Project Embedded Systems:
Infrarode Afstandsbediening

Semih Can Karakoç (695258)

6 maart 2024

Inhoudsopgave

1 Aanleiding en waarom

De aanleiding van de keuze van mijn eigenbijdrage is eigenlijk puur toeval. Ik wilde in eerste instantie eigenlijk iets doen met het *HCMS-29XX* scherm. Daarbij wilde ik een volledig functionele rekenmachine maken met remote controls.

Het probleem was dat we alleen één keer in de week konden testen, omdat we geen eigen afstandsbediening hadden. Ik had namelijk wel mijn eigen versterker al gebouwd met dezelfde functionaliteit als die van op school. Elke keer liep ik vast bij het programmeren van de controls en aansturing van de rekenmachine en ik kon dit elke keer thuis niet testen, wat mij behoorlijk teleurstelde.

Naar aanleiding van dit probleem nam ik het initiatief om mijn eigen afstandsbediening te bouwen en te programmeren. Ik liep tegen veel problemen aan, zoals het modelleren van de infrarode signalen die worden uitgezonden met de behorende frequentie.

Nadat ik het eindelijk werkend had gekregen, werkte mijn halve circuit niet meer. Wat bleek was dat de *breadboard* connecties (tulpen en *jumper-wires*) naar de knoppen waren gegaan.

Na veel *debuggen*, kwam ik tot de conclusie om het hele circuit in een elektrisch schema te zetten en deze na te bouwen op een printplaat. Nadat ik klaar was met het solderen deed de afstandsbediening zijn werk perfect! Ik heb daarna nog een indicatie LED, een aan/uit *switch* en een behuizing met batterijen toegevoegd.

Ik heb de afstandsbediening zo dusdanig gemaakt dat je eigen infrarode codes kunt samenstellen en kunt mappen aan een van de 9 knopjes naar keuze. Tijdens het schrijven van de code en testen, heb ik elke keer gecontroleerd of de signalen ook daadwerkelijk kloppen door middel van een *logic analyzer*.

Ik heb gekozen om de PIC16F1829 te gebruiken, omdat ik er nog 5 stuks van overhad en ze de perfecte formaat hadden voor op een afstandsbediening.

2 A. Welke gemotiveerde ontwikkelstappen in de hardware heb je genomen om tot je eindresultaat te komen? Neem VOOR ELKE HARDWARE-ONTWIKKELSTAP bovendien een bewijs van uitvoeren op.

Zoals ik al eerder heb gezegd, heb ik eerst onderzocht of het wellicht mogelijk is om daadwerkelijk een afstandsbediening te bouwen die kan functioneren onder mijn eisen. Na wat surfen op het internet zag ik dat mensen het deden met μC, zoals *Arduino's*, dus dacht ik dat het dan ook wel kon met een PIC16F1829. Ik heb het elektrisch schema van Martijn van de *"infrarood emitter"* (project opdracht 3) als voorbeeld genomen.

Hierin werden twee belangrijke componenten gebruikt, namelijk: de BC547B transistor gebruikt en een OS-5038F infrarode LED, waarbij deze LED in serie staat geschakeld met een 75Ω weerstand om de LED een langere levensduur te geven. Deze twee componenten en het deelcircuit heb ik overgenomen, omdat het letterlijk perfect werkt, zoals ik het nodig had. Voor PORTC van de PIC16F1829 waar de knoppen op zitten, heb ik ervoor gekozen om *pull-up resistors* te gebruiken, omdat PORTC een andere ingangslogica gebruikt ten opzichte van PORTB.

Verder bij de *indicator* LED heb ik gekozen om een 22kΩ weerstand te gebruiken om de stroom zo veel mogelijk te limiteren, want de LED hoeft alleen klein beetje licht uit te stralen om te laten zien dat de afstandsbediening aan staat. Ook heb ik de 100nF condensator overgenomen van martijn zijn elektrisch schema om de spanning over het circuit glad te houden.

Ik heb uiteindelijk nadat het circuit werkte met één knop, het circuit uitgebreid naar 9 knoppen op zowel PORTB als PORTC. Dit elektrisch schema heb ik gemaakt in *KiCAD* en heb ik ook gebouwd op mijn *breadboard*. Daarna heb ik de keuze gemaakt om het circuit zelf te solderen op een printplaat, omdat ik zeker wist dat het ging werken!

Dit pakte allemaal perfect uit en heb ik paar dagen erna ook een batterij houder toegevoegd met een aan/uit switch (dit was een optionele keuze en heb ik niet meegenomen in mijn elektrisch schema).

3 B. Welke gemotiveerde ontwikkelstappen in de software heb je genomen om tot je eindresultaat te komen? Neem VOOR ELKE SOFTWARE-ONTWIKKELSTAP bovendien een bewijs van uitvoeren op.

Het begon allemaal met dat we al op mijn andere breadboard jullie versterker bijna hadden na gebouwd, zodat ik thuis verder kon werken. Uiteindelijk moesten we ook het IR/remote gedeelte maken voor de code, dus wilde ik daar ook een replica van maken. We hebben gemakkelijk via seriële communicatie de IR codes gestuurd naar de versterker om de fysieke afstandsbediening na te bootsen. Dit werkte goed en op die manier hebben we de IR receiver code kunnen schrijven voor de versterker.

