

Academiejaar 2015-2016

Thomas More-Campus De Nayer

Wireless-Jukebox

Eindverslag Project 2

Voorwoord

Ik ben Olivier Van den Eede.

olivier.vandeneede@student.thomasmore.be

Voor mijn opleiding Electronica-Ict in de 2de fase aan Thomas More De Nayer heb ik een project gemaakt, namelijk de Wireless-Jukebox.

De reden voor het maken van dit project is omdat ik graag wilde werken met een 32bit microcontroller, een SD-kaart en een wifi-module. Dit project beantwoord aan al deze criteria.

Inhoudstafel

Voorwoord	2
Inhoudstafel	3
1 Introductie	4
1.1 Inleiding	4
1.2 Samenvatting	4
1.3 Probleem en doelstelling	4
1.4 Oplossingsstrategie	4
2 Bespreking blokschema	5
2.1 Microcontroller	5
2.2 Pwm + filter	5
2.3 Esp8266	6
2.4 Gsm/pc	7
3 Samenhang	8
3.1 Volledig schema pcb	8
3.2 Layout pcb	9
3.3 Foto's	10
4 Broncode	12
4.1 Esp8266	12
4.1.1 Titels opslaan	12
4.1.2 Placeholders vervangen	12
4.1.3 Opzetten en verwerken http webserver	13
4.1.4 Opvragen meest gestemde lied	14
4.2 Stm32	15
4.2.1 Sample timer interrupt	15
4.2.2 Begin programma	16
4.3.3 getTitels (wav.c)	17
4.3.4 startPwm (wav.c)	17
4.3.5 randomNummers (esp.c)	18
4.3.6 sendSongsToEsp (esp.c)	19
4.3.7 Main while	20
4.3.8 getData (wav.c)	22
5 Conclusie	23
5.1 Reflectie	23
5.2 Kostprijs	23

Bijlage 1: Datasheet features

Bijlage 2: Projectvoorstel

Bijlage 3: Plan van aanpak

Bijlage 4: Logboek

Bijlage 5: Flowcharts

1 Introductie

1.1 Inleiding

Dit verslag gaat over de Wireless Jukebox, dit is een module die aan een stereo installatie gekoppeld wordt in een ruimte waar zich een aantal mensen bevinden. Als deze mensen verbonden zijn met het wifi netwerk, kunnen ze naar een website op de wireless jukebox surfen en daar stemmen op het liedje dat als volgend afgespeeld zal worden.

1.2 Samenvatting

De module maakt gebruik van een µc met een sdio interface voor de communicatie met een sd-kaart waarop alle liedjes opstaan. De module is met een wifi netwerk verbonden via een esp-8266 wifi module waarop ook de website draait. Voor het afspelen van muziek wordt er gebruik gemaakt van een PWM output die verbonden is aan een jack-connector.

1.3 Probleem en doelstelling

Het initiële idee voor het project was om gebruik te maken van een 8bit-µc en te communiceren met een sd-kaart via spi. Maar omdat van gebruiksgemak en capaciteitsproblemen met spi-compatibele sd-kaarten was dit onhaalbaar.

Bij het bedenken was het plan om een webserver op te zetten met een php website en de esp-module data te laten ophalen van een externe webserver. Omwille van de mogelijkheden van de wifi-module en het gebruiksgemak is er besloten om de webserver op de wifi-module zelf te houden. Dit geeft tal van voordelen omdat nu ook de µc minder belast is met het opvragen van data.

Voor het afspelen van de muziek zijn er 2 mogelijkheden, ofwel afspelen via een DAC of via PWM, het was nog niet 100% duidelijk welke manier het beste was bij het starten van het project.

Om dit alles te bekomen zou ik gebruik maken van het stm32F4 discovery board omdat hierop een dac met een jack aansluiting voorzien was, maar omdat van en slechte pinout op het discovery board was dit echter niet mogelijk. De pinnen van de sdio en de dac voor de muziek overlappen, en het was dus niet mogelijk om deze samen te gebruiken op het board.

1.4 Oplossingsstrategie

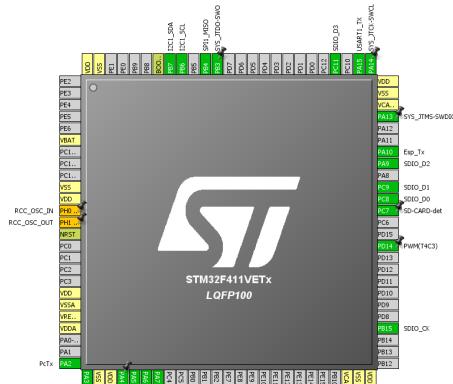
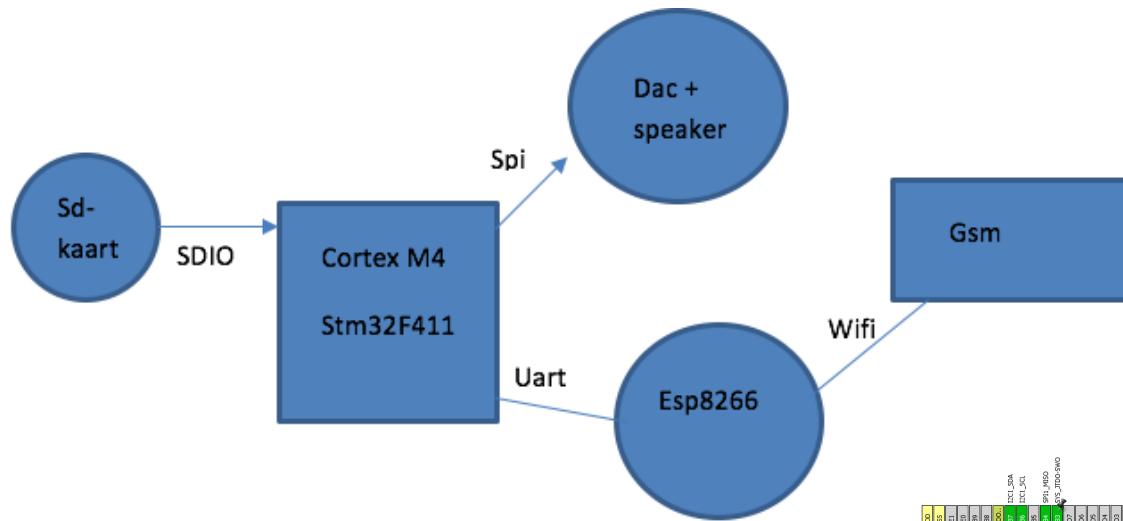
Voor de sd interface is er nu gekozen om een 32-bit µc te gebruiken met een sdio interface.

Om het probleem met de dac en de sdio interface op te lossen heb ik besloten om zelf een custom pcb te maken waarop de pinout wel klopt.

Bij het ontwerpen van de pcb is er gekozen om beide een DAC en PWM te ondersteunen op het prototype. Hierdoor is er meer vrijheid als het niet goed zou werken met de DAC of de PWM.

Uiteindelijk is er gekozen om alles te doen via pwm.

2 Bespreking blokschema



2.1 Microcontroller

Als microcontroller heb ik gekozen voor een cortex M4 van stm32. De reden voor het kiezen van de stm32f411 is omdat deze beschikt over een sdio interface en ondersteuning heeft voor het fatfs filesystem. Verder beschikt deze controller over een spi interface, om eventueel de dac aan te sturen. Ook beschikt het over meerdere uarts, waarvan er 1 gebruikt wordt voor communicatie met de wifi module, en 1 voor communicatie met de computer voor debugging. De controller was beschikbaar op het stm32f4 discovery board, wat gebruikt is voor alle ontwikkelingen van het project.

Voor de custom pcb aangekomen was.

2.2 Pwm + filter

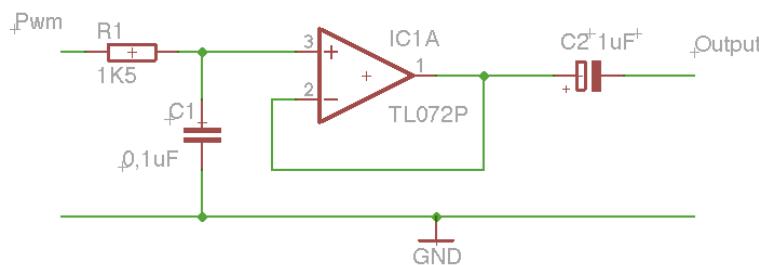
Voor het afspelen van muziek heb ik gekozen om dit via pwm te doen, Dit heb ik gedaan omdat de simpel te configureren timers van stm.

De muziek wordt afgespeeld uit wav-files met een sample frequentie van 44100Hz en een resolutie van 8bit. Om dit af te spelen heb ik een pwm output en een timer nodig. De pwm output frequentie is bepaald op 187500Hz, dit is de timerfrequentie van 48MHz / 8bit. De sample timer moet vastgelegd worden 44100Hz omdat de wav file zo opgebouwd is.

