommen.

Routing

Beim Routing wird besonders darauf geachtet, genug Abstand zwischen den Leiterbahnen einzuhalten und möglichst breite Leiterbahnen und keine rechten Winkel zu verwenden.

Diese Grundsätze helfen, elektromagnetische Störungen zu minimieren.

Die Unterseite der Platine besteht zum größten Teil aus einer Masse-Fläche, so werden Masse-Schleifen vermieden und das Routing der Masse-Leitungen vereinfacht [6].

Die Größe und Form der Durchkontaktierungen, die Breite der Leiterbahnen und die Dimensionen von Tastern und SMD-Bauteilen wurden in Absprache mit dem Labor gewählt.

Platine

Die fertige Platine ist in den Abbildungen 10 und 11 zu sehen.

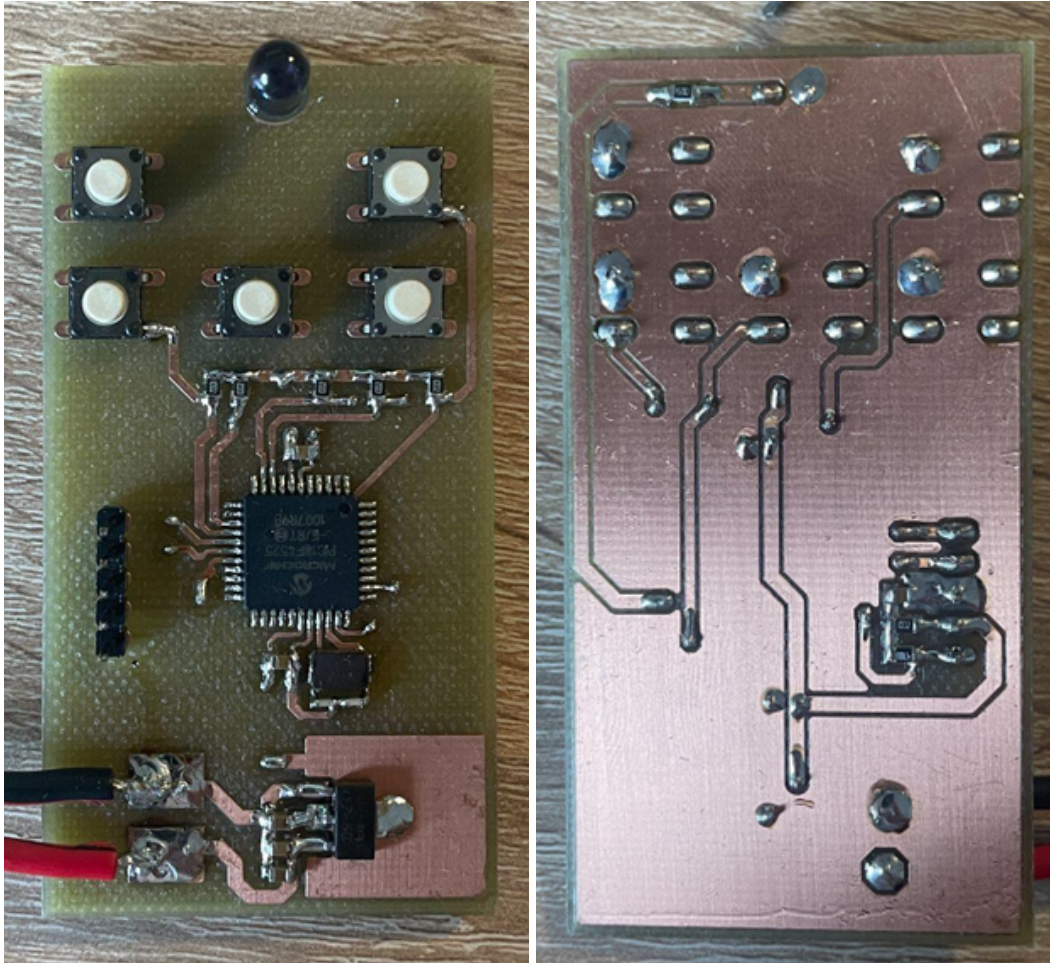


Abb. 10 (links): Fertige Platine obere Seite, Abb. 11 (rechts): untere Seite

Software

Nachdem die Hardware fertiggestellt ist, fehlt für eine funktionsfähige Fernbedienung nur noch die Programmierung des Mikrocontrollers. Diese wird zunächst in 2 Teile aufgeteilt. Der erste Teil beschäftigt sich mit dem Auslesen der gedrückten Taster und dem Zuordnen von Taster und Befehl. Im zweiten Teil wird der Befehl dann mit dem RC5-Protokoll gesendet.

Einsatz von Interrupts

Damit eine dauerhafte Schleife im Hauptprogramm vermieden werden kann, welche die Pegel an den Pins abfragt, kann die Interrupt-Funktion des Mikrocontrollers verwendet werden. So kann der PIC die meiste Zeit in den Sleep-Modus versetzt werden und spart dadurch Energie.

Das Hauptprogramm wird im Sleep-Modus gestoppt. Dann kann jederzeit durch eine Spannungsänderung an einem Interrupt-Pin eine Unteroutine aufgerufen werden, die den gedrückten Taster ausliest und den passenden Befehl sendet.

Der Ablauf dabei ist wie folgt: Bei einem Interrupt vervollständigt der Microcontroller die letzte Anweisung und speichert den Inhalt des Program-Counters und des Status-Registers auf dem Stack. Dann lädt er automatisch die Adresse 0x0008 (spezifisch bei PIC18F4525) in den Program-Counter und springt so an diese Stelle im Programm, um dort eine vom Benutzer geschriebene Routine auszuführen. Vor der letzten Anweisung muss das entsprechende Interrupt Flag, welches gesetzt wurde, um das Auftreten der Unterbrechung zu kennzeichnen, wieder zurückgesetzt werden. Die letzte Anweisung der Routine sorgt für eine sichere Rückkehr des Program-Counters zum aktuellen Befehl in der Hauptroutine, weitere Unterbrechungen werden dann wieder ermöglicht [3].

