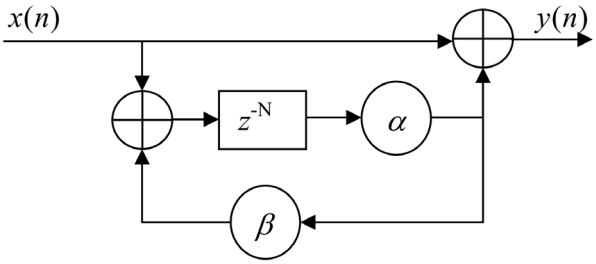
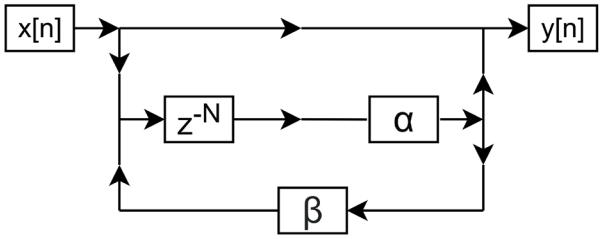
FEKT	Předmět	**Signálové procesory**	
	Jméno	**Marek Coufal**	
	Ročník 3.	Studijní skupina	**MPC-EKT**
	Spolupracoval	Měřeno dne	
Kontroloval	Hodnocení	Dne	**2.12.2025**
Číslo úlohy	Název úlohy		
	SPR Projekt – zvukový efekt Echo / Reverb		

Teoretická část



Obrázek 2: vývojový diagram ze zadání projektu.



Obrázek 1: Překreslené zadání s vyznačeným směrem signálových toků signal flow diagramu.

Přenosová funkce systému byla stanovena pomocí Masonova pravidla na základě signal–flow grafu odvozeného z blokového diagramu zapojení. Zpožďovací článek byl vyjádřen ve tvaru z^{-N} a zesílení konstantami α a β.
Z grafu byla identifikována přímá cesta ze vstupu $X(z)$ k výstupu $Y(z)$ vedoucí přes článek z^{-N} a zesílení α. Zisk této přímé cesty je tedy

$$P_1 = \alpha z^{-N}$$

Druhá přímá cesta představuje přímé propojení vstupu na výstup bez dalších bloků, proto její přenos je

$$P_2 = 1$$

V grafu byla dále nalezena jedna jednoduchá smyčka tvořená větví s přenosem β, zpožďovacím článkem z^{-N}a zesílením α. Zisk této smyčky je

$$L_1 = \alpha\beta z^{-N}$$

Jiné smyčky ani kombinace nesouhlasných smyček se v zapojení nevyskytují.
Determinant grafu je tedy

$$\Delta = 1 - L_1 = 1 - \alpha\beta z^{-N}$$

Determinant $\Delta_k$pro jednotlivé přímé cesty se stanoví vyloučením těch smyček, které se dotýkají dané cesty. Smyčka L_1se dotýká přímé cesty P_1, proto je

$$\Delta_1 = 1$$

Naopak nedotýká se přímé cesty P_2, tudíž

$$\Delta_2 = 1 - \alpha\beta z^{-N}$$

Dosazením do Masonova vztahu

$$H(z) = \frac{P_1\Delta_1 + P_2\Delta_2}{\Delta}$$

dostáváme

$$H(z) = \frac{\alpha z^{-N} \cdot 1 + 1 \cdot (1 - \alpha\beta z^{-N})}{1 - \alpha\beta z^{-N}}$$

Po úpravě čitatele:

$$H(z) = \frac{1 + \alpha z^{-N} - \alpha\beta z^{-N}}{1 - \alpha\beta z^{-N}}$$

Výslednou přenosovou funkci lze zapsat i ve zjednodušeném tvaru:

$$H(z) = 1 + \frac{\alpha z^{-N}}{1 - \alpha\beta z^{-N}}$$

Takto získaná přenosová funkce popisuje chování systému v z-doméně v závislosti na zpoždění N a koeficientech α a β.

