PŘEHRÁVAČ MELODIÍ

Karolína Sehnalová
ČVUT FEL, Katedra Měření
vytvořeno pro účely KPE 2021

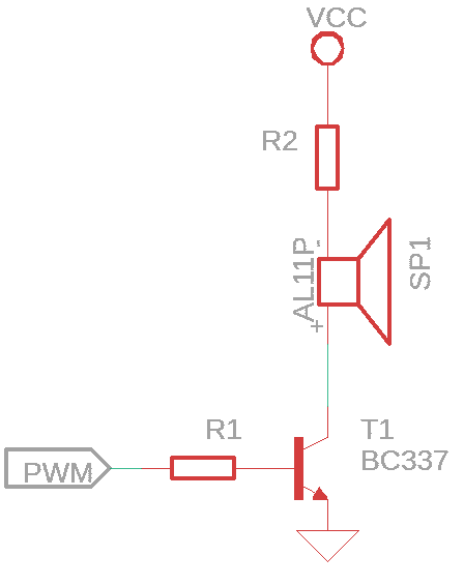
9. srpna 2021

1 Úvod

Přehrávač Melodií je jednoduchý program, který přehrává jednu melodii stále dokola a mění její tempo z pomalé na rychlou a obráceně v nekonečné smyčce. Tuto práci jsem vytvořila v rámci předmětu LPE 2021 jako semestrální práci. Původní práce byla oproti této rozšířena o spoustu funkcionalit, především možnost ovládání přes wi-fi a možnost využívat reproduktor jako piáno. Pro účely kurzu KPE 2021 byla zjednodušena na její samotné jádro, kterým je přehrávání melodií.

2 Zapojení

Využijeme procesor STM32F042, buzzer, tranzistor BC337, rezistor R1 do báze tranzistoru s odporem řádově $k\Omega$ (doporučuji 2700Ω), rezistor R2 do kolektoru do 47 Ω a nebo tam ani nemusí být. Připojíme k procesoru na pin PB1 a na napětí 3.3V.



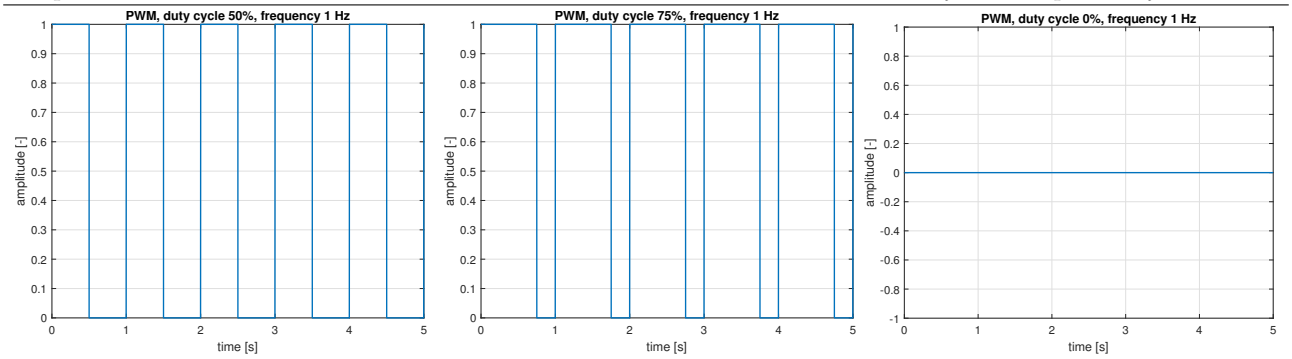
Obrázek 1: Schéma zapojení bzučáku pro aplikaci

Tranzistor zde funguje jako spínač. Důležité také je, aby reproduktorem netekl stejnosměrný proud. Lze zapojení i modifikovat a stejnosměrně ho oddělit s použitím kondenzátoru.