Nu moest ik nog mijn eigen bijdrage afmaken en daarvoor heb ik mijn oudere code gepakt en deze aangepast voor de IR-LED. Ik wilde bijna alle IR codes hard-coderen, maar daar had ik geen zin in, want dat kost veel tijd. De ontwikkel stap hierbij was dat ik de volgende functie heb geschreven om mijn tijd te besparen:

```
1   void send_ir_code(uint16_t ir_code)
2   {
3       /* Check for each bit if it is either an 1 or a 0: */
4       for (int i = 0; i < 12; i++)
5       {
6           if ((ir_code & 0b100000000000)) // AND operation with the IR-code and an 1.
7           {
8               hoog(); // If the answer is an one, then the bit is HIGH
9           }
10          else
11          {
12              laag(); // If the answer is a zero, then the bit is LOW
13          }
14          IR_LED = 0; // Make sure the IR-LED is off :)
15          ir_code <<= 1; // Bit-shift the IR code to the left
16      }
17  }
```

Dit stukje C code checkt letterlijk per iteratie van 12 keer of de IR-code die je meegeeft een ‘1’ of ‘0’ is. Zodra het een ‘1’ is, dan roep ik een andere functie op die de ‘1’ patroon genereert met de bijbehorende frequentie voor de IR LED. Hetzelfde doe ik ook voor als het een ‘0’ is, dan roep ik de `laag()` functie op. Vervolgens *shift* ik de IR-code per iteratie naar links, zodat de volgende *bit* gecheckt kan worden.

Met deze functie kun je gemakkelijk je eigen IR-codes genereren en pompen door de IR-LED.

Bovendien heb ik de originele IR-code die jullie afstandsbediening stuurt opgenomen met mijn logic analyzer om de *delay’s* te kunnen nabootsen.

Ik was aan het begin toen ik de `hoog()` en `laag()` functies aan het schrijven was best wel lang bezig, omdat ik niet echt wist hoe afstandsbedieningen precies werkten. Op

YouTube kwam ik uiteindelijk deze uitlegvideo tegen.

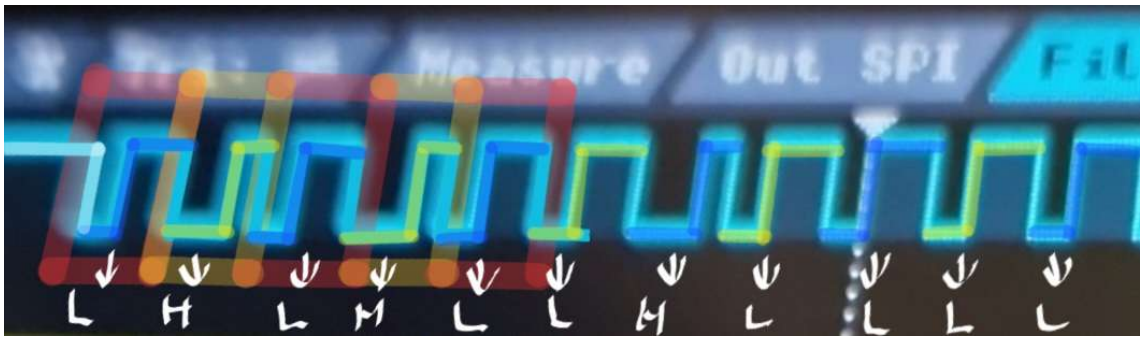
Onmiddellijk zodra ik de *thumbnail* van de video zag met die blauwe opname van de oscilloscoop wist ik meteen wat ik al die tijd al verkeerd deed. Later heb ik het probleem opgelost door de frequentie goed te berekenen en op de juiste manier toe te passen bij het alterneren van hoog naar laag bij het verzenden van IR-codes (bekijk de video als je niet snapt wat ik bedoel).

```
/* Generate a HIGH IR-signal */
void hoog()
{
    for (int i = 0; i < 21; i++) // 21 times high/low-signal at 1ms/26ms= 0.038 Hz (38 kHz)
    {
        __delay_us(26); // IR-LED AAN met frequentie van 38kHz
        IR_LED = 1;
        __delay_us(26); // IR-LED UIT met frequentie van 38kHz
        IR_LED = 0;
    }
    __delay_us(400); // Perfectly adjusted by using a Logic Analyzer
}
```

Hetzelfde principe pas ik bij de '0' signaal ook toe, alleen hier heb je minder iteraties in de for-loop, waardoor het signaal korter is:

```
/* Generate a LOW IR-signal */
void laag()
{
    for (int i = 0; i < 7; i++) // 7 times high/low-signal at 1ms/26ms= 0.038 Hz (38 kHz)
    {
        __delay_us(26); // IR-LED AAN met frequentie van 38kHz
        IR_LED = 1;
        __delay_us(26); // IR-LED UIT met frequentie van 38kHz
        IR_LED = 0;
    }
    __delay_us(1220); // Perfectly adjusted by using a Logic Analyzer
}
```

Die *delay* op het einde bij beide functies na de `for`-loep zorgt ervoor dat de IR-signaal netjes wordt afgemaakt met de tegenovergestelde signaal. Dus: als je hoog signaal hebt, dan duurt de eerste stuk lang en daarna kort, en bij laag signaal begin je met kort stukje en daarna een lang stuk, zie de afbeelding hieronder:



Figuur 1: Een foto van mijn logic analyzer. H = `hoog()` functie en L is `laag()` functie.