Normalizace a stabilita

Jelikož v závislosti na nastavených parametrech má přenosová funkce zesílení, je potřeba hodnoty normalizovat, aby nedošlo k překročení full scale rozsahu kodeku.
Zesílení přenosové funkce, kterým je potřeba podělit každý vzorek, abychom korektně normalizovali signál můžeme vyjádřit jako

$$k_{norm} = max|H(z)| = max\left|1 + \frac{\alpha z^{-N}}{1 - \alpha\beta z^{-N}}\right|$$

Výraz z^{-N} můžeme nahradit $e^{-j2\pi\frac{f}{fs}}$ a pro potřeby vyšetření maxima jej ještě můžeme nahradit číslem 1, jelikož větší absolutní hodnoty nenabyde. Pro výpočet normalizačního koeficientu použijeme tedy rovnici

$$k_{norm} = 1 + \frac{\alpha}{1 - \alpha\beta}$$

Ještě krátce se můžeme zaměřit na stabilitu systému, pomocí vyšetření jmenovatele

$$0 = 1 - \alpha\beta z^{-N}$$

Aby tedy platilo, že všechny póly jmenovatele leží uvnitř jednotkové kružnice musí platit

$$|\alpha\beta| \leq 1$$

Všechny tyto podmínky (podmínky stability a normalizace) jsou následně použity při implementaci funkcí na DSP, které kontrolují, že zadané hodnoty uživatelem jsou korektní a vypočtou normalizační koeficient.

Výchozí simulace v Pythonu

Jako první krok co se vývoje SW týče jsem nejdříve efekt implementoval v Pythonu, který načte vybraný audio soubor a poté zpracovává vzorek po vzorku ve smyčce for, což je shodné s budoucí implementací na DSP, kde je také zpracováván vzorek po vzorku.
Pomocí tohoto skriptu jsem si odladil postup jednotlivých matematických operací pro docílení echo efektu, aniž by bylo nutné řešit neefiktivitu floatových výpočtů.

Tento skript je dostupný společně s příkladem originálního a zprocesovaného audio soboru ve složce `Python`.

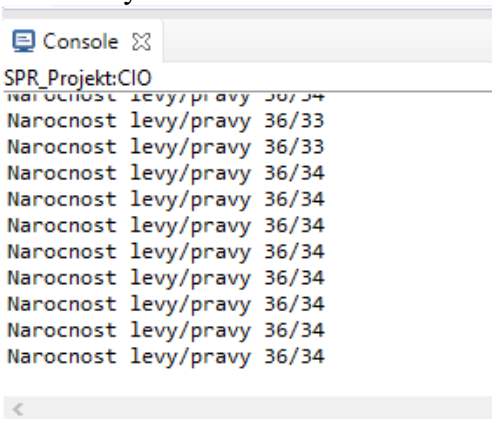
Implementace na DSP TMS320

Zdrojové soubory zvukového efektu se nachází v páru `.c / .h` souborů s názvem `effect`.

```
Int16 echoProcessing(Int16 inputSample, Uint16 channel)
{
    // circular read pointer
    Uint32 wp = writePtr[channel];
    Uint32 rp = wp + BUFFER_SIZE - delaySamples;
    if (rp >= BUFFER_SIZE)
        rp -= BUFFER_SIZE;

    Int32 d_delay = buffer[channel][rp];

    // normalize: x = input * normalization
    Int32 x = _smpy(inputSample, normalizationCoeff_fx);

    // s = alpha * delayed sample (high part -> Q15 result)
    Int32 s = _smpylh(alpha_fx, d_delay);

    // y = x + s
    Int32 y = _sadd(x, s);

    // d = x + beta * s (feedback into delay buffer)
    Int32 bs = _smpylh(beta_fx, s);
    Int32 d  = _sadd(x, bs);

    // write back into circular buffer
    buffer[channel][wp] = d;

    wp++;
    if (wp >= BUFFER_SIZE)
        wp = 0;

    writePtr[channel] = wp;

    // output sample (Q15)
    return (Int16)(y >> 16);
}
```