Initialisierung der Interrupts im Programm

In der Fernbedienung werden verschiedene Interrupts genutzt. Die Taster an den Pins RB 0, 1 und 2 lösen die flankengesteuerten, externen Interrupts INT0, INT1 und INT2 aus. Die Pins RB 4 und 5 dagegen lösen Port B On Change-Interrupts aus, welche auf Zustandsänderungen der beiden Pins reagieren. Diese Interrupts werden im Folgenden initialisiert.

Dies geschieht durch das Setzen oder Rücksetzen spezieller Bits in den passenden Konfigurations-Registern, um die Einstellungen der Interrupts festzulegen.

Dabei ist es wichtig die Interrupt-Flags zurückzusetzen, damit keine fälschliche Unterbrechung stattfindet, bevor das Hauptprogramm gestartet wird. Außerdem werden die Interrupts durch das Setzen des jeweiligen Interrupt Enable-Bits aktiviert und ihre Priorität festgelegt. Im Fall von INT0/INT1/INT2 wird auch die Flanke angegeben, bei welcher ein Interrupt ausgelöst wird. Am Ende werden die Global Interrupt Enable-Bits gesetzt, um die Interrupts mit hoher und niedriger Priorität im Allgemeinen zu aktivieren.

```
init_portb:
    clrf PORTB           ; reset port b and its latch
    clrf LATB
```

```

bcf ADCON0,ADON
; disable ad converter and reset its configuration bits

movlw 0xFF          ; declare all Pins at PORTB as digital
movwf ADCON1        ; pins

movlw 0xFF          ; set all port b pins as inputs
movwf TRISB

movlw b'00111011'   ; set pins with buttons to '1' to prevent
movwf PORTB
;triggering interrupt because buttons
;create high state as default (active low buttons)

bcf INTCON,INT0IF    ; clear interrupt flags for pins rb
bcf INTCON3,INT1IF   ; 0/1/2/4/5 to prevent getting multiple
bcf INTCON3,INT2IF   ; interrupts with one button press
bcf INTCON,RBIF

bcf INTCON2,RBPU
; activate internal weak pull-ups for all pins on port b

bsf INTCON,INT0IE
bsf INTCON3,INT1IE
bsf INTCON3,INT2IE
bsf INTCON,RBIE
; enable interrupts for interrupt 0/1/2 and on change
; interrupts on port b

bcf INTCON2,INTEDG0
bcf INTCON2,INTEDG1
bcf INTCON2,INTEDG2

; set interrupts 0/1/2 to trigger on falling edge

bsf RCON,IPEN        ; enable interrupt priorities

bsf INTCON3,INT1IP
; set interrupt priority for interrupts 1 & 2 , (0 always high)

bsf INTCON3,INT2IP
; and on change interrupt to high priority

bsf INTCON2,RBIP

bsf INTCON,GIEH
; activate all high priority interrupts

```

```

    bsf INTCON,GIEL
;activate all low priority interrupts
    bsf INTCON,GIE
; activate all interrupts

    return

```

Interrupt-Routine

Um in der Anwendung des Programms nachvollziehen zu können, welcher Schalter gedrückt wurde, wurde innerhalb der High Interrupt Routine ein Algorithmus implementiert, welcher die jeweiligen Interrupt-Flags und die Zustände der Pins abfragt. Wenn der entsprechende Schalter erkannt wurde, wird eine Sendefunktion mit einem bestimmten RC5-Befehl ausgeführt, welche dann den passenden Befehl lädt.

```

HighInt:
    btfsc INTCON,RBIF      ; pin 4/5 pressed
    call send_rb_on_change
    btfsc INTCON,INT0IF    ; pin 0 pressed
    call send_rb_0
    btfsc INTCON3,INT1IF   ; pin 1 pressed
    call send_rb_1
    btfsc INTCON3,INT2IF   ; pin 2 pressed
    call send_rb_2
    retfie FAST

```

In der Interrupt-Routine wird also zuerst ermittelt, ob sich eine Änderung an den Pins RB4 und RB5 ereignet hat. In der Routine `send_rb_on_change` wird dann mit den Pin-Zuständen ermittelt, welcher Pin genau für den Interrupt verantwortlich war.

```

send_rb_on_change:

    btfss PORTB,4      ; check if pin 4 was pressed
    call send_rb_4
    btfss PORTB,5      ; check if pin 5 was pressed
    call send_rb_5
    bcf INTCON,RBIF    ; clear interrupt flag to prevent multiple
                        ; interrupts
    return

```

In den daraufhin aufgerufenen Sende-Routinen wird das Interrupt-Flag zurückgesetzt, um mehrfache Unterbrechungen zu vermeiden und wieder in das Hauptprogramm zu springen. Außerdem wird der Bit-Buffer, welcher die Befehle zum Senden enthält, mit dem passenden Befehl überschrieben. Hier ist beispielhaft eine Sende-Routine dargestellt.

```
send_rb_0:
    bsf skip_release,0      ; set skip_release<0> to prevent
                           ; skipping this command
    movlw b'00110101'      ; move rc5 code for stop command to
    movwf bit_buffer_h     ; bit buffer
    movlw b'00110110'      ; 00 in beginning not significant
    movwf bit_buffer_l
    bcf INTCON,INT0IF      ; clear interrupt flag to prevent
                           ; multiple interrupts
    return
```

Die Schnittstelle wurde am Modell getestet, um das Programmverhalten beim Drücken der Taster zu beobachten.

