Praktikum Digitaltechnik

IR - Remote

Benedikt Schnörr, Sebastian Pasinski

# Gliederung

[**Gliederung 2**](#_9b6hsne96upf)

[**Bit-Übertragung in der Theorie 3**](#_cm634halug80)

[**Bit-Übertragung im Code 4**](#_khtf6pzi2waq)

[**Interrupt Service Routine 7**](#_b1zjqgm4hdzj)

[**Finales PCB-Layout 9**](#_4117adxup1yr)

[**Bildquellen 10**](#_bnrkn5alu4yg)

[**Quellen 10**](#_bnortr7ryggc)

# 

# Bit-Übertragung in der Theorie

Die im letzten Bericht erwähnten Pulse mit einer Pulsdauer von 888,89 μs, die eine Rechteckspannung mit einer Frequenz von 36 kHz beinhalten, werden nun für die Bit-Übertragung genutzt. Ein Bit besteht dabei immer aus einer Low-Phase und einer Puls-Phase. Wie in Abb. 1 zu sehen, verändert die Reihenfolge dieser zwei Phasen den Zustand des Bits zwischen High und Low. [1]

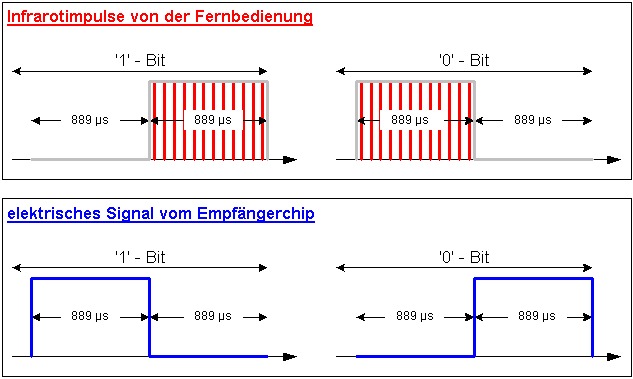


Abb. 1: Reihenfolge der Puls- und Ruhephasen und deren Bedeutung als Bit

Ein Befehl der Fernbedienung besteht aus insgesamt 14 Bits, die nacheinander gesendet werden. Die Bits sind in Abb. 2 dargestellt und haben die folgende Bedeutung: [2]

* 2 Start-Bits
* 1 Toggle-Bit, wodurch zwischen langem Tastendruck und erneutem Drücken unterschieden werden kann
* 5 Adress-Bits, um ein konkretes Gerät anzusprechen wie bspw. CD-Player, TV, Audio-Vorverstärker
* 6 Befehls-Bits

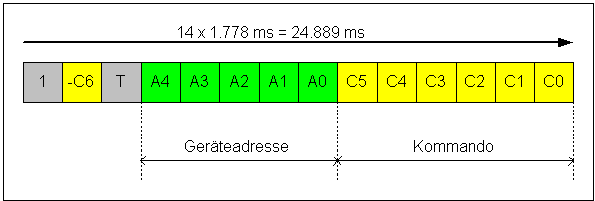


Abb. 2: Zeitliche Abfolge der Bits im RC5-Befehl

Der Code, um die Bits mit Ruhe- und Puls-Phase zu senden, ist im Folgenden dargestellt. Die Initialisierung von Ports und Timern ist zur Vereinfachung nicht berücksichtigt.

# Bit-Übertragung im Code

Der send\_bit-Loop iteriert durch alle zu sendenden Bits und ruft die passende Sender-Funktion auf, abhängig davon, ob das betrachtete Bild eine logische 0 oder 1 ist.

###### send\_bit:

###### rlcf bit\_buffer\_l ;get next bit from bit buffers

###### rlcf bit\_buffer\_h ;which contain command bits

###### btfss STATUS,C

###### call send\_0

###### call send\_1

###### decfsz bit\_ctr

###### bra send\_bit

Die entsprechenden send-Funktionen rufen dann die Ruhe- und Puls-Phase in der richtigen Reihenfolge auf.