Om de timer in te stellen op 44100Hz is volgende berekening gebruikt:

prescaler: $48.000.000/4 = 12\text{MHz} \Rightarrow$ prescaler is 4-1 => 3

timer count: $X \cdot 12\text{MHz} = 44100\text{Hz} \Rightarrow$ timer count is 272 steps



Na de pwm output moet er een filter komen, de filter is berekend op 1KHz door een $1,5\text{k}\Omega$ weerstand en een $0,1\mu\text{F}$ condensator. En een opamp om te bufferen samen met een condensator om de cd-signalen af te halen.

Fig2: Output filter en buffer

Aan de output van de pwm is het signaal nog een DC blokgolf, om dit om te zetten naar een analoog signaal moest er nog een filter geplaatst worden aan de output. Aan het einde na de buffer en filter zit nog een condensator in serie die dient om de dc-component van het signaal eraf te filteren om de luidspreker erachter te beschermen.

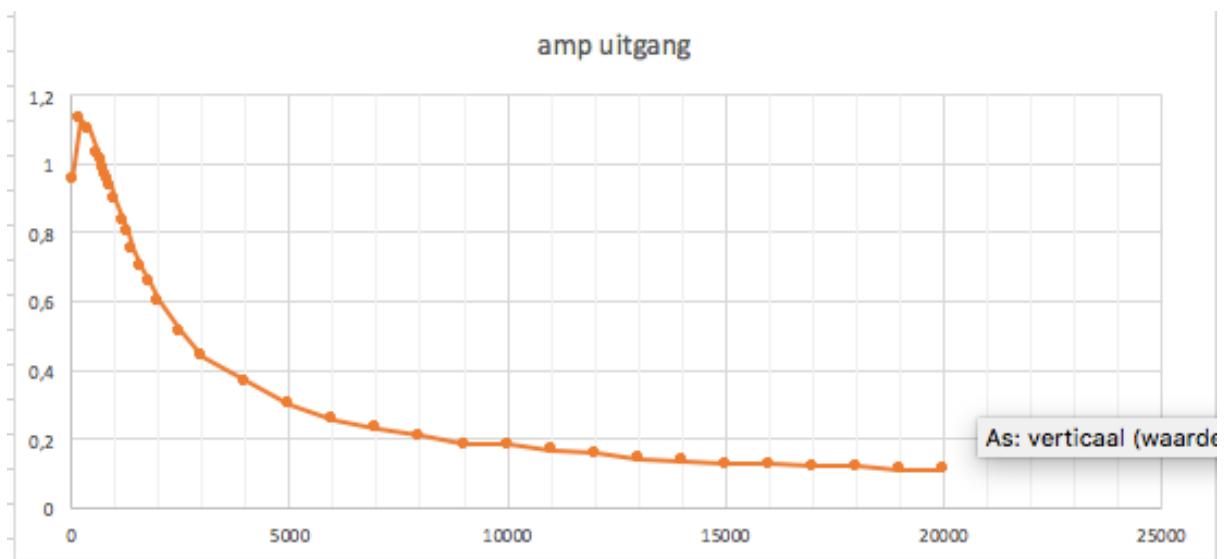


Fig3: Meetresultaat filter.

Figuur3 is het resultaat van het doormeten van de filter. Het geeft de waarde van de amplitude van een sinus opgemeten bij het aflopen van frequenties van 20Hz tot 20kHz. Op de figuur is te zien dat zoals berekend, de filter frequentie ligt rond de 1kHz.

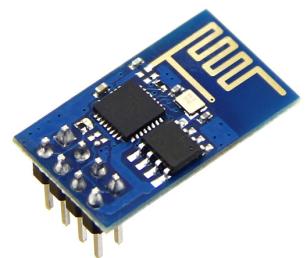
2.3 Esp8266

Voor de communicatie met gsm's of computers is er gekozen om gebruik te maken van de esp8266 wifi-module. Het is mogelijk om met deze module te communiceren via een uart interface. Deze module beschikt over een volledige tcp/ip stack en kan volledig zelf als server op het wifi netwerk draaien.

Voor het project is er gekozen om op de esp-module een webserver met een html pagina te draaien waarop gebruikers kunnen stemmen op het volgende liedje.

Om dit te bekomen heb ik gekozen om gebruik te maken van het nodeMCU platform, dit is een custom firmware voor de esp module waardoor deze geprogrammeerd kan worden in de programmeertaal lua. NodeMCU heeft tal van voordelen omdat dit een hogere programmeertaal is, waardoor het ontwikkelen van een applicatie veel sneller en simpeler is.

De microcontroller levert via uart de titels van liedjes aan, lua zet deze in de html pagina en wacht tot er clients verbinden. Als er een client verbonden is kunnen deze stemmen op de liedjes en het lua script houdt een telling bij. De microcontroller kan nadat dan het meest gestemde liedje opvragen.



2.4 Gsm/pc

Op de gsm of pc wordt een webpagina weergegeven, deze pagina is een simpele html pagina samen met een online stylesheet van het bootstrap framework, hierdoor wordt de website altijd juist weergegeven, of het nu op een mobiel apparaat of een laptop is.

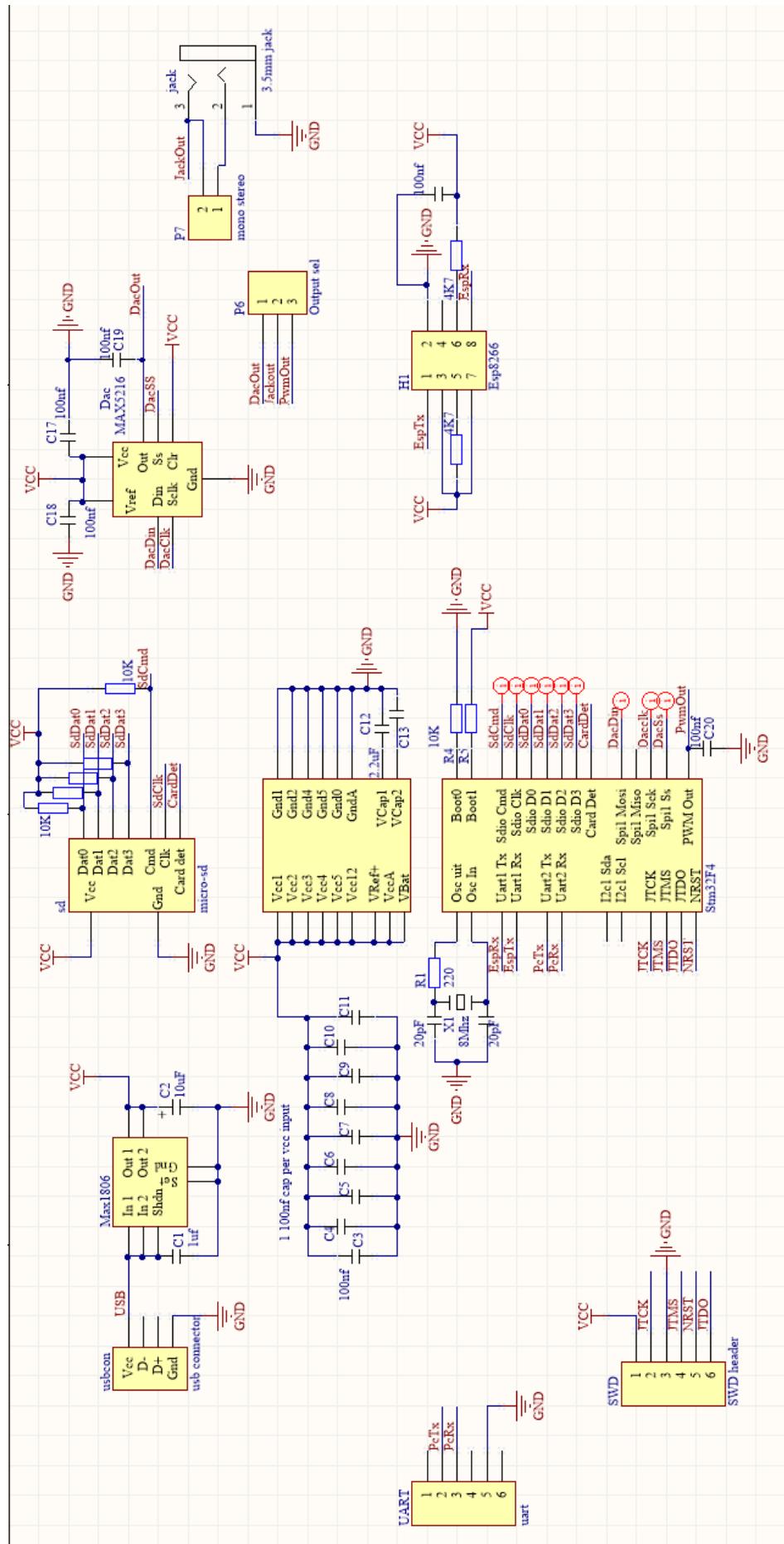
Wireless jukebox

Wat wordt het volgende liedje?