Bitübertragung mit dem RC5-Protokoll

Das RC5-Protokoll ist eine von Philips entwickelte Methode zur Kommunikation mit verschiedenen Endgeräten, wie z.B. CD-Spielern und Fernsehern. Eine Fernbedienung für dieses Protokoll sendet mit einer Wellenlänge von 940 bis 950 nm, welche bei der verwendeten Infrarot-LED mit 940 nm gegeben ist. Ein Befehl der Fernbedienung besteht aus insgesamt 14 Bits, die nacheinander gesendet werden. Die Bits sind in Abb. 13 dargestellt und haben die folgende Bedeutung [7]:

- 2 Start-Bits, immer logisch '1'
- 1 Toggle-Bit, welches seinen Wert bei jedem neuen Tastendruck ändert
- 5 Adress-Bits, um ein konkretes Gerät anzusprechen, wie bspw. CD-Player, TV, Audio-Vorverstärker
- 6 Befehls-Bits

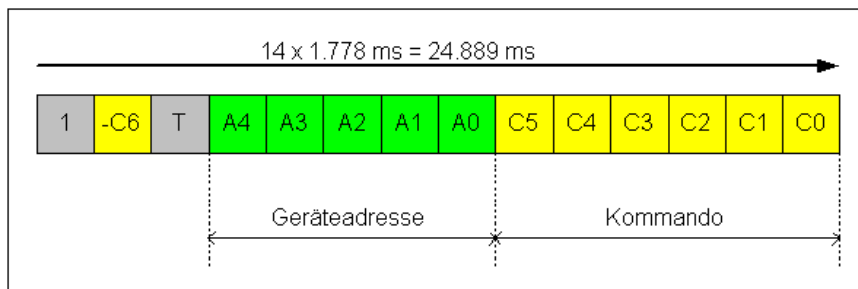


Abb. 12: Zeitliche Abfolge der Bits im RC5-Befehl

Ein Bit besteht dabei immer aus einer Low-Phase und einer Puls-Phase mit einer Länge von je $888,89 \mu\text{s}$. Wie in Abb. 12 zu sehen, verändert die Reihenfolge dieser zwei Phasen den Zustand des Bits zwischen High und Low [7].

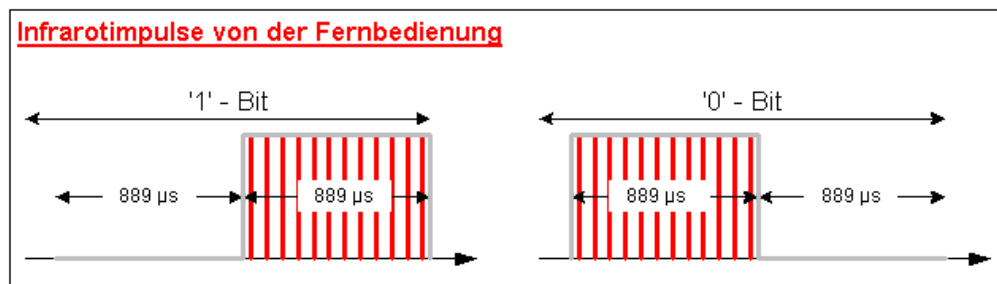


Abb. 13: Reihenfolge der Puls- und Ruhephasen und deren Bedeutung als Bit

Eine Puls-Phase besteht dabei immer aus 32 einzelnen Pulsen einer Rechteckspannung mit einer Frequenz von 36 kHz. Die Dauer der High-Phasen der einzelnen Pulse ist etwa $\frac{1}{3}$ bis $\frac{1}{4}$ der Periodendauer der Rechteckschwingung. So wird weniger Leistung benötigt, als bei einer High-Phase von 50%.

Konfiguration des PWM-Moduls

Die Rechtecksignale werden an Pin RC1 mit dem „Capture/Compare/PWM Module (CCP)“ im PWM-Modus erzeugt. Alle nachfolgenden Ausführungen beruhen auf Informationen im Kapitel 15.4 des Handbuchs für den PIC18F4525 [1].

Bei der Konfiguration wird zunächst die Periodendauer bzw. Frequenz der benötigten Trägerschwingung definiert. Diese kann über die Gleichungen 1 und 2 berechnet werden.

$$T_{Period} = [n_{PR2} + 1] \cdot 4 \cdot T_{OSC} \cdot n_{TMR}$$

Gleichung 1

$$n_{PR2} = \frac{f_{OSC}}{4 \cdot f_{Period} \cdot n_{TMR}} - 1 \approx 44h$$

Gleichung 2

Die Frequenz f_{Period} ist dabei die Träger-Frequenz von 36 kHz. Die Frequenz des Oszillators f_{OSC} ist durch den 10 MHz-Quarz vorgegeben. Der Prescaler-Value $n_{TMR} = 1$ wird aus den Beispielen im Handbuch gewählt. Der berechnete Wert n_{PR2} wird bei der Konfiguration in das PR2-Register eingetragen, um die gewünschte Frequenz der Rechteckspannung zu erreichen. Durch Leitungsverzögerungen und Ungenauigkeiten in der Schaltung muss dieser Wert im realen Programm angepasst werden.