Jednotlivé vzorky jsou zpracovávány funkcí `echoProcessing()`, jejíž vstupními argumenty jsou přečtený vzorek z audio kodeku a označení kanálu. Rozlišování kanálů je potřeba, aby byly ukládány vzorky do správných cyklických bufferů v případě, že chceme zpracovávat stereo (nebo více) kanálový signál. Z tohoto důvodu je použitý buffer dvourozměrný a jeho velikost je možné změnit pomocí direktiv preprocesoru.
Veškeré matematické operace, které je potřeba provádět v reálném čase, jsou prováděny ve fixed-point aritmetice pomocí intrinsic funkcí se saturací. Při manipulaci jsou vzorky uchovávány v 32bitové proměné, aby nedocházelo k zaokrouhlovacím chybám a až finální návratová hodnota je zpět převedena na 16 bitů.
Indexování kruhového bufferu je ošetřeno podmínkou `if` místo použití `modulo`, aby nebylo potřeba dělení, ale vzhledem k tomu, že použitá velikost bufferu je mocninou dvojky, tak by bylo možné nahradit tuto operaci bitovým posuvem.
Velikost bufferu byla vypočtena jako nejbližší vyšší mocnina dvojky minimálního počtu vzorků, které je potřeba uchovat pro delay 200 ms při zadané vzorkovací frekvenci.

Vzorkovací frekvence byla nastavena na 48 kHz, i když zadání zmiňovalo použití frekvence 44,1 kHz z toho důvodu, že toto nastavení se jako jediné nepodařilo zprovoznit, i po konzultaci s vyučujícím, s originálními TI drivery aic23.



Obrázek 3: Měření náročnosti zpracování.

Náročnost zpracování je možné změřit pomocí statistických funkcí DSP BIOSu a jak je vidět na obrázku výše, je poměrně nízká, srovnatelná s jednoduchými aplikacemi které jsme dělali na PC cvičení.

```
EchoStatus echoSetParams(Uint16 delayMs, float alpha_coeff, float beta_coeff)
{
    // Compute delay in samples
    delaySamples = SAMPLE_FREQ * delayMs / 1000;

    if (delaySamples >= BUFFER_SIZE)
    {
        return ECHO_DELAY_OVERFLOW;
    }

    if (alpha_coeff < 0 || beta_coeff < 0)
    {
        return ECHO_NEGATIVE_COEFF;
    }

    if (alpha_coeff * beta_coeff > 1.0f)
    {
        return ECHO_UNSTABLE_COEFF;
    }

    float normalizationCoeff = 1.0f / (1.0f + alpha_coeff/(1.0f +
alpha_coeff*beta_coeff) );

    normalizationCoeff_fx = FLOAT2FIXED(normalizationCoeff);
    alpha_fx = FLOAT2FIXED(alpha_coeff);
    beta_fx  = FLOAT2FIXED(beta_coeff);


    return ECHO_OK;
}
```

Změnu parametrů efektu (α, β a delay) je možné udělat při běhu efektu pomocí funkce `echoSetParams()`. Ta zároveň kontroluje validitu zadaných hodnot (maximální délku bufferu, stability filtru a definiční obor koeficientů) a tuto informaci vrátí jako návratovou hodnotu. Při změně parametrů je také vypočítán nový normalizační koeficient. Vzhledem k tomu, že rozsah hodnot se může lišit a být malý, nebo větší než 1, byla zde použita float aritmetika. Díky k tomu, že tuto funkci voláme pouze při požadavku na změnu parametrů a z tasku s nižší prioritou, tak nám to nevadí.

K otestování efektu uživatelem je v souboru `hello.c` v tasku `tskCheck()` kontrolován stav DIP přepínačů a v případě, že dojde ke změně, přeladí se efekt na předem definované hodnoty. Tím můžeme docílit pouze jemného, nebo naopak výrazného echo efektu.

Tabulka 1: Stavy přepínačů.