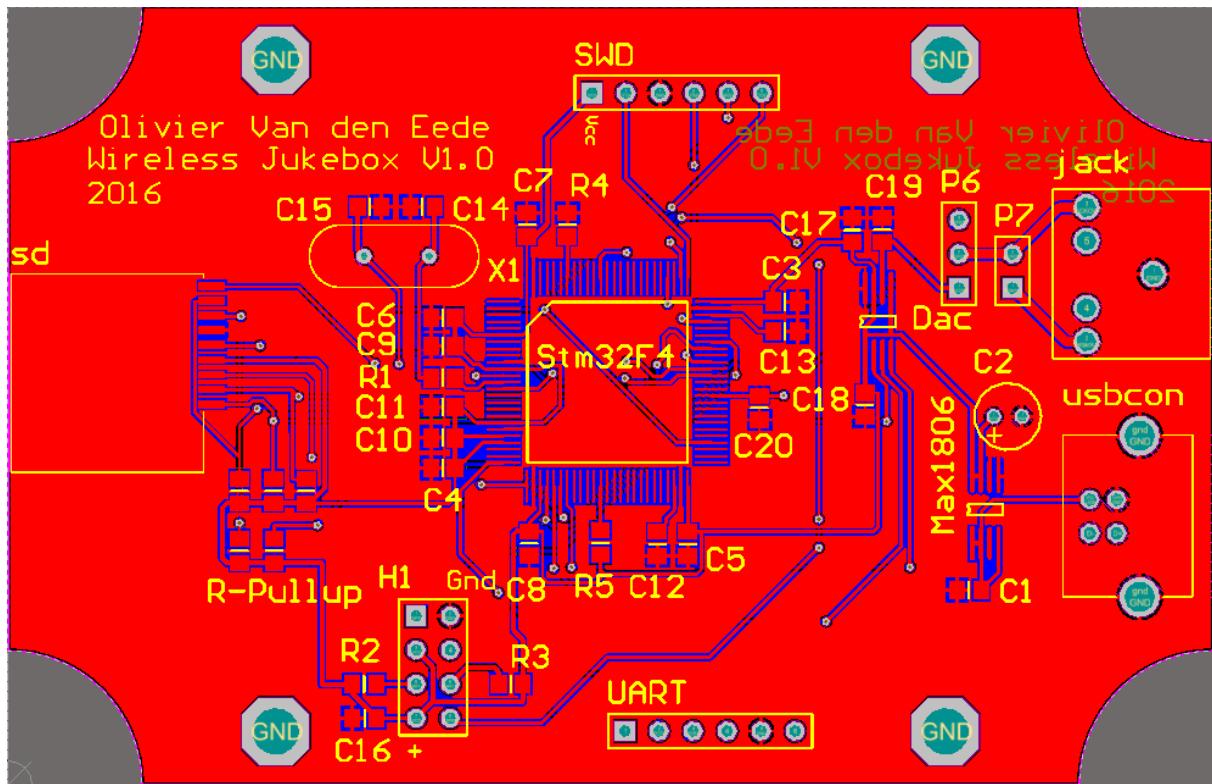
Missing - EDX
Faded - Alan Walker
Perfect - One Direction
Sorry - Justin Bieber

3 Samenhang

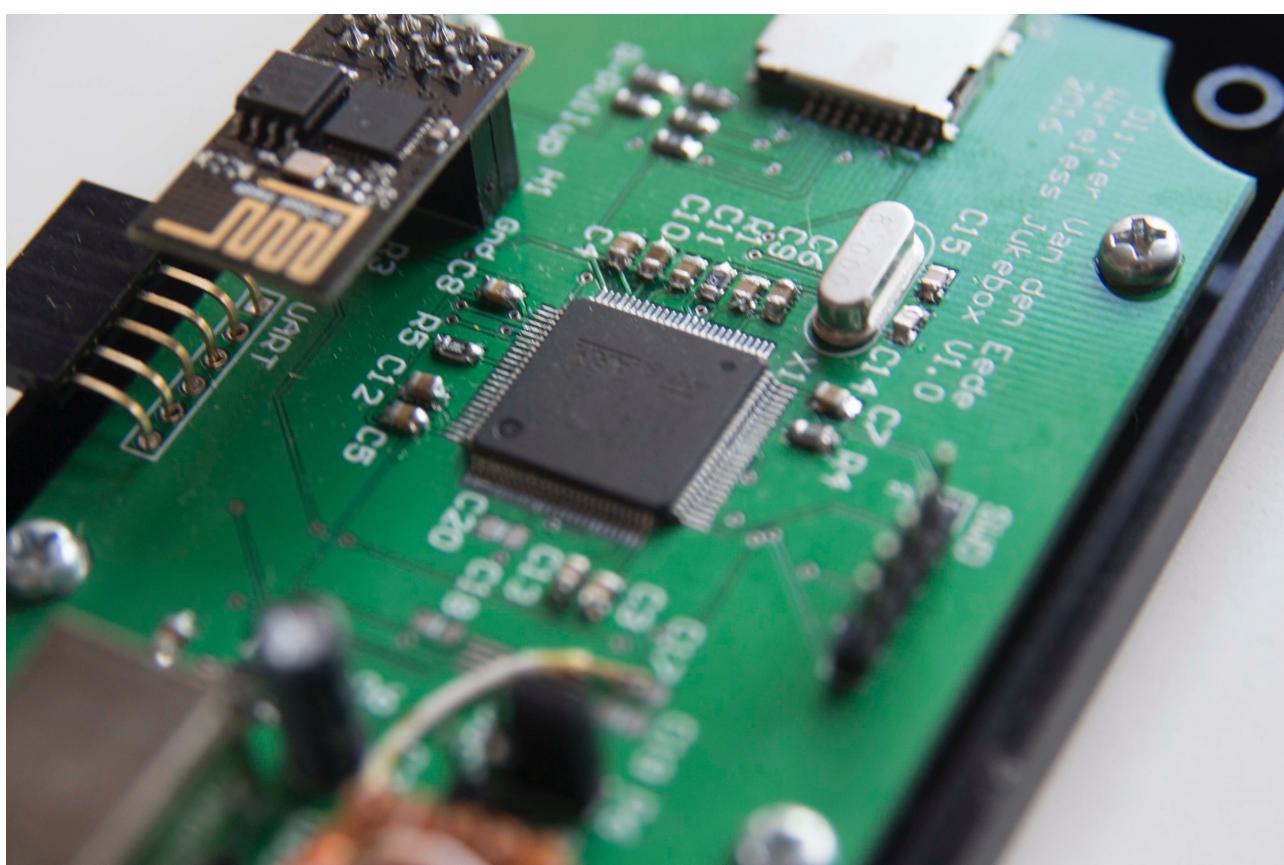
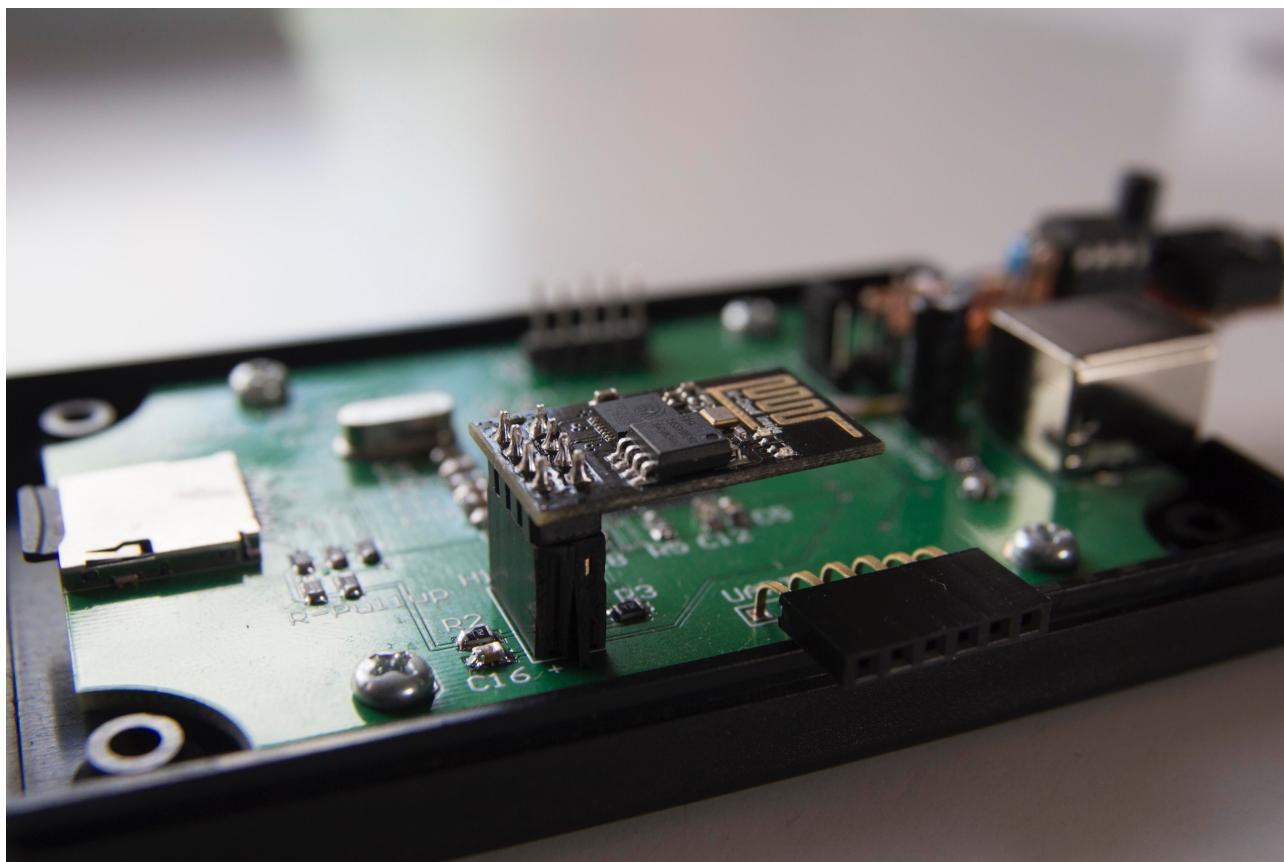
3.1 Volledig schema pcb