###### send\_0:

###### call tmr\_delay

###### call pwm\_delay

###### return

###### send\_1:

###### call pwm\_delay

###### call tmr\_delay

###### return

###### 

Beim RC5-Code kann die PWM-Funktion des PIC18F4525 genutzt werden, um auf einfachem Weg ein Rechtecksignal zu erzeugen. Dazu wird das CCP-Modul genutzt, welches das Signal an Pin RC1 ausgibt. In der Konfiguration wird dazu die Frequenz des Signals im PR2-Register, sowie die Länge des Pulses (Duty Cycle) in den CCPR2L- und CCP2CON-Registern festgelegt. Die entsprechenden Berechnungen wurden bereits im letzten Bericht erläutert. Der folgende Code-Abschnitt zeigt die eingestellten Register.

###### ; configure pwm module

###### movlw 0x44 ; configure period time with 0x44

###### movwf PR2

###### movlw b'00000010' ; configure duty cycle with 0x11

###### movwf CCPR2L

###### movlw b'00111100' ; configure pwm mode and

###### movwf CCP2CON ; 2 lsbs of duty cycle

###### movlw b'00000000' ; configure postscale and prescale

###### movwf T2CON ; value and turn timer 2 off

###### movlw 0x00 ; reset timer 2 counter

###### movwf TMR2

Während die Rechteckspannung vom CCP2CON-Modul und Timer 2 erzeugt wird, muss auch die Sendedauer des Pulssignals reguliert werden. Wie bereits erwähnt, werden in der Sendephase jedes Bits 32 Pulse gesendet, dies entspricht einer Dauer von 888,89μs. Diese Sendedauer wird von Timer 0 beendet, der von einem entsprechend eingestellten Wert 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 folgenden Berechnungen. [3]

Da der Timer diese Differenz zählen muss, bis er überläuft wird nTimer von der maximale Zahl beim Überlauf der 16 Bit des Timer 0 abgezogen.

Die pwm\_delay-Routine konfiguriert Timer 2 für die Rechteckspannung und startet Timer 0 und Timer 2.

###### pwm\_delay:

###### movlw 0xF7 ;load timer 0 for half bit

###### movwf TMR0H

###### movlw 0x52

###### movwf TMR0L

###### movlw 0x00 ;load timer 2 for pwm

###### movwf TMR2

###### bsf LATC,1 ;set first pulse manually

###### bsf T0CON,TMR0ON ;start timers

###### bsf T2CON,TMR2ON

###### call delay\_loop

###### return

In der folgenden Routine wird die Ruhe-Phase mit Timer 0 gestartet.

###### tmr\_delay:

###### movlw 0xF7 ;load timer 0 for half bit

###### movwf TMR0H

###### movlw 0x52

###### movwf TMR0L

###### bsf T0CON,TMR0ON ;start timer 0 for pausing

###### call delay\_loop

###### return

Die folgende Delay-Routine wird ausgeführt, bis beim Überlauf von Timer 0 das nullte Bit von tmr\_done gesetzt wird. In diesem Moment wird die Routine und somit auch die Ruhe- oder Puls-Phase beendet.

###### delay\_loop: ;wait for interrupt to end half bit

###### btfss tmr\_done,0

###### bra delay\_loop

###### bcf tmr\_done,0

###### return

Die High-Interrupt-Routine wird beim Überlauf von Timer 0 ausgelöst und setzt wie bereits erwähnt das tmr\_done-Flag. Die Interrupt-Priorität ist in der Testphase auf einen High-Interrupt eingestellt, kann aber später auch auf einen Low-Interrupt umgestellt werden, damit der High-Interrupt für den Tastendruck verwendet werden kann.