Nu werkte alles en kon ik mijn eigen IR-signalen genereren door middel van mijn programma. Ik wilde heel graag alles met *interrupts* doen, want dat is beter dan continu het *pollen* van knopjes (efficiënter). Echter heeft de PIC16F1829 haast maar één *interrupt* pen en de *Interrupt on Change* (IoC) heeft ook te weinig pennen, dus moest ik helaas wel pollen.

Ik heb dit op een simpele wijze aangepakt door in de `while(1)` loop met een `if/else` structuur continu te checken of de knoppen worden ingedrukt (pollen). Dit wordt om de elke 87.77 milliseconden gedaan door het programma om ongewenste inputs te voorkomen.

Daarnaast om de knoppen niet *continious* te laten functioneren (de *user* kan dan niet de knop ingedrukt houden) gebruikt het programma een variabele die '1' of '0' wordt gezet na één indruk van de knop. Hiermee kan het programma onderscheid maken tussen *continious* en *not continious* knoppen, net zoals in de echte afstandsbediening.

4 Source code van de afstandsbediening

```
/*****************************************************************
 * +-------------------------------------------------------+ *
 * | * PIC    : 16F1829                                    | *
 * | * Board  : PICkit 5                                   | *
 * | * Date   : A long time ago, in a galaxy far away      | *
 * | * Author : broodjesemih (Semih Can Karakoc)           | *
 * +-------------------------------------------------------+ *
 *****************************************************************/

#include <xc.h>                  // PIC hardware mapping
#define _XTAL_FREQ      8000000  // Used by the XC8 delay_ms(x) macro (8 MHz)

// For acceptable values use both datasheet and this file:
// /opt/microchip/xc8/v<compiler_version>/docs/pic_chipinfo.html

// CONFIG WORD 1
#pragma config FOSC     = 0x4   // Frequency OSCillator Selection
#pragma config WDTE     = 0x0   // Watch Dog Timer Enable
#pragma config PWRTE    = 0x1   // Power-up Timer Enable
#pragma config MCLRE    = OFF   // NOT Master CleaR Enable  (RA3/!MCLR/VPP)
#pragma config CP       = OFF   // NOT Code Protection
#pragma config CPD      = OFF   // NOT Data Code Protection
#pragma config BOREN    = ON    // Brown-out Reset Enable
#pragma config CLKOUTEN = OFF   // NOT Clock Out Enable
#pragma config IESO     = OFF   // Internal External Switchover
#pragma config FCMEN    = OFF   // Fail-Safe Clock Monitor

// CONFIG WORD 2 @ flash program memory location 0x8008
#pragma config WRT      = OFF   // Flash Memory Self-Write Protection
#pragma config PLLEN    = OFF   // PLL Enable bit
#pragma config STVREN   = OFF   // Stack Overflow/Underflow Reset Enable
#pragma config BORV     = LO    // Brown-out Reset Voltage Selection
//#pragma config DEBUG    = OFF   // NOT In-Circuit Debugger Mode -> Pre-processor short-circuits !
#pragma config LVP      = OFF // Low-voltage programming

/* Defines */
#define IR_LED LATCbits.LATC6 // IR LED
#define BUTTON PORTAbits.RA2  // Knopje

/* IR-code list: */
#define INPUT1          0b010010010000 // !continious (not)
#define INPUT2          0b010100001000 // continious  (yes)
#define INPUT3          0b010010100000 // !continious (not)
#define INPUT4          0b010010001000 // !continious (not)
#define VOLUME_UP       0b010100100000 // continious  (yes)
#define VOLUME_DOWN     0b010100010000 // continious  (yes)
#define MUTE            0b010010000010 // !continious (not)
#define DISPLAY         0b010010000100 // !continious (not)
#define LOOP            0b010100000010 // continious  (yes)

/* (Possible) IR-code combinations: */
#define LOOP_INPUT2 (INPUT2 | LOOP)
#define LOOP_INPUT2_DOWN (INPUT2 | LOOP | VOLUME_DOWN)
#define CONFIRM (INPUT2 | LOOP | VOLUME_UP)
#define DISPLAY_B_UP (VOLUME_UP | LOOP)
#define DISPLAY_B_DOWN (VOLUME_DOWN | LOOP)

/* Variables: */
int not_con_i1 = 0;     // !Con for input 1
int not_con_i3 = 0;     // !Con for input 3
int not_con_i4 = 0;     // !Con for input 4
int not_con_disp = 0;   // !Con for display button
int not_con_mute = 0;   // !Con for mute button
int not_con_litwo = 0;  // !Con for combi (ADC) button

/* Functions */
void pic_init(void);
void init_gpio(void);
void init_osc(void);
void hoog();    // Generate HIGH IR signal
void laag();    // Generate LOW IR signal
/* Generates and sends the IR code by combining IR signals: */
void send_ir_code(uint16_t ir_code);

/* THE main */
void main(void)
{
    pic_init(); // Initialize everything
    while (1)   // Enter infinte LOOP
    {
        /* PORTB: INPUT SWITCHES */
```

```
82          // Input 1 (!con.)
83          if (PORTBbits.RB7 == 1 && not_con_i1 == 0)
84          {
85              send_ir_code(INPUT1);
86              send_ir_code(INPUT1);
87              not_con_i1 = 1;
88          }
89          if (PORTBbits.RB7 == 0)
90              not_con_i1 = 0;
91
92              // Input 2 (con.)
93          else if (PORTBbits.RB4 == 1)
94          {
95              send_ir_code(INPUT2);
96                 send_ir_code(INPUT2);
97          }
98
99          if (not_con_litwo == 1)