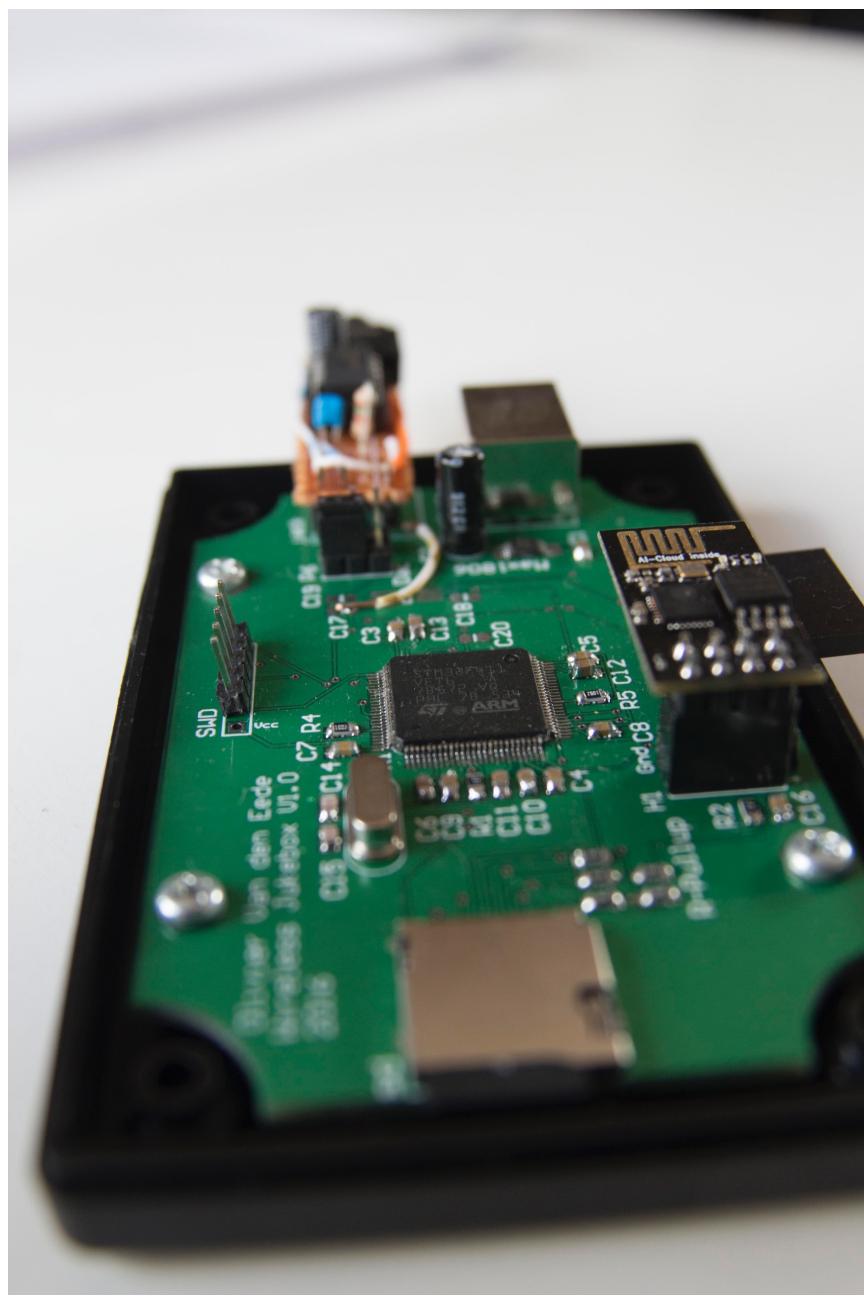


3.2 Layout pcb



3.3 Foto's





4 Broncode

4.1 Esp8266 (init.lua)

4.1.1 Titels opslaan

```
-- Deze functie zet de juiste liedjes in de variabelen
function setSong(nr, titel)
    if(titel ~= nil) then
        if(nr == 1) then
            songTitle1 = titel
        elseif(nr == 2) then
            songTitle2 = titel
        elseif(nr == 3) then
            songTitle3 = titel
        elseif(nr == 4) then
            songTitle4 = titel
        else
            uart.write(0,"ERROR")
        end
    end
end
```

Functie1: Liedjes in juiste var zetten.

De eerste functie is gemaakt om via uart de titels van de liedjes door te sturen. De microcontroller kan nu via uart bv. setSong(1,"Missing - edx") sturen. Lua zal dit dan begrijpen en automatisch deze functie starten. Deze functie zal dan het juiste liedje in de juiste variabele zetten, of een error terugsturen als de index van het liedje niet bestaat.

4.1.2 Placeholders vervangen

```
-- Een functie die in het html bestand zoekt naar de plaats om de titels in te vullen
function checkIfSongTitle( line )
    if (line:find("##SONG1##") ~= nil) then
        line = songTitle1 .. "(" .. song1 .. ")"
    elseif (line:find("##SONG2##") ~= nil) then
        line = songTitle2 .. "(" .. song2 .. ")"
    elseif (line:find("##SONG3##") ~= nil) then
        line = songTitle3 .. "(" .. song3 .. ")"
    elseif (line:find("##SONG4##") ~= nil) then
        line = songTitle4 .. "(" .. song4 .. ")"
    end
    return line;
```

Functie2: Placeholder vervangen.

Deze functie vervangt bij het voorkomen van een placeholder, de lijn door de titel van een liedje.

4.1.3 Opzetten en verwerken http webserver

```
-- We starten de server op op poort 80
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
        end
        local _GET = {}
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
    end

    count(_GET.next)      -- We tellen het aantal stemmen

    file.open('webpaginaOnline.html','r')
    local line = file.readline()
    while (line) do
        line = checkIfSongTitle(line)    -- We kijken of er een titel moet komen
        client:send(line)              -- en vervangen als het nodig is
        line = file.readline()
    end
    file.close()
    client:close();

    collectgarbage();
end)
end)
```

Functie3: Http webserver.

De webserver wordt gestart op poort 80, en er wordt daarna gewacht op een http get request.

Als er een get request toekomt, wordt dit gestoken in de variabele GET. Als er op 1 van de knoppen op de website gedrukt wordt, wordt er een get request gestuurd met een waarde in "next". Deze waarde wordt meegegeven als parameter aan de functie count. Deze functie houd een telling bij van de liedjes waarop gestemd is.

Als er geen request met next is, wordt gewoon de pagina geladen. Dit laden gebeurd lijn per lijn uit een html file die opgeslagen is op de flash van de esp module.

Bij het lijn per lijn lezen van de file, wordt elke lijn door de functie checkIfSongTitle gehaald, deze functie kijkt of er in de html file een placeholder staat waar de songtitels moeten komen.

4.1.4 Opvragen meest gestemde lied

```
-- Returnt het meest gestemde liedjesId
function getBest()
    local lijst = {}
    lijst[1] = song1
    lijst[2] = song2
    lijst[3] = song3
    lijst[4] = song4
    table.sort(lijst)
    if(lijst[4] == song1) then
        uart.write(0,"1\n")
    elseif(lijst[4] == song2) then
        uart.write(0,"2\n")
    elseif(lijst[4] == song3) then
        uart.write(0,"3\n")
    elseif(lijst[4] == song4) then
        uart.write(0,"4\n")
    end
    resetSongs()
end
```

Functie4: Beste liedje opvragen.

Deze functie is gemaakt om via uart het meest gestemde liedje op te vragen. Als de microcontroller getBest() stuurt via uart, dan zal de esp het meest gestemde liedje terugsturen. Dit wordt gedaan door het sorteren van een lijst, en het laatste item in de lijst is dan het meest gestemde liedje. De functie resetSongs zet de tellers van de liedjes op 0.