Außerdem muss der Wert für den Duty Cycle berechnet und eingestellt werden. Aus Gleichung 3, die von Microchip vorgegeben wurde [8], ergibt sich ein Wert von 0,250. Da dieser Wert in diesem Zusammenhang nicht als zehnstellige binäre Zahl dienen kann, wird dieser Wert experimentell zu 0x54 bestimmt. Die ersten acht MSBs werden in das CCPR2L-Register und die zwei verbleibenden LSBs in das CCP2CON-Register an Stelle 5 und 4 eingetragen.

$$Pulse\ Width = (CCPR2L:CCP2CON < 5:4 > \cdot \frac{1}{f_{OSC}} \cdot n_{TMR})$$

Gleichung 3

```
init_pwm:
    banksel PR2
    movlw 0x46                ; configure period time with 0x46 for
    movwf PR2                 ; 36 kHz frequency
    movlw b'00010101'         ; configure duty cycle to be between 1/3
    movwf CCPR2L               ; and 1/4 of the whole signal
    movlw b'00001100'         ; configure pwm mode and 2 lsbs of
    movwf CCP2CON              ; duty cycle

    bcf PIR1,TMR2IF           ; clear interrupt flag to prevent
                                ; multiple interrupts
    bcf T2CON,T2CKPS1         ; set timer 2 clock prescaler value
    bcf T2CON,T2CKPS0         ; to '00'
```

Konfiguration von Timer 0

Während die Rechteckspannung von CCP-Modul und Timer 2 erzeugt wird, muss auch die Sendedauer der Rechteckspannung reguliert werden. Wie bereits erwähnt, werden in der Sendephase für jedes Bit 32 Pulse gesendet, dies entspricht einer Dauer von 888,89 μs . Diese Sendedauer wird von Timer 0 beendet, der von einem entsprechend eingestellten Wert bis zum Überlauf hochzählt. Beim Überlauf von 0xFFFF zu 0x0000 wird ein Interrupt erzeugt, über dessen Interrupt Service Routine dann die Sendephase beendet werden kann. Der Startwert des Timers ergibt sich durch die Berechnungen in Gleichung 4 [9].

$$f_{OSC} = 10\text{MHz} \rightarrow f_{Timer} = \frac{f_{OSC}}{4} = 2,5\text{MHz} \rightarrow T_{Timer} = \frac{1}{f_{Timer}} = 0,4\mu\text{s}$$

$$n_{Timer} = 65536 - \frac{T_{Sendedauer}}{T_{Timer}} = 65536 - \frac{888,89\mu\text{s}}{0,4\mu\text{s}} \approx 0xF752$$

Gleichung 4

Im Code wird der Timer dann als 16 Bit-Timer konfiguriert, der mit der Taktfrequenz des PIC vom Startwert 0xF752 hochzählt. Der Prescaler-Wert folgt ebenfalls aus den Berechnungen und kann hier auf "000" gesetzt werden. Außerdem wird für den Overflow von Timer 0 ein Low Priority Interrupt aktiviert, der, wie bereits erwähnt, das Ende der Dauer eines halben gesendeten Bits markiert.

```
movlw b'00001000'      ; configure timer 0 with 16 bit, internal
movwf T0CON             ; clock and prescaler value of '000'
movlw 0xF7              ; set start value for timer 0
movwf TMR0H
movlw 0x52
movwf TMR0L
bcf INTCON,TMR0IF       ; clear interrupt flag to prevent multiple
                        ; interrupts
bsf INTCON,TMR0IE       ; enable timer 0 interrupt
bcf INTCON2,TMR0IP      ; set interrupt priority to low
bsf INTCON,GIE          ; enable all interrupt
```

RC5-Übertragung im Code

Der folgende Code-Ausschnitt zeigt beispielhaft die Übertragung einer logischen '0' in Assembler. Dabei wird aus der Main-Routine die entsprechende Sende-Routine aufgerufen, die dann zuerst 888,89 μs lang die 32 Pulse der Rechteckspannung sendet und dann 888,89 μs lang den Ausgang auf den Low-Zustand legt.

```
send_0:
    call pwm_delay      ; send logic '0' by first sending 32 pwm
    call tmr_delay      ; pulses and then 889 us delay
    return
```


In der `pwm_delay`-Routine werden Timer 0 und Timer 2 auf ihre Anfangswerte gesetzt und dann gestartet. Beim Ausführen des Codes auf dem realen PIC wurde festgestellt, dass der erste Puls nur aus einer Low-Phase besteht, daher wird der Ausgangspin vor dem Starten des ersten Pulses manuell gesetzt.

```
pwm_delay:
    movlw 0xF7          ; set start value for timer 0 which
    movwf TMR0H          ; counts to overflow
    movlw 0x70
    movwf TMR0L
    movlw 0x00          ; set start value for timer 2 which
    movwf TMR2           ; counts for each pulse
    bsf LATC,1          ; set first pulse manually

    bsf T0CON,TMR0ON     ; start timer 0 for 889 us delay
    bsf T2CON,TMR2ON     ; and timer 2 for pwm

    call delay_loop      ; wait for timer 0 low interrupt

    return
```

Die `tmr_delay`-Routine ist ähnlich aufgebaut. Sie startet allerdings nur Timer 0 und nicht Timer 2, da für die Ruhephase nur Timer 0 gebraucht wird, um die 888,89 μ s abzuzählen.

```
tmr_delay:
    ;start timer 0 for pausing
    movlw 0xF7          ; set start value for timer 0 which counts
    movwf TMR0H          ; to overflow
    movlw 0x70
    movwf TMR0L
    bsf T0CON,TMR0ON     ; start timer 0
    call delay_loop      ; wait for timer 0 low interrupt
    return
```

Außerdem gibt es noch eine Routine, die durchlaufen wird, bis ein Timer 0 Low Interrupt ausgelöst wird. Diese Schleife wartet darauf, dass das `tmr_done`-Flag von der Low Interrupt-Routine gesetzt wird.

```
delay_loop:                ;wait for timer 0 interrupt to end half bit
    btfss tmr_done,0
    bra delay_loop
    bcf tmr_done,0
    return
```