100             not_con_litwo = 0;
101
102         // Input 3 (!con.)
103         if (PORTBbits.RB6 == 1 && not_con_i3 == 0)
104         {
105             send_ir_code(INPUT3);
106             send_ir_code(INPUT3);
107             not_con_i3 = 1;
108         }
109         if (PORTBbits.RB6 == 0)
110             not_con_i3 = 0;
111
112         // Input 4 (!con.)
113         if (PORTBbits.RB5 == 1 && not_con_i4 == 0)
114         {
115             send_ir_code(INPUT4);
116             send_ir_code(INPUT4);
117             not_con_i4 = 1;
118         }
119         if (PORTBbits.RB5 == 0)
120             not_con_i4 = 0;
121
122
123         /* PORTC: REST OF THE SWITCHES: */
124         // DISPLAY
125         if (PORTCbits.RC5 == 0 && not_con_disp == 0) //&& not_con_disp == 0)
126         {
127             send_ir_code(DISPLAY);
128             send_ir_code(DISPLAY);
129             not_con_disp = 1;
130         }
131         if (PORTCbits.RC5 == 1)
132             not_con_disp = 0;
133
134         // Volume: UP
135         if (PORTCbits.RC7 == 0)
136         {
137             send_ir_code(VOLUME_UP);
138             send_ir_code(VOLUME_UP);
139         }
140
141         // Volume: DOWN
142         if (PORTCbits.RC4 == 0)
143         {
144             send_ir_code(VOLUME_DOWN);
145             send_ir_code(VOLUME_DOWN);
146         }
147
148         // MUTE
149         if (PORTCbits.RC2 == 0 && not_con_mute == 0)
150         {
151             send_ir_code(MUTE);
152             __delay_ms(10);
153             send_ir_code(MUTE);
154             not_con_mute = 1;
155         }
156         if (PORTCbits.RC2 == 1)
157             not_con_mute = 0;
158
159             // LOOP
160         else if (PORTCbits.RC3 == 0)
161         {
162             send_ir_code(LOOP);
163             send_ir_code(LOOP);
164         }
```

```
165
166         // Combination SWITCHES:
167         // ADC control ON/OFF
168         if (PORTCbits.RC3 == 0 && PORTBbits.RB4 == 1 && not_con_litwo == 0)
169         {
170             send_ir_code(LOOP | INPUT2);
171             send_ir_code(LOOP | INPUT2);
172             not_con_litwo = 1;
173         }
174
175         // Polling Delay (Set to realistic - Measured with a logic analyzer)
176         __delay_ms(88); // Real delay is around: 87.7714286 milliseconds
177     }
178 }
179
180 /* Generate an IR-signal pattern for the IR emitter */
181 void send_ir_code(uint16_t ir_code)
182 {
183     /* Check for each bit if it is either an 1 or a 0: */
184     for (int i = 0; i < 12; i++)
185     {
186         if ((ir_code & 0b100000000000)) // AND operation with the IR-code and an 1.
187         {
188             hoog(); // If the answer is an one, then the bit is HIGH
189         }
190         else
191         {
192             laag(); // If the answer is a zero, then the bit is LOW
193         }
194         IR_LED = 0; // Make sure the IR-LED is off :)
195         ir_code <<= 1; // Bit-shift the IR code to the left
196     }
197 }
198
199 /* Generate a HIGH IR-signal */
200 void hoog()
201 {
202     for (int i = 0; i < 21; i++) // 21 times high/low-signal at 1ms/26ms= 0.038 Hz (38 kHz)
203     {
204         __delay_us(26);
205         IR_LED = 1;
206         __delay_us(26);
207         IR_LED = 0;
208     }
209     __delay_us(400); // Perfectly adjusted by using a Logic Analyzer
210 }
211
212 /* Generate a LOW IR-signal */
213 void laag()
214 {
215     for (int i = 0; i < 7; i++) // 7 times high/low-signal at 1ms/26ms= 0.038 Hz (38 kHz)
216     {
217         __delay_us(26);
218         IR_LED = 1;
219         __delay_us(26);
220         IR_LED = 0;
221     }
222     __delay_us(1220); // Perfectly adjusted by using a Logic Analyzer
223 }
224
225 /* Start-of Init functions ----------------------------------------- */
226 void pic_init(void)
227 {
228     init_osc(); // Oscillator initialization
229     init_gpio(); // GPIO initialization
230 }
231
232 void init_osc(void)
233 {
234     // System Clock Select (SCS)
235     OSCCONbits.SCS = 0b00;    // Clk determined by FOSC in Conf-1
236
237     // Internal Resistor-Capacitor Frequency select (IRCF)
238     OSCCONbits.IRCF = 0b1110; // 8 MHz clock speed
239
240     // Software Phase-Locked Loop ENable (SPLLEN)
241     OSCCONbits.SPLLEN = 0b0;  // 4 x Phase-Locked Loop disabled
242 }
243
244 void init_gpio(void)
245 {
246     /* REGISTER B: */
247     TRISB = 1;  // Configure PORT B as INPUT
```

```
248     ANSELB = 0; // Disable ANALOG for PORT B
249
250
251     /* REGISTER C: */
252     TRISCbits.TRISC6 = 0; // Set Register C6 as OUTPUT (IR_LED)
253     ANSELC = 0;           // Analog OFF for PORT C
254
255     /* Input buttons: */
256     TRISCbits.TRISC7 = 1; // Pin 9  - Volume UP
257     TRISCbits.TRISC5 = 1; // Pin 5  - Display
258     TRISCbits.TRISC4 = 1; // Pin 6  - Volume DOWN
259     TRISCbits.TRISC3 = 1; // Pin 7  - Loop
260     TRISCbits.TRISC2 = 1; // Pin 14 - Mute
261
262     IR_LED = 0;           // TURN IR LED OFF
263 }
264 /* End-of Init functions ------------------------------------------ */
```