4.2 Stm32

4.2.1 Sample timer interrupt

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    static uint16_t teller = 0;

    // We willen enkel de compare van timer3
    if(htim->Instance == TIM3)
    {
        if(wavBufferSelect == 0) // Kijken uit welke buffer we moeten lezen
        {
            __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,
                                  (wavBuffer0[teller] + 100));
        }
        else
        {
            __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,
                                  (wavBuffer1[teller] + 100));
        }
        teller++;

        if(teller >= 512)
        {
            bufferEnd = 1;
            wavBufferSelect = ! wavBufferSelect;
            teller = 0;
        }
    }
}
```

Functie1: Muziek op pwm zetten

Dit is de functie die opgeroepen wordt als timer3 een compare match heeft. Deze functie wordt dus aan 44100Hz opgeroepen en past aan deze frequentie de pwm output aan. De waarde die de pwm zal outputen wordt gehaald uit de 2 bufferArrays die afgewisseld worden als ze helemaal uitgelezen zijn.

4.2.2 Begin programma

```
uint8_t aantalNummers = 0;  
  
// De variabele waar de file inzit  
FIL MyFile;  
  
HAL_UART_Transmit(&huart2,"Wireless jukebox V1.0",21,10);  
  
// We halen alle titels op vanaf de sd-kaart en slagen deze op in een array  
char * titels[40];  
aantalNummers = getTitels(&MyFile,titels,20);  
  
// We sturen de eerste liedjes naar de esp  
uint8_t selectedSongs[4] = {0};  
randomNummers(selectedSongs,aantalNummers);  
  
// We gaan nu de eerste wav buffer vullen, hiervoor pakken we steeds nummer 0  
probeer(f_open(&MyFile,"0.wav",FA_READ),"openen file 0.wav");  
getData(&MyFile,wavBuffer0,512);  
getData(&MyFile,wavBuffer1,512);  
  
// Timer3 en pwm output starten  
startPwm();  
  
// We zetten de eerste titels op de esp  
HAL_Delay(5000); // We moeten wachten zodat de esp zeker opgestart is  
sendSongsToEsp(titels,selectedSongs);
```

Functie2: Alles initialiseren

Na het sturen van een welkom message over de uart gaan we op de sd-kaart zoeken naar alle titels van de liedjes die erop staan. Deze worden opgeslagen in de array titels. Hieruit krijgen we ook het aantal liedjes dat op de sd-kaart staat. Vervolgens gaan we een array van 4 elementen vullen met 4 random id's van liedjes. Als eerste liedje openen we altijd lied0 op de kaart en we starten met het vullen van de 2 buffers. Eens de 2 buffers beide opgevuld zijn kunnen we de pwm starten en kan de muziek beginnen spelen. Hierna wordten de titels van de huidige random nummers doorgestuurd naar de esp zodat er op gestemd kan worden.

4.3.3 getTitels (wav.c)

```
uint16_t getTitels(FIL * file, char ** titels, uint16_t maxTitels)
{
    uint16_t teller = 0;
    char buffer[100];

    // We proberen de file met de titels te openen op de sd-kaart
    probeer(f_open(file, "titels.txt", FA_READ), "open titels.txt");

    // Nu lezen we alle lijnen met titles in de file
    while(f_eof(file) == 0)
    {
        f_gets(buffer, sizeof(buffer), file); // We lezen 1 lijn
        // We zoeken wat plaats voor de nieuwe titel
        titels[teller] = (char *) malloc(strlen(buffer));
        // We zetten de titel mee in de array
        strcpy(titels[teller], buffer);

        // Als we het max aantal titels gelezen hebben, moeten we zeker
        // stoppen
        if(teller > maxTitels) break;
        teller++;
    }
    f_close(file);
    return teller;
}
```

Functie3: Alle titels van de sd-kaart halen

In deze functie gaan we alle titels vanaf de sd-kaart ophalen uit het bestand met de titels en slaagt deze op in de titels array via een malloc. Eerst wordt de data ingelezen in een buffer en vervolgens wordt er genoeg plaats in het geheugen gezocht om deze titel op te slaan. Als alle titels opgeslagen zijn wordt er nog een aantal titels teruggestuurd.

4.3.4 startPwm (wav.c)

```
void startPwm(void)
{
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
}
```

Functie4: Pwm starten

Deze functie start de pwm sample timer en activeert de pwm output.

4.3.5 randomNumbers (esp.c)

```
void randomNumbers(uint8_t * selectedSongs, uint8_t aantalSongs)
{
    uint8_t temp = 0;
    uint8_t teller = 0;

    for (teller = 0; teller < 4; teller++)
    {
        selectedSongs[teller] = 5;
    }

    teller = 0;

    // we nemen 4 random nummers en slagen deze op, ze mogen niet dezelfde
    // zijn
    while(teller <= 4)
    {
        temp = rand() % aantalSongs;
        if(temp != selectedSongs[0] && temp != selectedSongs[1] &&
           temp != selectedSongs[2] && temp != selectedSongs[3] &&
           temp <= aantalSongs-1)
        {
            selectedSongs[teller] = temp;
            teller++;
        }
    }
}
```

Functie5: random nummers genereren.

In deze functie gaan we 4 verschillende random nummers genereren hiervoor genereren we elke keer een random nummer en kijken we of dit nog niet gegenereerd was deze keer.

4.3.6 sendSongsToEsp (esp.c)

```
void sendSongsToEsp(char * titels[], uint8_t selectedSongs[])
{
    uint8_t teller = 0;
    // We sturen 4 liedjes naar de esp
    for(teller = 0; teller<4; teller++)
    {
        setSong(1,titels[selectedSongs[0]]);
        HAL_Delay(100);
        setSong(2,titels[selectedSongs[1]]);
        HAL_Delay(100);
        setSong(3,titels[selectedSongs[2]]);
        HAL_Delay(100);
        setSong(4,titels[selectedSongs[3]]);
        HAL_Delay(100);
    }
}
```

Functie6: titels naar esp sturen

Deze functie stuurt de 4 titels van de random gegenereerde liedjes door naar de esp zodat erop gestemd kan worden. Er zit een for loop in om dit een paar keer te doen om er zeker van te zijn dat de liedjes er goed opkomen.

```
void setSong(uint8_t nummer, char titel[])
{
    char buffer[100];
    sprintf(buffer,"setSong(%d,%s')\n",nummer,titel);
    HAL_UART_Transmit(&huart1,(uint8_t *)buffer,strlen(buffer),10);
```

Functie7: titel doorsturen via uart.

Deze functie stuurt 1 titel door via uart in het formaat dat de esp verwacht.

4.3.7 Main while

```
if((songEnd == 1) && (bufferEnd == 1)) // als liedje gedaan is
{
    // We stoppen de pwm
    stopPwm();

    // We sluiten de file
    f_close(&MyFile);

    volgendLiedId = getBest() - 1;

    // Id van het volgend liedje ophalen
    volgendLiedId = selectedSongs[volgendLiedId];

    // Zo staat het liedje op de schijf
    sprintf(openSongString,"%d.wav",volgendLiedId);

    // We openen de nieuwe file
    probeer(f_open(&MyFile,openSongString,FA_READ),"open nieuw liedje");

    // Liedjes voor de volgende stemming kiezen
    randomNummers(selectedSongs,aantalNummers);
    sendSongsToEsp(titels,selectedSongs);

    // Alles terug in de beginpositie
    getData(&MyFile,wavBuffer0,512);
    wavBufferSelect = 0;
    bufferEnd = 0;
    songEnd = 0;
    startPwm();
```

Functie8: als liedje gedaan is.

Als het liedje gedaan is dan stoppen we de pwm, en vragen aan de esp wat het volgende liedje is. Het volgende liedjes id wordt dan omgezet in de titel van het liedje op de sd-kaart en deze wordt dan geladen. Dit liedje wordt dan open gedaan en alles wordt herhaald, we genereren nieuwe random nummers en sturen ze terug naar de esp. Hierna worden alle variabelen en buffers terug klaargezet om opnieuw muziek af te spelen.

```

if(bufferEnd == 1)
{
    bufferEnd = 0;
    // Kijken in welke buffer de interupt aan het lezen is
    if(wavBufferSelect == 0)
    {
        // Interupt leest in 0, wij schrijven in 1
        if(getData(&MyFile,wavBuffer1,512) == 1 ) songEnd = 1;
    }
    else
    {
        if(getData(&MyFile,wavBuffer0,512) == 1 ) songEnd = 1;
    }
}

```

functie9: wavBuffers vullen.

Dit stuk code bevind zich in de main en zal true zijn als de interrupt 512 bytes naar de pwm gestuurd heeft en dus aan het einde van zijn buffer is.

Als het einde bereikt is schakelt de interrupt automatisch over naar de 2de buffer, en kunnen wij dus de lege buffer terug vullen met data van de sd-kaart.

4.3.8 getData (wav.c)

```
uint8_t getData(FIL * fp, char data[],int aantal)
{
    int teller = 0;
    static uint8_t dataGevonden = 0;

    char byte;
    uint8_t bytesRead;
    uint16_t i;

    // eerst moeten we de header over springen tot aan de data chunck
    while(f_eof(fp) == 0)          // we lezen tot aan het einde van de file
    {
        f_read(fp,&byte,1 * sizeof(char),(UINT*)&bytesRead);