In der Low Interrupt-Routine werden dann die Timer gestoppt, die das halbe Bit erzeugen, und das bereits erwähnte `tmr_done`-Flag gesetzt.

```
LowInt:
    bcf T0CON,TMR0ON    ; stop timers 0 and 2 which are used for
    bcf T2CON,TMR2ON    ; pwm pulses and do not need to be used
                        ; after sending these pulses is finished

    bcf INTCON,TMR0IF    ; clear interrupt flag of timer 0 which
                        ; triggers the low interrupt, clearing
                        ; prevents multiple interrupts for one
                        ; overflow event

    bsf tmr_done,0
    retfie              FAST
```

Main-Routine

In der Main-Routine, die zuerst ausgeführt wird, werden zunächst alle Routinen zur Initialisierung aufgerufen. So werden die Kontrollbits für die Ein- und Ausgangsports und das PWM-Modul, wie in den vorherigen Abschnitten angesprochen, passend gesetzt.

```
Main:
    call init_portc      ; configure ports and pwm module
    call init_portb
    call init_ports
    call init_pwm
```

Dann wird der Mikrocontroller direkt in den Sleep-Modus versetzt, um dort auf den nächsten Tastendruck und den dadurch ausgelösten Interrupt zu warten. Dazu wird das `IDLEN`-Bit zurückgesetzt, womit der Modus des Mikrocontrollers ausgewählt werden kann.

```
wait_cmd_loop
    bcf OSCCON,IDLEN ; go to sleep until next button is pressed
    sleep
    nop
    nop
    nop
```

Nachdem der PIC dann durch einen Tastendruck in den normalen Run-Modus versetzt und der gewünschte Befehl in der Interrupt-Routine in den Bit-Buffer kopiert wurde, wird das Toggle-Bit des Befehls angepasst. Wie im Abschnitt zur Theorie hinter dem RC5-Code erwähnt, toggelt dieses Bit bei jedem neu gesendeten Befehl. Das Toggeln und Setzen im Bit-Buffer ist im folgenden Code-Abschnitt dargestellt.

```

    btg bit_buffer_toggle,0    ; set toggle bit in bit_buffer
    btfsc bit_buffer_toggle,0
    bsf bit_buffer_h,3
    btfss bit_buffer_toggle,0
    bcf bit_buffer_h,3

```

Dann werden die Sende-Routinen gestartet. Dazu werden die Bits des Bit-Buffers nach und nach in das Carry-Bit übernommen und dann die passende Sende-Routine für eine logische ,0' oder ,1' gestartet. Damit der Bit-Buffer seinen Wert behält und beim Senden nicht verloren geht, werden die Schiebe-Operation mit einem temporären Bit-Buffer durchgeführt, dessen Wert vor dem Senden aus dem Bit-Buffer kopiert wird.

Die ersten zwei Bits werden beim Senden übersprungen, da ein Befehl beim RC5-Code immer nur 14 Bits enthält. Der Bit-Counter zählt diese 14 Bits ab und beendet die Sende-Schleife, wenn alle Bits gesendet wurden.

```

send_cmd_loop
    movlw d'14'                ; send 14 bits
    movwf bit_ctr
    clrf tmr_done              ; reset flag that stops delay loop
                                ; during sending
    movf bit_buffer_h,W        ; send bit buffer from temporary
                                ; constants to keep value
    movwf bit_buffer_h_tmp
    movf bit_buffer_l,W
    movwf bit_buffer_l_tmp

    rlcw bit_buffer_l_tmp      ; skip first 2 bits because only 14
    rlcw bit_buffer_h_tmp      ; bits hold information
    rlcw bit_buffer_l_tmp
    rlcw bit_buffer_h_tmp

send_bit_loop
    rlcw bit_buffer_l_tmp      ; transfer the next bit that should
    rlcw bit_buffer_h_tmp      ; be sent in carry bit

    btfss STATUS,C             ; test carry bit to see if '0' or
    call send_0                ; '1' should be sent
    btfsc STATUS,C
    call send_1

    decfsz bit_ctr             ; send next bit if not all bits are
    bra send_bit_loop          ; sent already

```

Dieser Sende-Vorgang wird für den zu sendenden Befehl mehrmals wiederholt. Dies beruht auf der Beobachtung, dass der Empfänger bei einzeln gesendeten Befehlen nicht zuverlässig reagiert. Das mehrmalige Senden simuliert einen längeren Tastendruck, da ein richtiger Algorithmus für längere Tastendrücke aus Zeitgründen nicht realisiert werden

konnte. Hierfür wird der Command-Counter zuerst mit der Anzahl der für einen Tastendruck gesendeten Befehle initialisiert.

```
movlw d'3'          ; send every command 3 times
movwf cmd_ctr
```

Dieser Counter wird nach dem Senden des Befehls dekrementiert, um die Sende-Routine so oft durchzuführen, wie bei der Initialisierung gewünscht.

```
decfsz cmd_ctr      ; send command again if it was not sent 3
bra send_cmd_loop   ; times already
```

Zwischen den gesendeten Befehlen wird mit dem folgenden Code-Abschnitt eine Pause eingebaut. So kann der Empfänger besser zwischen den Befehlen unterscheiden. Dieses Prinzip wurde der tatsächlichen Fernbedienung des CD-Spielers entnommen. Der Code besteht aus zwei verschachtelten Schleifen, deren Zähler nacheinander herunterzählen. Die Verzögerung, die durch diesen Code entsteht, liegt bei ca. 9 ms.

```
movlw 0xFF          ; load inner and outer delay counter to
movwf rand_delay_ctr ; create delay between sending the
movlw 0x40          ; command multiple times
movwf rand_delay_ctr_outer

wait_rand_delay_outer ; create delay of around 9 ms
wait_rand_delay
    nop
    nop
    decfsz rand_delay_ctr
    bra wait_rand_delay
    decfsz rand_delay_ctr_outer
    bra wait_rand_delay_outer
```