5 EXTRA Video links

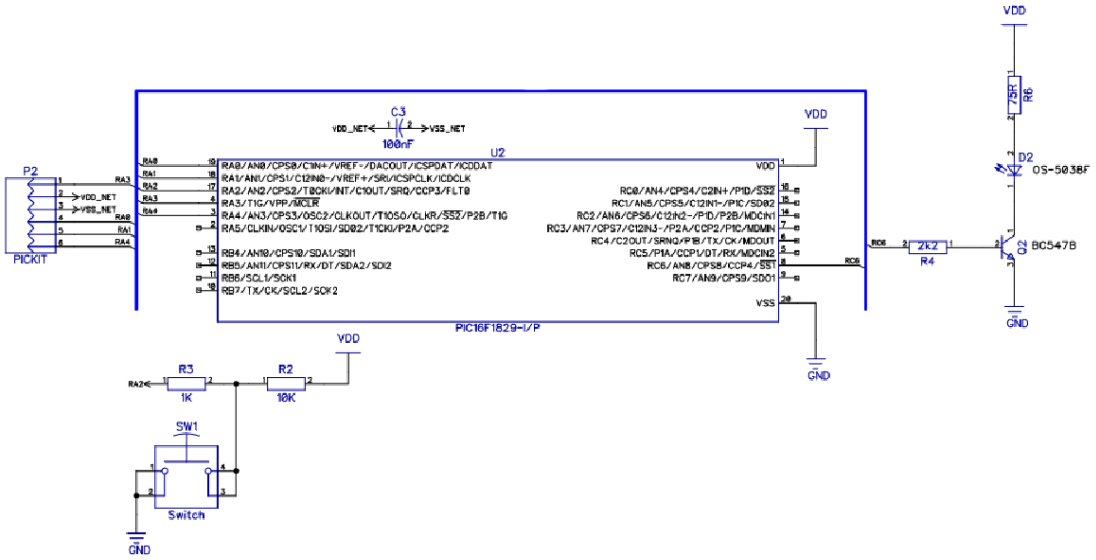
Video 1

Video 2

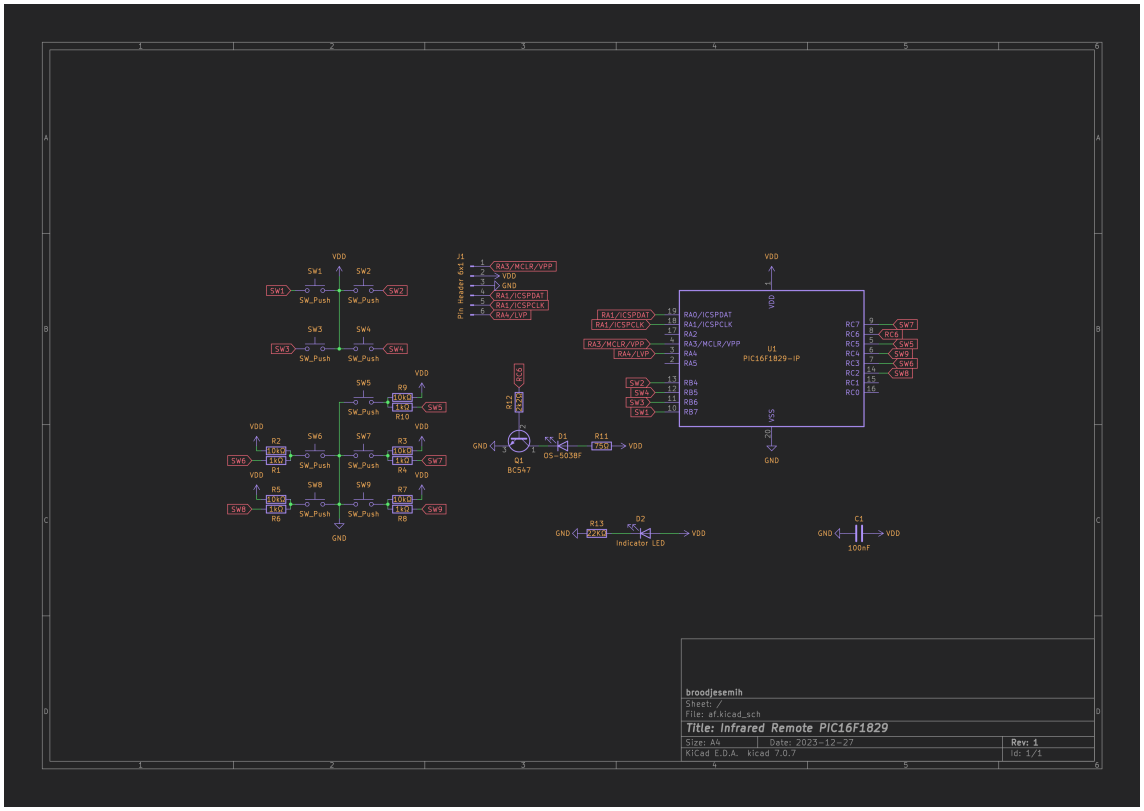
Video 3

Video 4

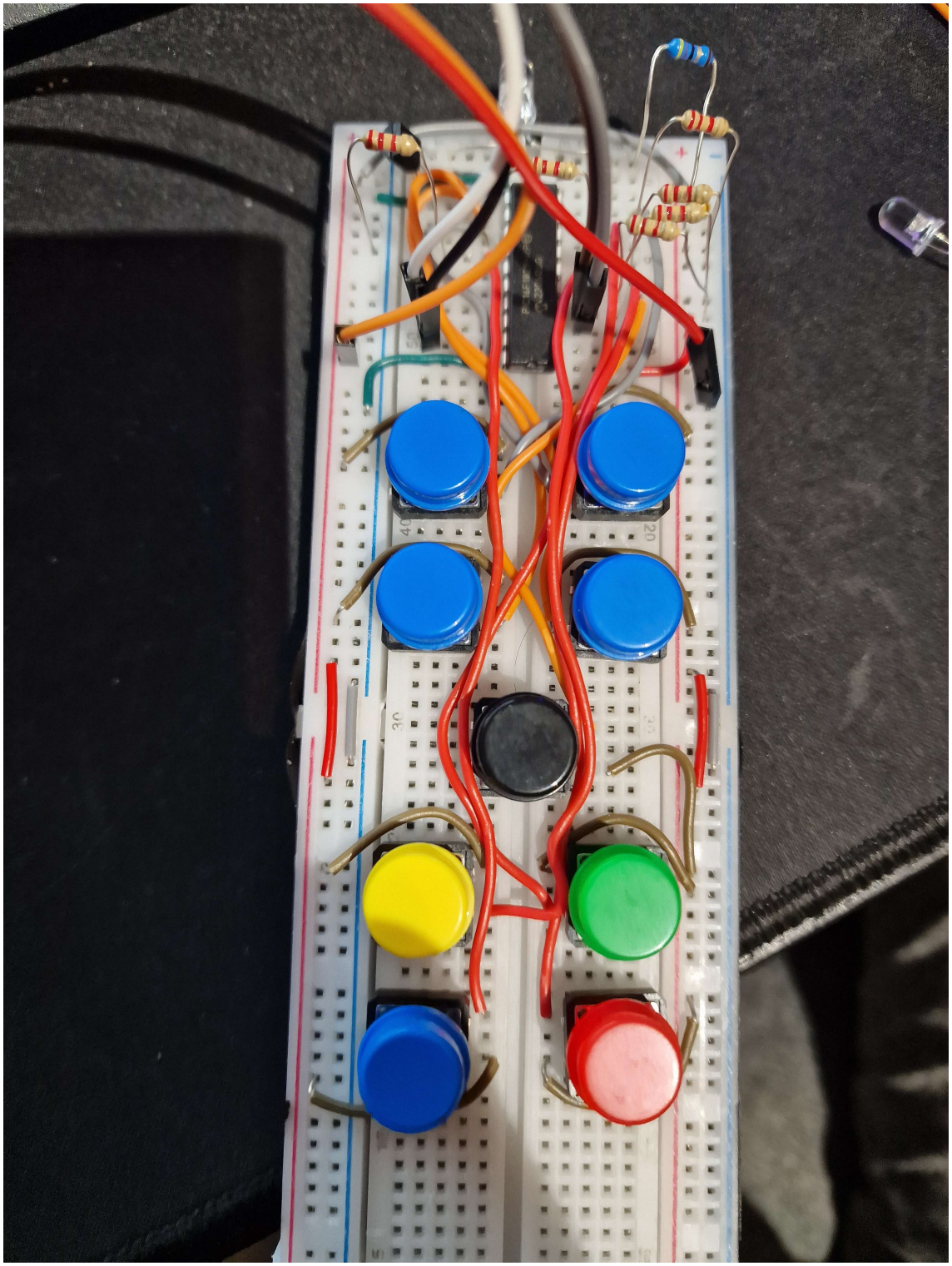