3 Melodie

Melodie jsou v mém programu tři. První je má melodie, druhá je Twinkle Twinkle Little Star a třetí je znělka ze SuperMario. Jsou přehrávány pomocí PWM o měnící se frekvenci podle tónu, který chci hrát. Frekvence jsou vysoké, protože buzzer má omezený frekvenční rozsah o frekvencích okolo $2kHz$. Melodie mám uložené ve struktuře zvané pole, což je vlastně vektor pevné délky prvků jednoho stejného typu. Ke každé melodii mám takováto pole dvě. Jedno reprezentuje frekvence tónů, jak jdou po sobě a druhé obsahuje doby trvání hraní konkrétního tónu. Pak ještě tato pole obsahují speciální znaky, aby se dal dobře poznat konec melodie a podobně.

PWM (pulzně šířková modulace) je signál, který nabývá dvou hodnot. Má frekvenci a střídu. Střída je poměr jedné hodnoty signálu ku druhé. Příklady třech PWM signálů s různou střídou jsou pro ilustraci vidět na Obrázku 2. Frekvence PWM zůstává ve všech případech stejná (na Obrázku 2c už určování frekvence nemá smysl, ale graf vznikl jako PWM o frekvenci 1 Hz a střídou 0%).



(a) PWM se střídou 50% (b) PWM se střídou 75% (c) PWM se střídou 0%

Obrázek 2: Ukázka PWM signálů s různými střídami o stejné frekvenci

Pauza mezi tóny je vyřešena snížením střídy PWM na 0. Jak je vidět na Obrázku 2c, tak v tuto chvíli nedochází vlastně k žádným změnám, které by způsobovaly zvuk. Co je také důležité, je to, že v tuto chvíli neprochází reproduktorem proud. Stejnosměrný proud by reproduktorem procházet neměl nikdy.

4 Princip funkce programu

Jak je již uvedeno výše, program přehrává jednu melodii ve smyčce a mění rychlost přehrávání. O to se stará funkce ***main()***, což je vždy první funkce, která se volá hned po spuštění programu a obsahuje samotnou strukturu programu. Ta zpravidla volá další "podfunkce,"které dělají jednotlivé dílčí části programu. Také dochází k alokaci proměnnných (vytvoření prostoru v paměti pro proměnné) a ještě před spuštěním programu se všechny *#DEFINE* nahradí čísly. Všechny proměnné musí mít přesně daný typ (*int*, *float*, *char*,...), právě kvůli alokaci, protože každý typ zabere jinak velkou paměť.

4.1 Funkce main

Funkce *main()* v mé práci je složená z nekonečného opakování smyčky, která přehraje melodii buď pomalu, nebo rychle. Rozhodne se podle toho, zda proměnná *fast* je nastavená zrovna na 1 nebo 0. Pokud je *fast* 1, tak naposledy byla melodie hraná rychle, a proto se spustí přehrávání pomalu a *fast* se změní na 0. Kód hlavní smyčky je vidět níže. Použité proměnné jsou předem definovány funkcí main ať už jsou lokální (pouze pro funkci main) a nebo globální (společné pro celý soubor).

```
while (1) { // simple while loop playing one defined melody in two tempos
    if (fast == 1 && cur_tone == NULL) {
        wait_ms(3000);
        tempo--;
        cur_times = &mario_times[0];
        cur_tone = &mario[0];
        fast = 0;
        ticker.detach();
        ticker.attach(&play_fragment, tempo_arr[tempo]);
    }
    else if (cur_tone == NULL) {
        wait_ms(3000);
        fast = 1;
        tempo++;
        cur_times = &mario_times[0];
        cur_tone = &mario[0];
        ticker.detach();
        ticker.attach(&play_fragment, tempo_arr[tempo]);
    }
}
```

4.2 Interrupt Service Routine

Interrupt service routine (ISR), česky přerušení, je proces, který se spustí přednostně a přeruší ostatní procesy, které procesor provádí. Po jeho dokončení pokračuje procesor v činnostech, které byl nucen přerušit.

V mém programu je pomocí ISR řešeno přehrávání melodií. Každý pevně definovaný časový interval se spustí ISR, která zahraje tón. Přerušením to řeším proto, aby bylo zachované tempo (a rytmy) melodie. Procesor v tuto chvíli sice mnoho jiných činností nedělá, ale obecně by mohl. Například v mém původním projektu například obsluhoval komunikaci přes wi-fi nebo hrál tóny na žádost uživatele (podobně jako elektronické piáno).