Přepínač	OFF		ON	
DIP0	$\alpha = 0{,}3$		$\alpha = 0{,}6$	
DIP1	$\beta = 0{,}3$		$\beta = 0{,}6$	
Bin. kombinace	OFF+OFF	OFF+ON	ON+OFF	ON+ON
DIP2+DIP3	Delay = 50 ms	Delay = 100 ms	Delay = 150 ms	Delay = 200 ms

Ověření funkčnosti

A) Subjektivní poslechový test

Jelikož se jedná o zvukový efekt, nejrychlejší metodou ověření funkčnosti je poslechový test. V přiložených soborech v podsložce `Documentation` se nachází 3 nahrávky – jedna originální a dvě, které byly přehnány skrz DSP vývojový kit a nahrány zpětně zvukovou kartou PC.

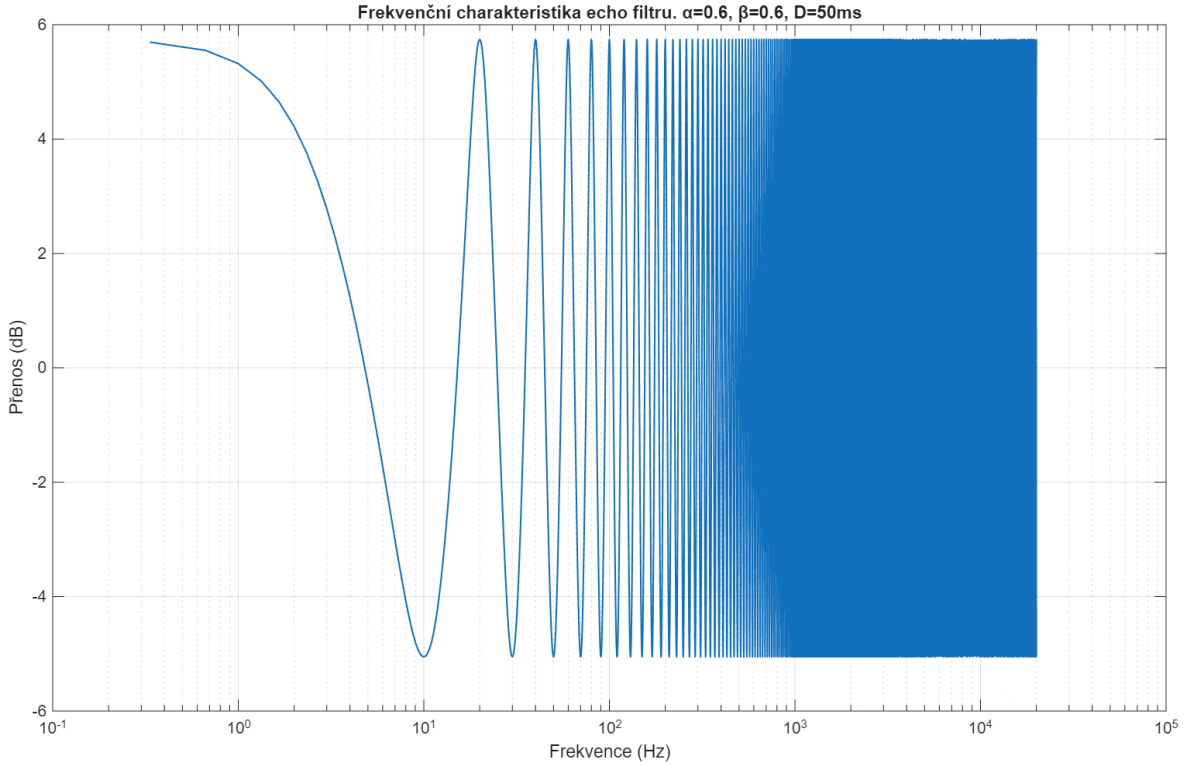
Tabulka 2: Podklady poslechového testu.

Název souboru	Nastavení efektu
`audio_original.wav`	Originální nahrávka
`dsp_buttons_not_pressed.wav`	$\alpha = 0{,}3$ $\beta = 0{,}3$ delay = 50 ms
`dsp_buttons_pressed.wav`	$\alpha = 0{,}6$ $\beta = 0{,}6$ delay = 200 ms

Poslechem a porovnáním těchto soborů můžeme usoudit, že efekt funguje správně a výsledky se shodují s předchozí realizací v Pythonu.