6 Bijlagen en Afbeeldingen



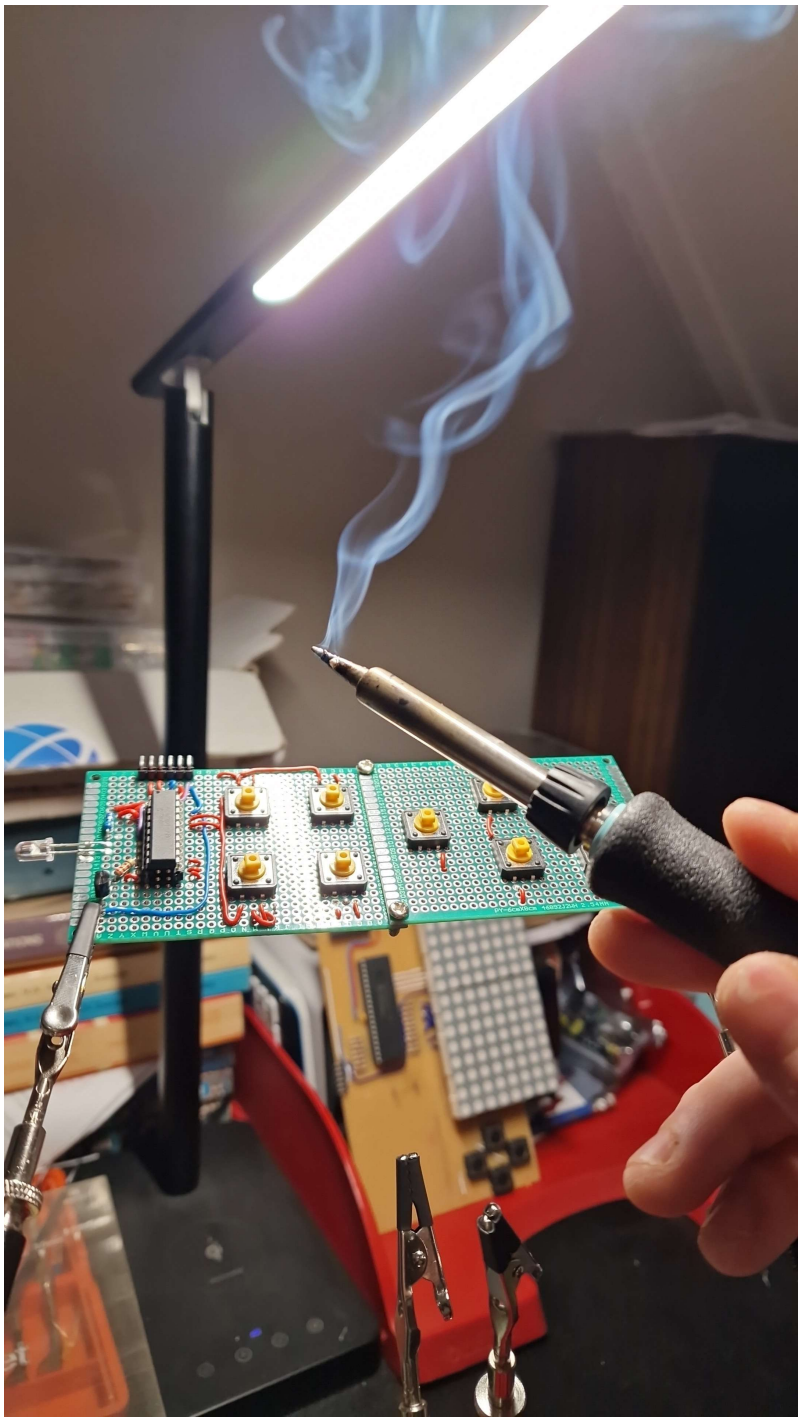
Figuur 2: Een schermafdruk van Martijn zijn ontworpen elektrisch schema voor de