Wenn der Befehl mehrmals gesendet wurde, wird der PIC wieder in den Sleep-Modus versetzt, um auf den nächsten Befehl zu warten.

```
bra wait_cmd_loop    ; go to sleep after sending 3 commands
```

Eine weitere Beobachtung war, dass der Befehl beim Drücken der Taster an den Pins RB4 und RB5 zweimal gesendet wurde. Die On Change-Interrupts dieser Pins sind nämlich sowohl auf die steigende, wie auch auf die fallende Flanke sensitiv. Als Lösung wird das skip_release-Flag eingeführt. Dieses wird bei jedem der zwei ausgelösten Interrupts getoggelt, sodass es für die zwei Flanken unterschiedliche Werte hat. Das Senden des Befehls wird dann ausgesetzt, wenn das Flag den Wert ,1' hat, also der Interrupt beim Drücken der Taste und nicht beim Loslassen erzeugt wurde. So wird für einen Tastendruck auch nur einmal die Sende-Routine durchlaufen. Beim Interrupt durch RB0, RB1 und RB2 wird das Bit manuell in der Interrupt-Routine gesetzt, um diesen Befehl auf jeden Fall zu senden. Der folgende Code-Abschnitt zeigt den Einstiegspunkt beim Überspringen des Befehls und die Abfrage des Flags.

```
skip_cmd
    movlw 0x00          ; initialize skip_release flag
    movwf skip_release

wait_cmd_loop
    bcf OSCCON,IDLEN    ; go to sleep until next button is
    sleep              ; pressed

    btfss skip_release,0 ; skip second command when button is
    bra skip_cmd         ; released at rb4 and rb5 because
                        ; interrupt is triggered on change, not
                        ; only on falling edge
```

Vollständiger Code

```

LIST P=18F4525                ; directive to define processor
#include "P18F4525.INC" ; processor specific variable definitions

;*****
****
;Configuration bits

; Oscillator Selection:
CONFIG OSC = HS                ; set oscillator mode to high-speed crystal
with
                                ; external 10 MHz quartz
CONFIG WDT = OFF               ; disable watchdog timer
CONFIG LVP = OFF               ; disable low voltage programming, because
MPLAB                           ; ICD 3 can do high voltage programming

;*****
****
;Variable definitions

bit_buffer_h equ 0x00 ; hold bits 13-8 of command
bit_buffer_l equ 0x01 ; hold bits 7-0 of command
bit_ctr equ 0x02 ; count 14 bits while
sending
tmr_done equ 0x03 ; signal that half bit is
finished
cmd_ctr equ 0x04 ; count times of sending
command
skip_release equ 0x05 ; decide between falling
and rising edge at rb 4/5
bit_buffer_toggle equ 0x06 ; toggle 3rd sent bit
bit_buffer_h_tmp equ 0x07 ; hold bits 13-8 of command
while sending
bit_buffer_l_tmp equ 0x08 ; hold bits 7-0 of command
while sending
rand_delay_ctr equ 0x09 ; count during inner delay
loop between commands
rand_delay_ctr_outer equ 0x0A ; count during outer delay
loop between commands

;*****
****
;EEPROM data

ORG 0xf00000

DE "Test Data",0,1,2,3,4,5

;*****
****
;Reset vector

```

```

        ORG    0x0000

        goto   Main            ; go to start of main code

;*****
****
;High priority interrupt vector

        ORG    0x0008

        bra    HighInt        ; go to high priority interrupt routine

;*****
****
;High priority interrupt routine

HighInt:

        call   interrupt_routine ; read pressed button and put the right
                                   ; command in bit buffer to send it

        bcf    INTCON,INT0IF      ; clear interrupt flags for pins rb
        bcf    INTCON3,INT1IF     ; 0/1/2/4/5 to prevent getting multiple
        bcf    INTCON3,INT2IF     ; interrupts with one button press
        bcf    INTCON,RBIF

        retfie    FAST           ; pops the contents of the program counter
                                   ; previously pushed before going to the
                                   ; service routine, enables all interrupts,
                                   ; and returns control to the appropriate
                                   ; place in the main program.

;*****
****
;Low priority interrupt vector

        ORG    0x0018

        bra    LowInt          ; go to high priority interrupt routine

;*****
****
;Low priority interrupt routine

LowInt:

        bcf    T0CON,TMR0ON      ; stop timers 0 and 2 which are used for pwm
pulses
        bcf    T2CON,TMR2ON      ; and do not need to be used after sending
these
                                   ; pulses is finished