        // Kijken of we de data header al gevonden hadden
        if(dataGevonden == 0)
        {
            // we hebben een d gevonden, nu zoeken we naar "ata" van data
            if(GET3CHARS = "ata")
            {
                dataGevonden = 1;
            }
        }
        else           // de data header is gevonden, vanaf nu komt de data
        {
            // We slagen het aantal gevraagde databytes op in de array
            data[teller] = byte;
            teller++;

            if(teller == aantal) //Als we genoeg data gelezen hebben stoppen we
            {
                return 0;
            }
        }
    }
    return 1;
}
```

Functie10: Data uit wav file halen.

Deze functie haalt een opgegeven aantal bytes uit een opgegeven wav file. Eerst wordt er gekeken of we al voorbij de wav-header zijn, als dit het geval is dan kunnen we data uitlezen en opslaan in de meegegeven buffer. Als dit niet het geval is dan moeten we eerst zoeken naar de "DATA" chuck, en dan kunnen we pas de data uitlezen.

5 Conclusie

5.1 Reflectie

Na 13 weken werken aan het project is alles perfect in orde gekomen zoals het doel gesteld was, en is het resultaat een perfect werkend geheel.

Echter is het slechts een prototype en zouden er een paar extra features aan toegevoegd moeten worden om het product echt bruikbaar te maken.

Door een probleem met de fatFS middleware is het niet mogelijk om de titels van de liedjes te halen uit de naam van de bestanden, want dat is niet direct ondersteund door fatFs. Dit is echter wel mogelijk maar dat zou een paar dagen extra zoekwerk en programmeren betekenen.

Ook zou het leuker zijn moest het project overweg kunnen met mp3 bestanden i.p.v. 8bit wav bestanden, dit zou zeker mogelijk zijn bij een revisie van het project.

Verder zou het project ook uitgevoerd kunnen worden in stereo wat nu mono is.

Als al deze features toegevoegd zouden worden zou het project echt zijn nut kunnen hebben in bv. een restaurant waar de klanten dan mee een zegje hebben in de muziek die er afgespeeld wordt.

5.2 Kostprijs

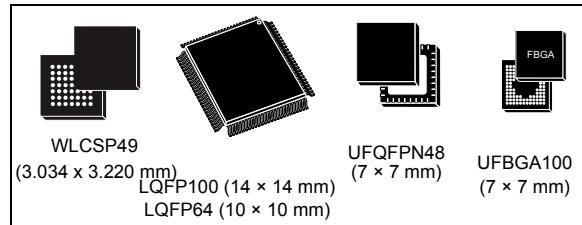
Onderdeel	Aantal	Prijs	Totaal
esp8266	1	2,40 €	2,40 €
sd-kaart (4gb)	1	6 €	6 €
cap + res + connector	1	10 €	10 €
sd-socket	1	1 €	1 €
3v regulator	1	Free sample	0 €
Stm32F4	1	8 €	8 €
Pcb	5	35 €	35 €
Totaal			62,4 €

ARM® Cortex®-M4 32b MCU+FPU, 125 DMIPS, 512KB Flash,
128KB RAM, USB OTG FS, 11 TIMs, 1 ADC, 13 comm. interfaces

Datasheet - production data

Features

- Dynamic Efficiency Line with BAM (Batch Acquisition Mode)
- Core: ARM® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 100 MHz, memory protection unit, 125 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
 - up to 512 Kbytes of Flash memory
 - 128 Kbytes of SRAM
- Clock, reset and supply management
 - 1.7 V to 3.6 V application supply and I/Os
 - POR, PDR, PVD and BOR
 - 4-to-26 MHz crystal oscillator
 - Internal 16 MHz factory-trimmed RC
 - 32 kHz oscillator for RTC with calibration
 - Internal 32 kHz RC with calibration
- Power consumption
 - Run: 100 µA/MHz (peripheral off)
 - Stop (Flash in Stop mode, fast wakeup time): 42 µA Typ @ 25°C; 65 µA max @25 °C
 - Stop (Flash in Deep power down mode, fast wakeup time): down to 10 µA @ 25 °C; 30 µA max @25 °C
 - Standby: 2.4 µA @25 °C / 1.7 V without RTC; 12 µA @85 °C @1.7 V
 - V_{BAT} supply for RTC: 1 µA @25 °C
- 1×12-bit, 2.4 MSPS A/D converter: up to 16 channels
- General-purpose DMA: 16-stream DMA controllers with FIFOs and burst support
- Up to 11 timers: up to six 16-bit, two 32-bit timers up to 100 MHz, each with up to four IC/OC/PWM or pulse counter and quadrature (incremental) encoder input, two watchdog



- timers (independent and window) and a SysTick timer
- Debug mode
 - Serial wire debug (SWD) & JTAG interfaces
 - Cortex®-M4 Embedded Trace Macrocell™
- Up to 81 I/O ports with interrupt capability
 - Up to 78 fast I/Os up to 100 MHz
 - Up to 77 5 V-tolerant I/Os
- Up to 13 communication interfaces
 - Up to 3 x I²C interfaces (SMBus/PMBus)
 - Up to 3 USARTs (2 x 12.5 Mbit/s, 1 x 6.25 Mbit/s), ISO 7816 interface, LIN, IrDA, modem control)