afstandsbediening opdracht.



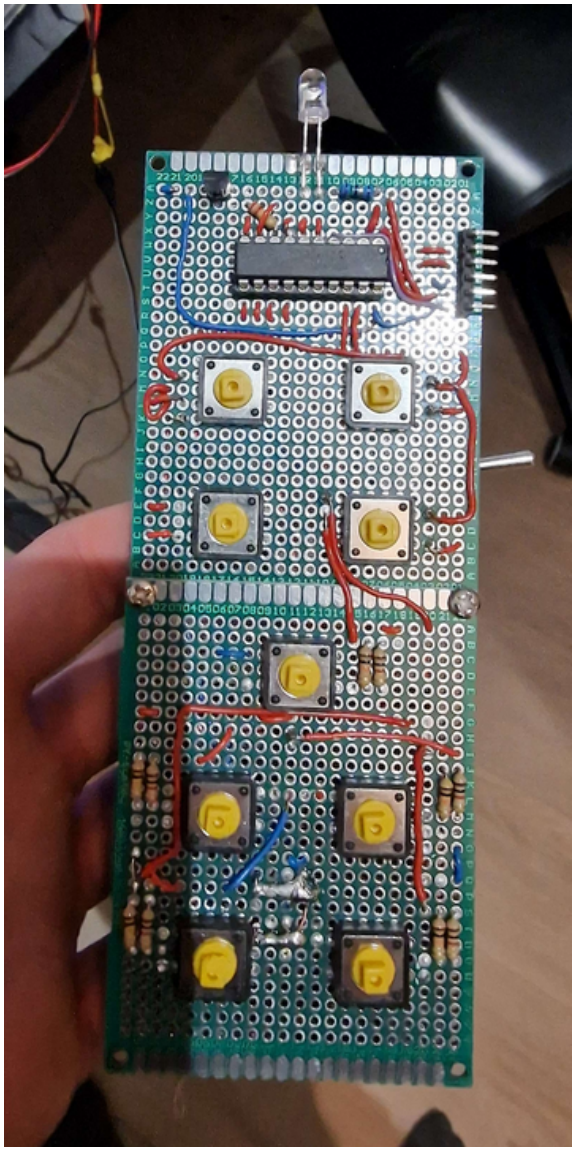
Figuur 3: Een schermafdruk van mijn ontworpen elektrisch schema voor de afstandsbediening.