ISR jsem implementovala pomocí objektu Ticker, který umožňuje volat mnou napsanou funkci opakovaně v pevných intervalech. To se provede metodou *attach()*. Zápis kódu je pak:

```
Ticker ticker;
ticker.attach(&isr_name, time_interval);
```

kde na prvním řádku je definice objektu *Ticker*, který pojmenuji *ticker*, *isr_name*[1] je jméno funkce, kterou chci opakovaně volat (přerušení) a *time_interval* je časový interval, po kterém chci funkci opakovat.

V mém kódu si se funkce zpracování přerušení nazývá ***play_fragment()***. Ta se tedy spustí každých řekněme 10 ms. Počet takto uběhlých intervalů se počítá globální proměnnou (takovou proměnou, která je vidět z funkce i mimo ni) *ms_counter*. Ještě využívá globální proměnné *cur_tone* a *cur_times*. To jsou pointery. Ty ukazují na jednu konkrétní adresu v paměti. Slouží k orientaci v poli melodií.Ukazují přesně na tón a k němu náležící dobu přehrávání. Jsou to tedy ukazatele do pole tónů a pole příslušných časů. Po dosažení (resp. přesažení) příslušné doby přehrávání tónu se oba pointery posunou v poli o jeden prvek dál - tedy ukazují na další tón a jeho délku. V tuto chvíli se také vynuluje *ms_counter*, aby u dalšího tónu měřil správný čas. Kód celé ISR je vidět níže.

```
void play_fragment() { // ISR (interrupt service routine)
// this function is called every fixed time interval (in this case 10 ms is the default)
    if (cur_tone == NULL) {  // no melody
        my_pwm.write(0.0f);
        ms_counter = 0;
    }
    else if (cur_tone[0] == 1) {  // end of melody
        my_pwm.write(0.0f);
        ms_counter += 10;
        cur_tone = NULL;
        ms_counter = 0;
    }
    else {  // playing the tone (PWM, duty cycle 50%)
        ms_counter += 10;
        if (ms_counter >= cur_times[0]) {
            cur_times++;
            if (cur_tone[0] == 0) {  // pause (implemented by lowering duty cycle to 0%)
                my_pwm.write(0.0f);
            }
            else {  // regular tone read from the melody array
                float T = 1/cur_tone[0];
                my_pwm.write(0.50f);
                my_pwm.period(T);
            }
            cur_tone++;
            ms_counter = 0;
        }
    }
}
```

4.3 Přehrávání PWM

Pro vytvoření PWM slouží objekt PwmOut. Podobně jako Ticker obsahuje metody, kterými usnadňuje ovládání pwm. Mezi nimi důležité jsou:

- write(duty_cycle) → nastaví střídu o hodnotě *duty_cycle*
- period(per) ⇒ nastaví periodu PWM na hodnotu *per*

Přistupuje se k nim stejně jako u Timeru tzv. tečkovou notací, to znamená napsání jména mé instance objektu, za ním tečku a poté požadovanou metodu(např. pokud si pojmenuji tento objekt my_pwm, budu psát my_pwm.write(0.5f)).

5 Modifikace a úpravy kódu

Kód je jednoduchý a svou strukturou vybízí k mnoha úpravám. Jsou v něm dostupné tři melodie, ale v mém kódu se využívá jen jedna, proto můžete melodie měnit a střídat nebo i zkusit napsat nové vlastní. Můžete měnit počet jejich přehrávání, pauzy mezi nimi, klidně v době mezi přehráváními využít bzučák k dalším činnostem. Aplikace slouží především k zábavným účelům, seznámení se s kódem psaným v c++ a buzzerem. Nápady na úpravy jsou již napsány v kódu, který jste obdrželi, s označním "TODO".

[1]Poznámka: jedná se tedy vlastně o adresu funkce v paměti, kterou lze teoreticky předat i jinak než symbolem & se jménem funkce