 - Up to 5 SPI/I²Ss (up to 50 Mbit/s, SPI or I²S audio protocol), SPI2 and SPI3 with muxed full-duplex I²S to achieve audio class accuracy via internal audio PLL or external clock
 - SDIO interface (SD/MMC/eMMC)
 - Advanced connectivity: USB 2.0 full-speed device/host/OTG controller with on-chip PHY
- CRC calculation unit
- 96-bit unique ID
- RTC: subsecond accuracy, hardware calendar
- All packages (WLCSP49, LQFP64/100, UFQFPN48, UFBGA100) are ECOPACK®2

Table 1. Device summary

Reference	Part number
STM32F411xC	STM32F411CC, STM32F411RC, STM32F411VC
STM32F411xE	STM32F411CE, STM32F411RE, STM32F411VE



Espressif Systems' Smart Connectivity Platform (ESCP) demonstrates sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

1.2. Features

- 802.11 b/g/n
- Integrated low power 32-bit MCU
- Integrated 10-bit ADC
- Integrated TCP/IP protocol stack
- Integrated TR switch, balun, LNA, power amplifier and matching network
- Integrated PLL, regulators, and power management units
- Supports antenna diversity
- WiFi 2.4 GHz, support WPA/WPA2
- Support STA/AP/STA+AP operation modes
- Support Smart Link Function for both Android and iOS devices
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IR Remote Control, PWM, GPIO
- STBC, 1x1 MIMO, 2x1 MIMO
- A-MPDU & A-MSDU aggregation & 0.4s guard interval
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)
- +20 dBm output power in 802.11b mode
- Operating temperature range -40C ~ 125C
- FCC, CE, TELEC, WiFi Alliance, and SRRC certified

1.3. Parameters

Table 1 Parameters

TL07xx Low-Noise JFET-Input Operational Amplifiers

1 Features

- Low Power Consumption
- Wide Common-Mode and Differential Voltage Ranges
- Low Input Bias and Offset Currents
- Output Short-Circuit Protection
- Low Total Harmonic Distortion: 0.003% Typical
- Low Noise
 $V_n = 18 \text{ nV}/\sqrt{\text{Hz}}$ Typ at $f = 1 \text{ kHz}$
- High-Input Impedance: JFET Input Stage
- Internal Frequency Compensation
- Latch-Up-Free Operation
- High Slew Rate: 13 V/ μs Typical
- Common-Mode Input Voltage Range
Includes V_{CC+}

2 Applications

- Motor Integrated Systems: UPS
- Drives and Control Solutions: AC Inverter and VF Drives
- Renewables: Solar Inverters
- Pro Audio Mixers
- DLP Front Projection System
- Oscilloscopes

3 Description

The TL07xx JFET-input operational amplifier family is designed to offer a wider selection than any previously developed operational amplifier family. Each of these JFET-input operational amplifiers incorporates well-matched, high-voltage JFET and bipolar transistors in a monolithic integrated circuit.

The devices feature high slew rates, low-input bias and offset currents, and low offset-voltage temperature coefficient. The low harmonic distortion and low noise make the TL07xseries ideally suited for high-fidelity and audio pre-amplifier applications. Offset adjustment and external compensation options are available within the TL07x family.

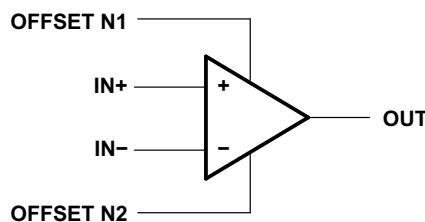
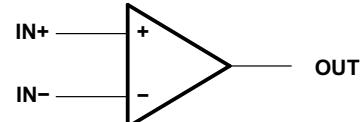
Device Information⁽¹⁾

PART NUMBER	PACKAGE	BODY SIZE (NOM)
TL07xxD	SOIC (14)	8.65 mm × 3.91 mm
	SOIC (8)	4.90 mm x 3.90 mm
TL07xxFK	LCCC (20)	8.89 mm × 8.89 mm
TL07xxJG	PDIP (8)	9.59 mm x 6.67 mm
TL074xJ	CDIP (14)	19.56 mm × 6.92 mm
TL07xxP	PDIP (8)	9.59 mm x 6.35 mm
TL07xxPS	SO (8)	6.20 mm x 5.30 mm
TL074xN	PDIP (14)	19.3 mm × 6.35 mm
TL074xNS	SO (14)	10.30 mm × 5.30 mm
TL07xxPW	TSSOP (8)	4.40 mm x 3.00 mm
TL074xPW	TSSOP (14)	5.00 mm × 4.40 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Logic Symbols

TL071

TL072 (each amplifier)
TL074 (each amplifier)

An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

	Datum goedkeuring	Versie	Datum herziening		Redacteur(s)

Projectvoorstel

1 Teamleden :

Olivier Van den Eede

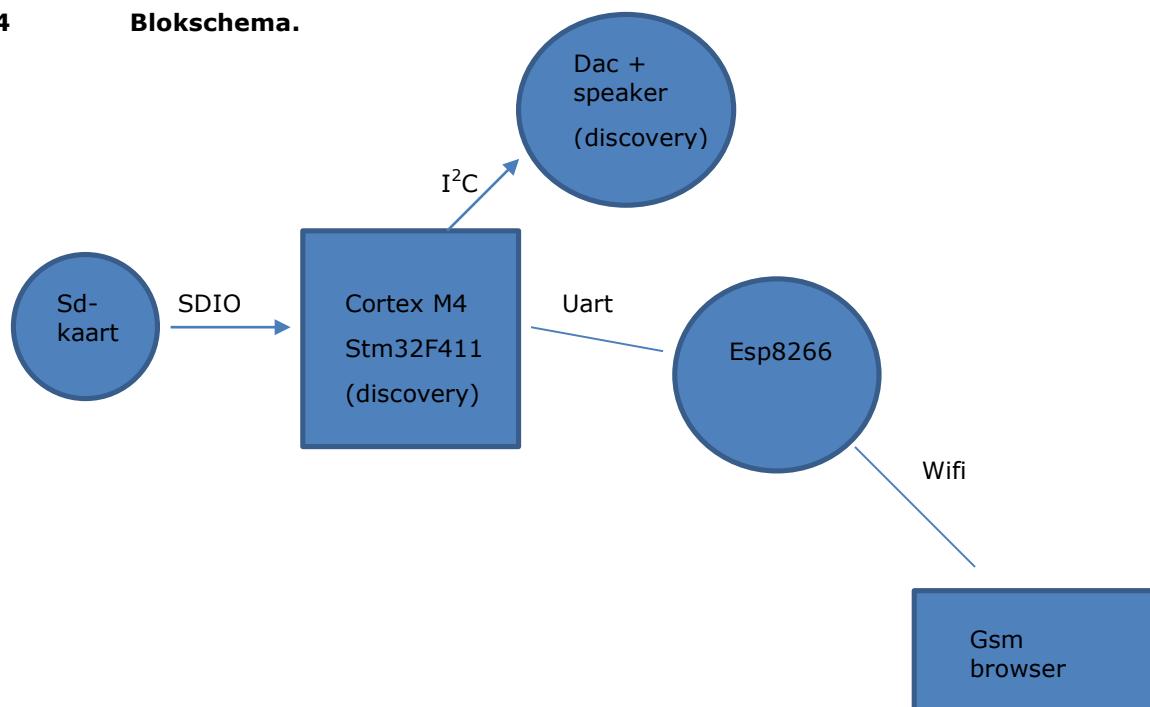
2 Titel project.

Wireless Jukebox

3 Korte omschrijving van het project.

Ontwikkelen van een embedded systeem op basis van een cortex M4(stm32F411) dat muziek van een sd-kaart leest en deze afspeeld. De module wordt verbonden met een esp8266 wifi module waarop een webserver zal draaien. Gebruikers kunnen nu inloggen via wifi op de esp8266 en stemmen voor het volgende liedje. Als uitbreiding is het eventueel nog mogelijk om een wachtrij van liedjes te maken.