###### HighInt:

###### bcf T0CON,TMR0ON ;stop timers

###### bcf T2CON,TMR2ON

###### bcf INTCON,TMR0IF ;reset interrupt flag

###### bsf tmr\_done,0

###### retfie FAST

Dieser Code kann also alle 14 Bits für einen Befehl senden. Der Befehl wird von einem anderen Teil des Programms festgelegt, in dem der gedrückte Taster ausgelesen wird. Der bereits beschriebene Programmteil wird dann nach dem Tastendruck ausgeführt.

# 

# 

# Interrupt Service Routine

Um die Funktionsweise der Schalter im Sleep-Mode zu gewährleisten und Software-Polling zu vermeiden,werden die Interrupts des PIC18F4525 verwendet. Die Pins 0-2 und 4-7 am Port B sind dazu in der Lage. Zunächst soll der Pin an RB0 ein Interrupt Request auslösen, wenn ein Schalter an diesem Eingang gedrückt wurde. Die ISR wurde unabhängig von der Initialisierung der PWM auf dem “picdem 2 plus demo board 2002” getestet.

Das Programm beginnt mit der High Priority Interrupt Routine, da INT0 immer von hoher Priorität ist. In der Routine soll nur das Flag zurückgesetzt werden und das Programm an die Stelle in der Hauptschleife springen, an der unterbrochen wurde. Wenn eine High Priority Interrupt ausgelöst wird, geht das Programm an die Stelle 0x0008 im Speicher und wird zur Routine geschickt.

###### 

###### ORG 0x0008

###### bra HighInt ; go to routine

###### 

###### ORG 0x0018

###### bra LowInt

###### 

###### HighInt:

###### bcf INTCON, INT0IF ;clear flag to avoid double interrupt

###### nop

###### nop

###### nop

###### nop

###### nop

###### retfie ; return to address saved on stackpointer

###### 

Im Hauptprogramm soll zunächst der Pin RB0 als digitaler Eingang festgelegt und dessen Interrupt-Einstellungen gesetzt werden. Diese sind als erstes das Flag zu clearen, dann Pullup-Widerstände zu setzen, den Interrupt zu befähigen und ihn bei einer fallenden Flanke zu setzen.

###### Main:

###### clrf PORTB

###### 

###### bcf ADCON0,ADON ; AD Converter disabled

###### 

###### movlw 0xFF

###### movwf TRISB

###### movlw 0xFF

###### movwf ADCON1 ; set PortB0 as digital Input

###### bcf INTCON, INT0IF ; clear Interrupt flag

###### bcf INTCON2, RBPU ; set Pull Ups on Portb

###### bsf INTCON, INT0IE ; Enable interrupt

###### bcf INTCON2, INTEDG0; Interrupt on falling edge

###### bsf INTCON, GIE ; Enable interrupts

###### 

In der Hauptschleife soll der Mikrocontroller dann auf ein Auslösen des Interrupts warten, bzw. setzen des Flags.

###### 

###### MLOOP

###### nop

###### nop

###### nop

###### 

###### nop

###### 

###### GOTO MLOOP

###### 

###### End

###### 

Im Programm wurde dann ein Breakpoint in der High Interrupt Service Routine gesetzt, um ein Auslösen zu identifizieren.

# Finales PCB-Layout

Es wurde ein endgültiges Layout in Rücksprache mit dem Labor angelegt. Im Vergleich zur letzten Version wurde kleine Fehler an den Verbindungen ausgebessert.

# 

Abb. 3: PCB-Layout - Farbig

# 

# Bildquellen

Abb. 1 und 2: <https://www.opendcc.de/info/rc5/rc5.html>, zuletzt aufgerufen am 02.06.24

# Quellen

[1] <https://www.opendcc.de/info/rc5/rc5.html>, zuletzt aufgerufen am 02.06.24

[2] <http://www.stefan-buchgeher.info/elektronik/rc5/rc5.html>, zuletzt aufgerufen am 02.06.24

[3] <https://www.electronicwings.com/pic/pic18f4550-timer>, aufgerufen am 23.05.24

# Anhang 1

PCB - Layout in Eagle Dark Mode.