B) Měření frekvenční charakteristiky

Pomocí MATLAB skriptu (přiložený ve složce `MATLAB`) byla vykreslena přenosová charakteristika efektu.



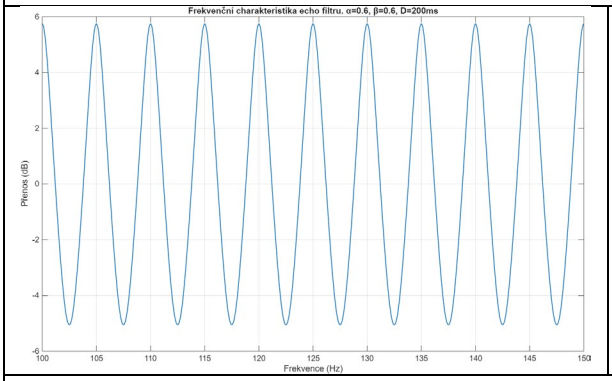
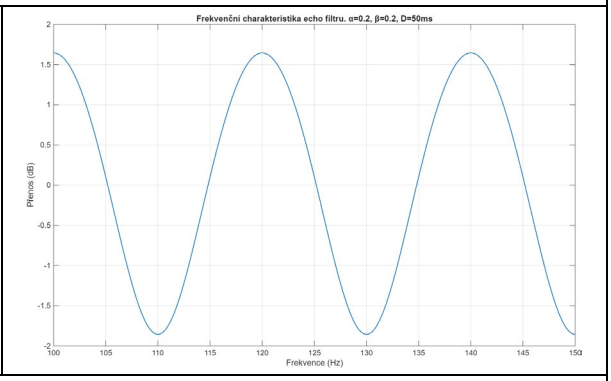
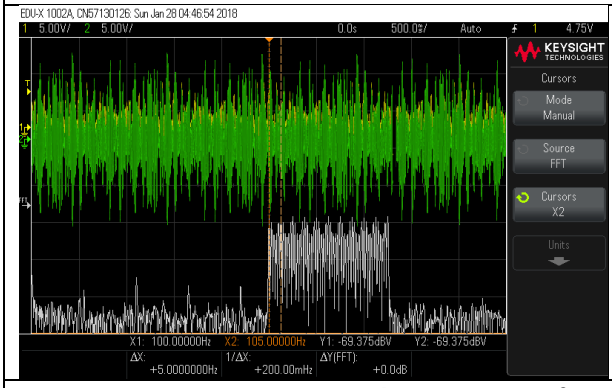
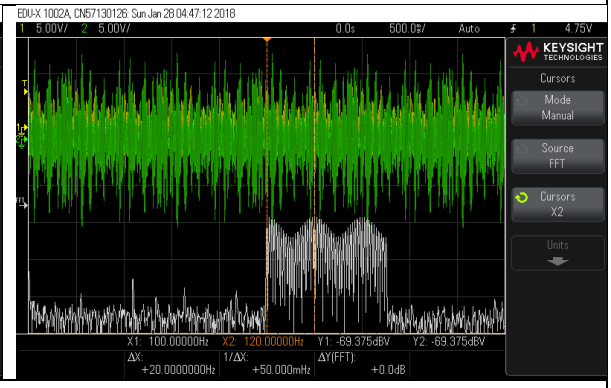
Obrázek 4: Simulovaná frekvenční charakteristika filtru v MATLABu ve frekvenčním rozsahu 1 Hz – 20 kHz.

Jak je vidět, tak tvarově odpovídá comb filtru. Vzdálenost jednotlivých špiček na ose x je dána nastavením zpožďovacího prvku

$$\Delta f = \frac{1}{t_{delay}}$$

Z praktických důvodů (kvůli rozlišení FFT) nebylo testováno celé audio spektrum, ale jenom část 100 – 150 Hz. V programu MULTISINE byl vygenerován bílý šum v této frekvenční oblasti a pomocí osciloskopu zaznamenáno spektrum na výstupu z DSP.