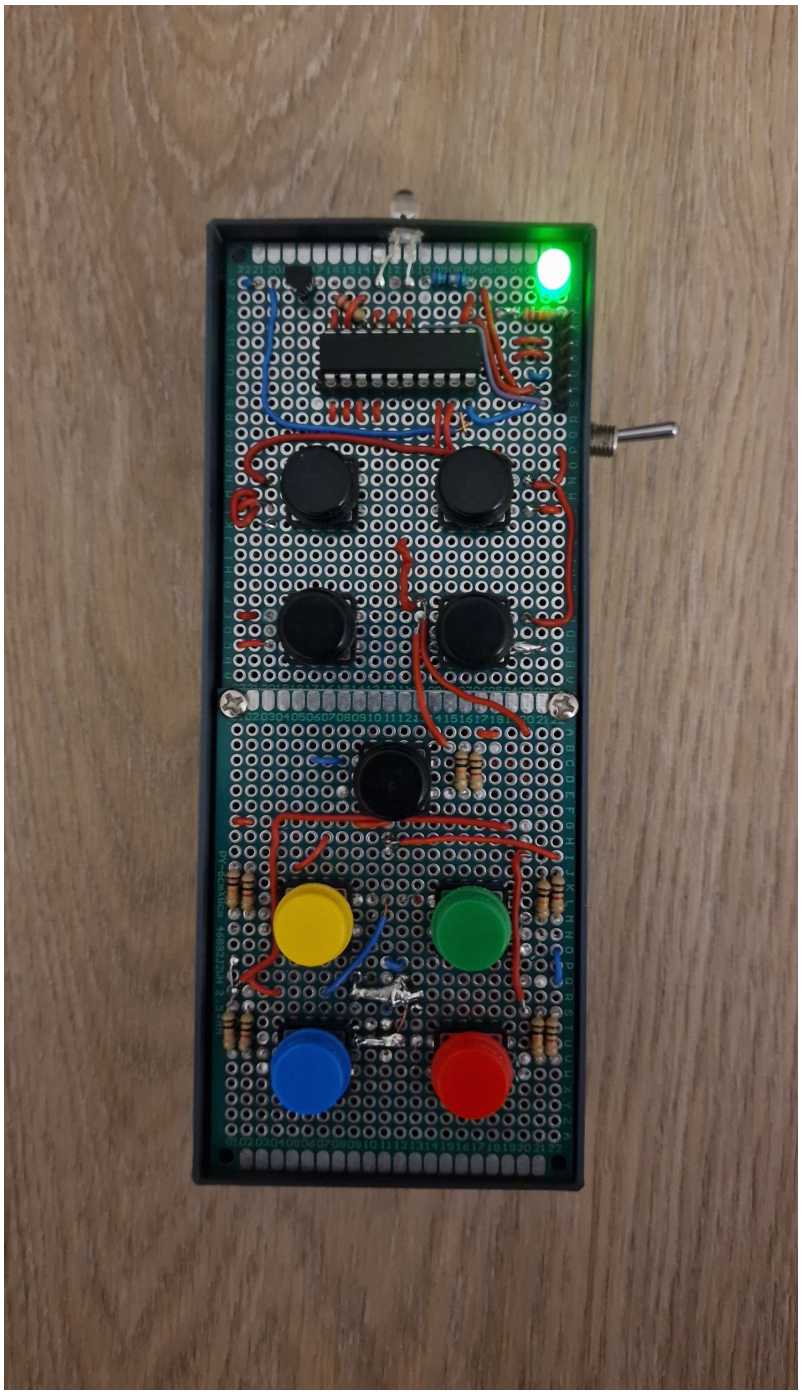
Figuur 4: Een foto van mijn prototype voor de afstandsbediening op een breadboard.



Figuur 5: Een foto terwijl ik aan de afstandsbediening' printplaat het solderen ben (half af).



Figuur 6: Een foto van afstandsbediening' printplaat toen ik net klaar was met solderen (zonder indicator LED, want die heb ik later toegevoegd).



Figuur 7: Een (vanaf bovenaanzicht) foto van het eindproduct: de afstandsbediening.

Figuur 8: Andere foto van het eindproduct: de afstandsbediening.