```

```

        bcf INTCON,TMR0IF ; clear interrupt flag of timer 0 which
triggers
                                ; the low interrupt, clearing prevents
multiple
                                ; interrupts for one overflow event

        bsf tmr_done,0

        retfieFAST

;*****
****

interrupt_routine:

        ; test interrupt flags to identify pressed button
        btfscl INTCON,RBIF      ; pin 4/5 pressed
        call send_rb_on_change
        btfscl INTCON,INT0IF     ; pin 0 pressed
        call send_rb_0
        btfscl INTCON3,INT1IF    ; pin 1 pressed
        call send_rb_1
        btfscl INTCON3,INT2IF    ; pin 2 pressed
        call send_rb_2

        return

; handle pressing pin 4 or 5, because both pins trigger on change interrupt
send_rb_on_change:
        btg skip_release,0; toggle skip_release flag to prevent sending
                                ; commands twice (by pressing and releasing
                                ; button), command is skipped when

skip_release<0>

                                ; is '0'
        btfss PORTB,4          ; check if pin 4 was pressed
        call send_rb_4
        btfss PORTB,5          ; check if pin 5 was pressed
        call send_rb_5
        bcf INTCON,RBIF      ; clear interrupt flag to prevent multiple
                                ; interrupts
        return

; send stop command with pin RB0
send_rb_0:
        bsf skip_release,0; set skip_release<0> to prevent skipping
this
                                ; command
        movlw b'00110101' ; move rc5 code for stop command to bit
buffer
        movwf bit_buffer_h
        movlw b'00110110'
        movwf bit_buffer_l
        bcf INTCON,INT0IF ; clear interrupt flag to prevent multiple

```



```

                                ; interrupts
                                return

; send forward command with pin RB1
send_rb_1:
    bsf skip_release,0; set skip_release<0> to prevent skipping
this
                                ; command
    movlw b'00110101' ; move rc5 code for forward command to bit
buffer
    movwf bit_buffer_h
    movlw b'00110100'
    movwf bit_buffer_l
    bcf INTCON,INT1IF ; clear interrupt flag to prevent multiple
                                ; interrupts
    return

; send play/pause command with pin RB2
send_rb_2:
    bsf skip_release,0; set skip_release<0> to prevent skipping
this
                                ; command
    movlw b'00110101' ; move rc5 code for pause command to bit
buffer
    movwf bit_buffer_h
    movlw b'00110000'
    movwf bit_buffer_l
    bcf INTCON,INT2IF ; clear interrupt flag to prevent multiple
                                ; interrupts
    return

; send mute command with pin RB4
send_rb_4:
    movlw b'00110101' ; move rc5 code for play command to bit
buffer
    movwf bit_buffer_h
    movlw b'00001100'
    movwf bit_buffer_l
    return

; send backward command
send_rb_5:
    movlw b'00110101' ; move rc5 code for backward command to bit
buffer
    movwf bit_buffer_h
    movlw b'00110010'
    movwf bit_buffer_l
    return

;*****
****
;Initialization routines

```

```

; configure port c
init_portc:
    clrf PORTC          ; reset port c and its latch
    clrf LATC
    movlw b'11111101' ; set pin RC1 to be output, all other RC pins
to be
    movwf TRISC          ; inputs
    return

; configure port b as input with interrupts
init_portb:
    clrf PORTB          ; reset port b and its latch
    clrf LATB

    bcf ADCON0,ADON      ; disable ad converter and reset its
configuration
    movlw 0xFF           ; bits
    movwf ADCON1

    movlw 0xFF           ; set all port b pins as inputs
    movwf TRISB

    movlw b'00111011' ; set pins with buttons to '1' to prevent
    movwf PORTB          ; triggering interrupt because buttons create
                        ; high state as default (active low buttons)

    bcf INTCON,INT0IF ; clear interrupt flags for pins rb
    bcf INTCON3,INT1IF ; 0/1/2/4/5 to prevent getting multiple
    bcf INTCON3,INT2IF ; interrupts with one button press
    bcf INTCON,RBIF

    bcf INTCON2,RBPU ; activate internal weak pull-ups for all pins
                    ; on port b

    bsf INTCON,INT0IE ; enable interrupts for interrupt 0/1/2 and
    bsf INTCON3,INT1IE ; on change interrupts on port b
    bsf INTCON3,INT2IE
    bsf INTCON,RBIE

    bcf INTCON2,INTEDG0 ; set interrupts 0/1/2 to trigger on falling
    bcf INTCON2,INTEDG1 ; edge
    bcf INTCON2,INTEDG2

    bsf INTCON,GIEH      ; activate all high priority interrupts
    bsf INTCON,GIEL      ; activate all low priority interrupts
    bsf INTCON,GIE        ; activate all interrupts

    bsf RCON,IPEN        ; enable interrupt priorities

    bsf INTCON3,INT1IP    ; set interrupt priority for interrupts

```

1/2

```

        bsf    INTCON3,INT2IP    ; and on change interrupt to high
priority
        bsf    INTCON2,RBIP

        return

; configure ports a and d as inputs, routines are taken from pic18f4525
manual
init_ports
    clrf PORTA                ; initialize port a by clearing output data
latches
    clrf LATA
    movlw 0x07                ; configure A/D for digital inputs
    movwf ADCON1
    movlw 0x07                ; configure comparators for digital input
    movwf CMCON
    movlw 0xFF                ; initialize data direction for all pins as
output
    movwf TRISA

    clrf PORTD                ; initialize port d by clearing output data
latches
    clrf LATD
    movlw 0xFFh               ; initialize data direction for all pins as
output
    movwf TRISD

    return

; configure pwm module
init_pwm:
    banksel PR2
    movlw 0x46                ; configure period time with 0x46 for 36 kHz
frequency
    movwf PR2

    movlw b'00010101'         ; configure duty cycle to be between 1/3 and
1/4 of
    movwf CCPR2L              ; the whole signal
    movlw b'00001100'         ; configure pwm mode and 2 lsbs of duty cycle
    movwf CCP2CON

    bcf PIR1,TMR2IF           ; clear interrupt flag to prevent multiple
                                ; interrupts
    bcf T2CON,T2CKPS1         ; set timer 2 clock prescaler value to '00'
    bcf T2CON,T2CKPS0

    movlw b'00001000'         ; configure timer 0 with 16 bit, internal
clock
    movwf T0CON                ; and prescaler value of '000'
    movlw 0xF7                ; set start value for timer 0
    movwf TMR0H
    movlw 0x52