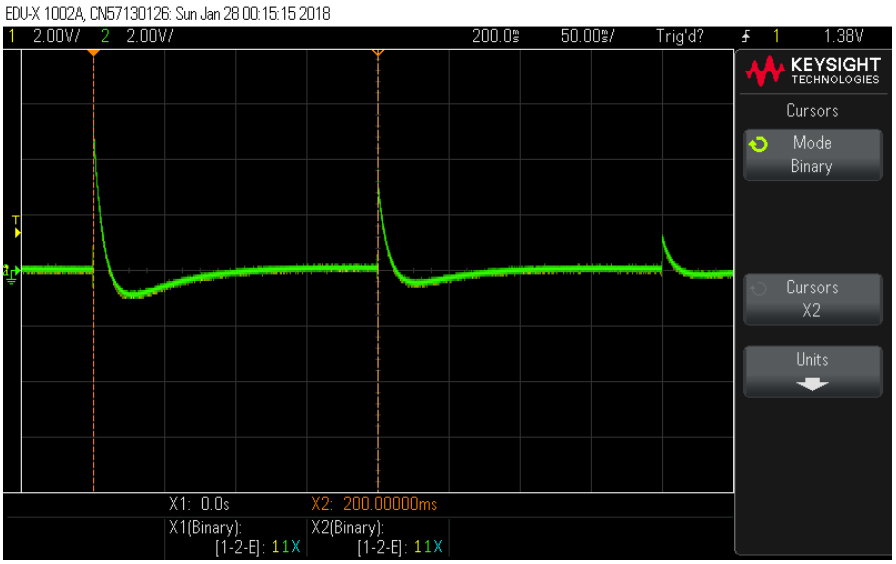
Tabulka 3: Porovnání simulace a měření frekvenční charakteristiky.

<table>
<tr><td colspan="2">Nastavení efektu</td></tr>
<tr><td>α = 0,6 β = 0,6 delay = 200 ms</td><td>α = 0,2 β = 0,2 delay = 50 ms</td></tr>
<tr><td colspan="2">Matlab simulace</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Měření</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Nastavení FFT: center frequency 100 Hz, span 200 Hz</td></tr>
</table>

Jak je vidět, tak naměřený přenos odpovídá tomu odsimulovanému jak tvarově, tak i vzdáleností špiček na ose x (odečteno pomocí kurzorů na obrazovce), tím pádem můžeme usoudit, že efekt pracuje tak, jak bylo předpokládáno.

C) Měření zpožďovacího členu

Aby bylo možné ověřit správné nastavení spožďovacího členu (zdali nastavená hodnota v ms opravdu odpovídá skutečnosti), byl přehrán zvukovou kartou jeden pulz. Nastavení efektu bylo α = 0,6 β = 0,6 delay = 200 ms.



Obrázek 5: Měření času zpoždění.

Jak je vidět na oscilogramu, doba zpoždění mezi opakujícími se pulzy je opravdu 200 ms, takže se efekt chová tak, jak bylo předpokládáno.

Závěr

V rámci projektu byl navržen a implementován digitální zvukový efekt echo / reverb na DSP TMS320. Přenosová funkce systému byla odvozena pomocí Masonova pravidla a byly stanoveny podmínky stability a normalizace signálu. Tyto podmínky byly následně využity při implementaci algoritmu na DSP.

Funkčnost efektu byla nejprve ověřena simulací v Pythonu a následně implementací v reálném čase na DSP s využitím fixed-point aritmetiky. Naměřená výpočetní náročnost byla nízká a odpovídá jednoduchým aplikacím v reálném čase.

Správné chování efektu bylo potvrzeno poslechovým testem, měřením frekvenční charakteristiky i ověřením zpožďovacího členu. Naměřené výsledky odpovídají simulačním datům, což potvrzuje správnost návrhu i implementace.