4 Blokschema.



5**Een budgetraming (mini).**

Naam	Prijs
Stm32F4 Discovery	16,00 €
Esp8266	2,40 €
Sd-kaart	10,00 €
Basis totaal	28,40 €
Eventueel pcb	
pcb	20,00€
weerstanden en condensatoren	8,00€
Cortex M4	7,00€
Sd connector	3,00€
Totaal pcb	38,00 €
totaal	66,40 €

Plan van aanpak

Titel van het project: Wireless jukebox
Projectmedewerkers: Olivier Van den Eede

Doel & ontwerpspecificaties

Doel

Met een microcontroller, een sd-kaart en een wifi module een soort jukebox maken die muziek af kan spelen. Men kan stemmen op het volgende liedje via een gsm.

Ontwerpspecificaties

De gebruiker kan via wifi inloggen op de module, in de browser naar de webserver surfen en daar stemmen op het volgende liedje.

De muziek wordt vanaf een sd-kaart ingelezen in de Uc en afgespeeld via een dac.

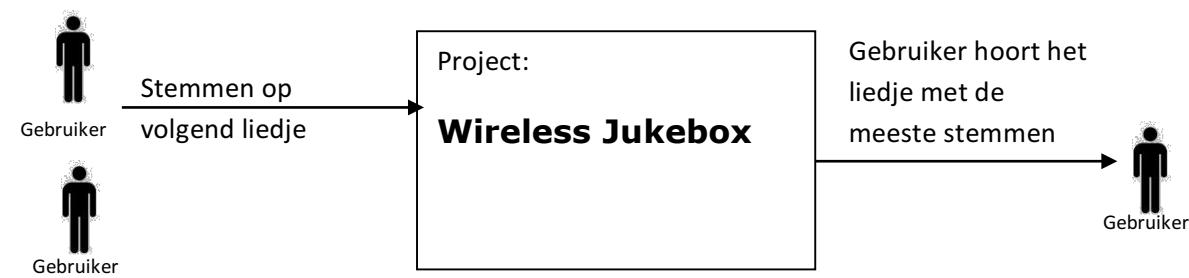
Grootte: een microcontroller, een sd-kaart, een esp8266 en een dac + audio connector.

Prijs: +-70 euro

Tijd: n/a

Levensduur: enkele jaren

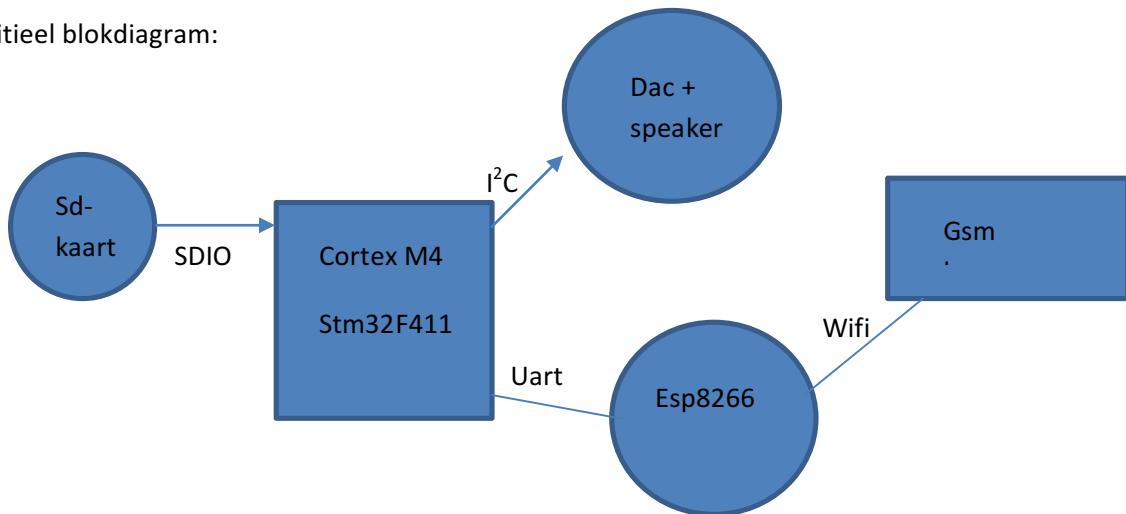
Functionele analyse



Wireless jukebox:

- Module krijgt voedingsspanning en speelt vanzelf een eerste liedje
- Microcontroller stuurt een 3 tal titels van liedjes en id's naar de esp
- Esp8266 zet deze titels om in een website met knoppen waar gebruikers op kunnen klikken
- Esp houdt een telling bij van het meest gestemde liedje
- Bij het einde van het liedje vraagt de Uc aan de esp het liedjes-id dat gewonnen heeft
- De microcontroller zoekt het volgende liedje en alles begint opnieuw

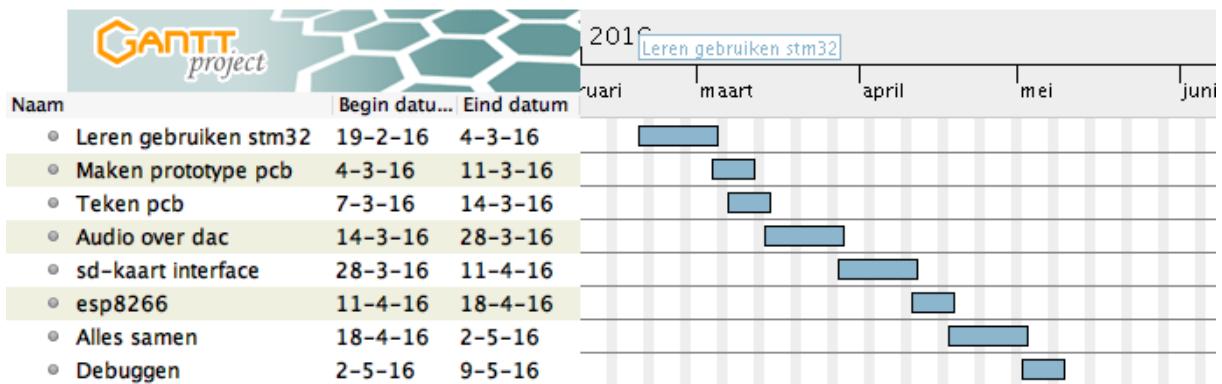
Initieel blokdiagram:



Work Breakdown Structure (WBS)

1. Leren gebruiken stm32F4 c en cubeMx	2 Weken
2. Maken print met dac en sd interface	2 Weken
2.1 Prototype op gaatjesprint	1 week
2.2 Pcb ontwerpen	1 weken
3. Programmeren	7 Weken
3.1 Audio sturen over dac	2 weken
3.2 Sd-kaart inlezen	2 weken
3.3 Esp8266	1 week
3.4 Alles samen laten werken	2 weken
4. Debuggen en Testen	1 Week

Gantt-chart



Logboek project2: Wireless-Jukebox		
dag	tijd(u)	taak
20/Feb	1	installeren toolchain en ide
21/Feb	2	stm32f4 testen en programmeren
21/Feb	1	Maken projectvoorstel
23/Feb	1	Maken plan van aanpak
23/Feb	1	testen uart stm32
24/Feb	2	Begin tekenen pcb
25/Feb	1,5	testen spi stm32
25/Feb	2	verder tekenen schema
27/Feb	2	verder tekenen schema
27/Feb	0,5	testen i2c stm32
29/Feb	1	verder tekenen schema
29/Feb	2	uitzoeken hoe .wav fileheader in een zit en data mogelijkheden
1/Mrt	1	Uitzoeken interrupt stm32f4
1/Mrt	2	Rx- interrupt bij ontvangst uart data
1/Mrt	1,5	tekenen schema+begin pcb
2/Mrt	1,5	Verder tekenen pcb
5/Mrt	3	Verder tekenen pcb
5/Mrt	1,5	Timers en pwm op stm32
6/Mrt	3	pcb+stm32 pinout valideren
8/Mrt	4	Uitzoeken werking sdio en fatfs
9/Mrt	2	Sdio+fatfs
11/Mrt	4	testen esp: lua, opensdk en sming
15/Mrt	3	Installeren open-sdk(esp8266) op linux
15/Mrt	1,5	lua op esp8266
17/Mrt	3	webpagina op esp(in html file)
20/Mrt	2	Webpagina + stemmen afwerken op esp
20/Mrt	2	testen uart op esp
21/Mrt	4	maken kleine app + json response from esp
22/Mrt	6	communicatie tussen esp en stm
24/Mrt	2	stm programma voor uart
25/Mrt	3	communicatie stm en esp afgewerkt
10/Apr	2	Begin maken eindverslag
12/Apr	2	Schrijven functie voor uitlezen wav file
14/Apr	2	begin muziek spelen met pwm
17/Apr	2	verder werken eindverslag
19/Apr	3	muziek afspelen via pwm
20/Apr	2	Verderwerken pwm muziek
21/Apr	2	opamp schakeling voor muziek
26/Apr	2	verder maken eindverslag
28/Apr	2	solderen stm32 op pcb
28/Apr	2	verder solderen pcb + basis test
29/Apr	2	sd-files uitlezen op pcb

30/Apr	4,5	muziek afspelen vanaf sd-kaart
3/Mei	3	geluidsqualiteit verbeteren
3/Mei	3	stemmen op liedjes
5/Mei	3	afwerken project
10/Mei	3	uitmeten audio filter
11/Mei	1	Solderen filter
10/Mei	2	Werken verslag
Totaal:	109,5	u

Bijlage 5:

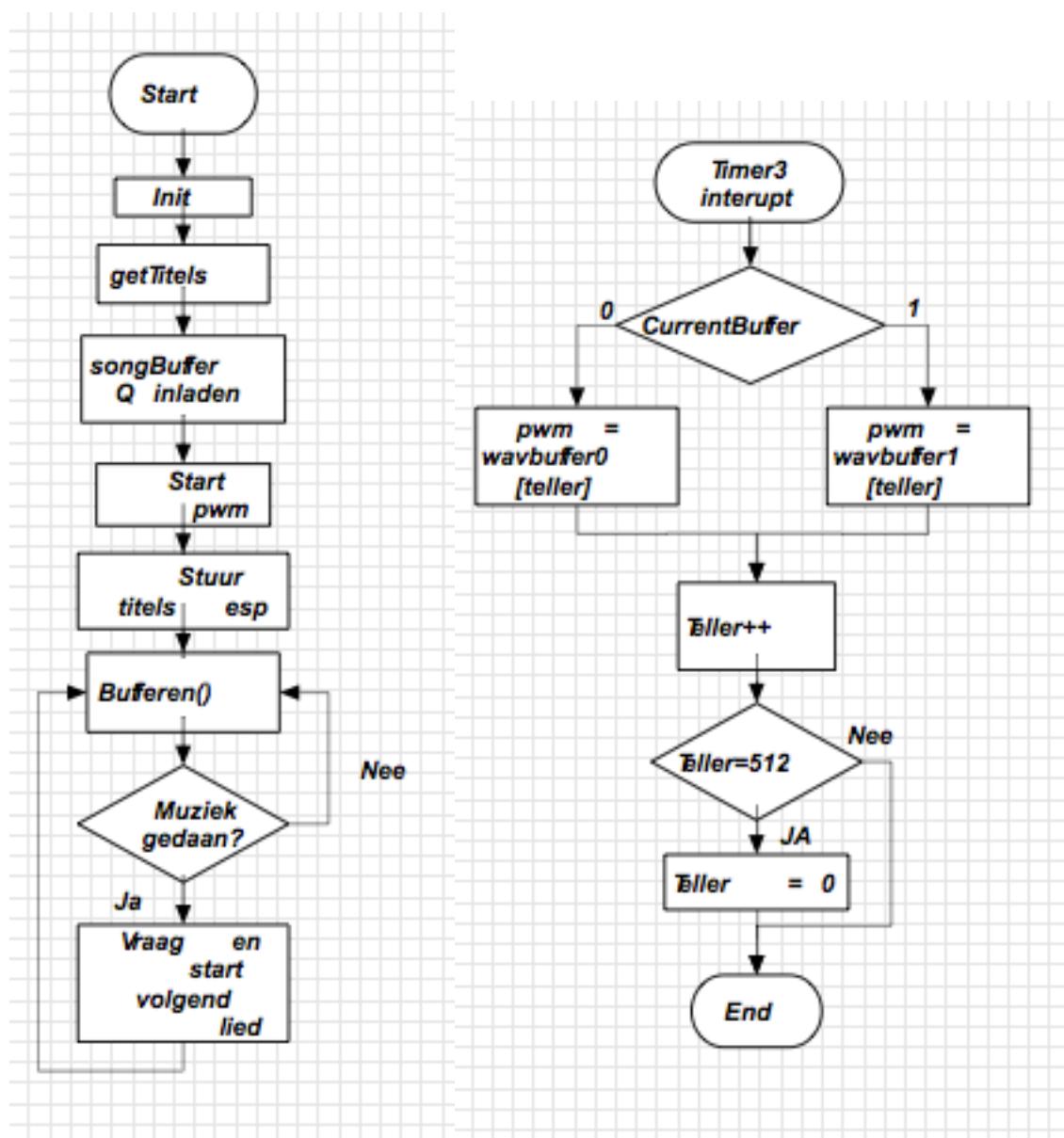


Fig1: Flowchart main

Fig2: Flowchart interrupt

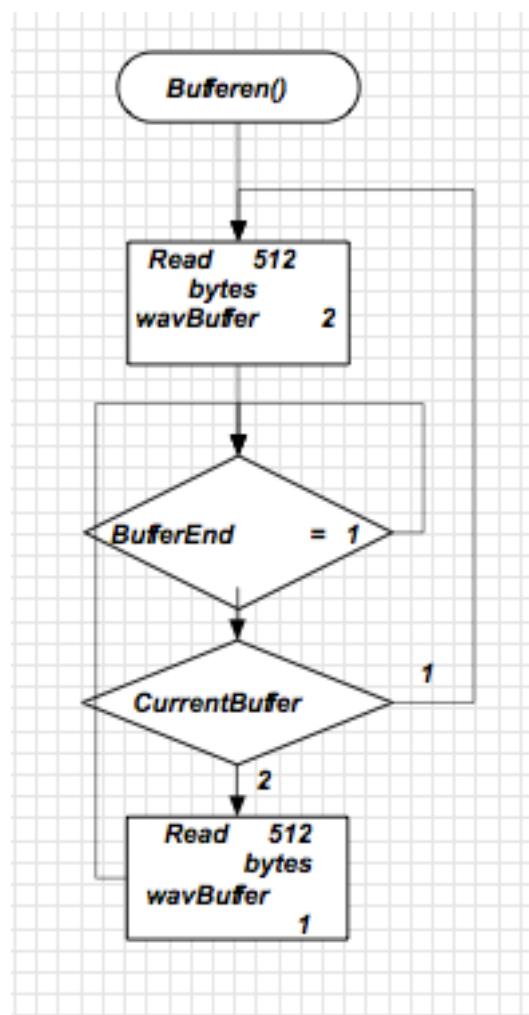


Fig3: Flowchart bufferen muziek