```

```

    movwf TMR0L
    bcf INTCON,TMR0IF ; clear interrupt flag to prevent multiple
                        ; interrupts
    bsf INTCON,TMR0IE ; enable timer 0 interrupt
    bcf INTCON2,TMR0IP ; set interrupt priority to high
    bsf INTCON,GIE     ; enable all interrupts
    return

;*****
****
;Sending routines

; start half bit with low state by waiting for 889 us
tmr_delay:
    ;start timer 0 for pausing
    movlw 0xF7          ; set start value for timer 0 which counts to
overflow
    movwf TMR0H
    movlw 0x70
    movwf TMR0L
    bsf T0CON,TMR0ON ; start timer 0
    call delay_loop   ; wait for timer 0 low interrupt
    return

; start half bit with rectangle signal by running pwm for 889 us
pwm_delay:
    movlw 0xF7          ; set start value for timer 0 which counts to
overflow
    movwf TMR0H
    movlw 0x70
    movwf TMR0L
    movlw 0x00          ; set start value for timer 2 which counts
for each pulse
    movwf TMR2
    bsf LATC,1          ; set first pulse manually

    bsf T0CON,TMR0ON ; start timer 0 for 889 us delay and timer 2
for pwm
    bsf T2CON,TMR2ON

    call delay_loop   ; wait for timer 0 low interrupt

    return

;wait for timer 0 interrupt to end half bit
delay_loop:
    btfss tmr_done,0
    bra delay_loop
    bcf tmr_done,0
    return

```

```
; send logic '1' by first sending 889 us delay and then 32 pwm pulses
send_1:
    call tmr_delay
    call pwm_delay
    return

; send logic '0' by first sending 32 pwm pulses and then 889 us delay
send_0:
    call pwm_delay
    call tmr_delay
    return

;*****
****

;Start of main program

Main:
    call init_portc      ; configure ports and pwm module
    call init_portb
    call init_ports
    call init_pwm

    clrf bit_buffer_toggle

skip_cmd
    ; send when rb4 or rb5 pressed, skip when released
    movlw 0x00
    movwf skip_release

wait_cmd_loop

    bcf OSCCON,IDLEN    ; go to sleep until next button is pressed
    sleep
    nop
    nop
    nop

    movlw d'3'          ; send every command 3 times
    movwf cmd_ctr

    btfss skip_release,0 ; skip second command when button is
released
    bra skip_cmd        ; at rb4 and rb5 because interrupt is
triggered
                        ; on change, not only on falling edge

    btg bit_buffer_toggle,0 ; set toggle bit in bit_buffer
    btfsc bit_buffer_toggle,0
    bsf bit_buffer_h,3
    btfss bit_buffer_toggle,0
```

```

        bcf bit_buffer_h,3

send_cmd_loop

        movlw d'14'                ; send 14 bits
        movwf bit_ctr

        clrfsz tmr_done            ; reset flag that stops delay loop
during sending

        movf bit_buffer_h,W        ; send bit buffer from temporary
constants to keep value
        movwf bit_buffer_h_tmp
        movf bit_buffer_l,W
        movwf bit_buffer_l_tmp

        rlcfsz bit_buffer_l_tmp    ; skip first 2 bits because only 14
bits hold information
        rlcfsz bit_buffer_h_tmp
        rlcfsz bit_buffer_l_tmp
        rlcfsz bit_buffer_h_tmp

send_bit_loop

        rlcfsz bit_buffer_l_tmp    ; transfer the next bit that should be
sent in carry bit
        rlcfsz bit_buffer_h_tmp

        btfss STATUS,C            ; test carry bit to see if '0' or '1'
should be sent
        call send_0
        btfsc STATUS,C
        call send_1

        decfsz bit_ctr            ; send next bit if not all bits are
sent already
        bra send_bit_loop

        movlw 0xFF                ; load inner and outer delay counter to
create delay
        movwf rand_delay_ctr      ; between sending the command multiple
times
        movlw 0x40
        movwf rand_delay_ctr_outer

wait_rand_delay_outer              ; create delay of around 9 ms
wait_rand_delay
        nop
        nop
        decfsz rand_delay_ctr
        bra wait_rand_delay
        decfsz rand_delay_ctr_outer
    
```

```
        bra wait_rand_delay_outer

        decfsz cmd_ctr          ; send command again if it was not sent
3 times already
        bra send_cmd_loop

        bra wait_cmd_loop      ; go to sleep after sending 3 commands

;*****
****
;End of program
    goto $
    END
```

Quellen

Auf alle Quellen wurde am 01.07.24 zuletzt zugegriffen.

- [1] <https://ww1.microchip.com/downloads/en/devicedoc/39626e.pdf>
- [2] <https://www.datasheets.com/part-details/ncv1117st50-onsemi-34665214>
- [3] Microcontroller Theory and Applications by Rafiquzzaman
- [4] <https://www.ged-pcb-mcm.de/2018/05/13/gutes-pcb-layout-am-beispiel-des-quarzoszillators/>
- [5] <https://docs.rs-online.com/175a/0900766b80de9fc1.pdf>
- [6] <https://www.allaboutcircuits.com/technical-articles/pcb-layout-tips-and-tricks-use-a-ground-plane-whenever-possible/>
- [7] <https://www.opendcc.de/info/rc5/rc5.html>
- [8] <https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8bit-pic/peripherals/ccp/pwm/10-bit/>
- [9] <https://www.electronicwings.com/pic/pic18f4550-timer>

Abbildungsquellen

- Abb. 12 und 13: <https://www.opendcc.de/info/rc5/rc5.html>, letzter Zugriff am 01.07.24