

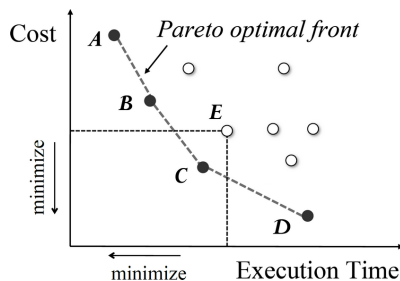
Opdrachten week 4 – Design space exploration

Als het goed is, dan heb je de eerste vier leerdoelen die beschreven staan in de [cursushandleiding](#) inmiddels bereikt. Deze week ga je werken aan leerdoel 5:

De student is in staat om te beslissen of bepaalde functionaliteit van een embedded applicatie beter op een soft core, op een hard core of in hardware geïmplementeerd kan worden.

Bij het ontwerp van een embedded systeem zijn er altijd veel mogelijke implementaties. Het verkennen van deze ontwerpruimte wordt in het Engels ‘design space exploration’¹ genoemd. De verschillende implementaties worden dan met elkaar vergeleken op een aantal relevante eigenschappen. Voor een embedded systeem zijn dat meestal: prestatie (snelheid), energieverbruik en kosten (productiekosten en/of ontwikkelkosten). Maar soms ook ontwikkeltijd, afmetingen en/of gewicht. Als de verschillende implementaties op n eigenschappen met elkaar vergeleken worden dan kunnen de systemen weergegeven worden in een n -dimensionale ruimte.

In [figuur 1](#) zie je een eenvoudig voorbeeld waarbij kosten en uitvoersnelheid van tien verschillende implementaties zijn weergegeven als punten in een 2-dimensionale ruimte.



Figuur 1: Een voorbeeld van een ontwerpruimte.²

¹ Zie eventueel: https://en.wikipedia.org/wiki/Design_space_exploration.

We willen in dit geval de kosten en de uitvoersnelheid minimaliseren. Je ziet dat er geen optimale oplossing beschikbaar is waarbij zowel de kosten als de uitvoersnelheid minimaal zijn. In de praktijk is dit ook vaak het geval. We zoeken dan de implementaties die zogenoemd Pareto-optimaal³ zijn. Een implementatie is Pareto-optimaal als het niet mogelijk is om één van de eigenschappen te verbeteren zonder dat een andere eigenschap verslechterd. In [figuur 1](#) zijn de zware punten Pareto-optimaal, maar de witte punten zijn dat niet. Voor implementatie *E* geldt bijvoorbeeld dat deze niet Pareto-optimaal is omdat implementatie *C* voor minder kosten een snellere uitvoersnelheid levert. Afhankelijk van het beschikbare budget en de wensen van de klant kunnen we vervolgens een keuze maken uit de Pareto-optimale implementaties. In dit geval kunnen we dus kiezen tussen implementaties *A*, *B*, *C* of *D*, een keuze voor één van de zes overige implementaties is niet zinvol (niet Pareto-optimaal).

Je snapt dat als we de verschillende implementaties op bijvoorbeeld zes verschillende eigenschappen met elkaar gaan vergelijken, er een complex optimalisatie-probleem ontstaat.

Als oefening gaan we drie verschillende implementaties van een audio-filter met elkaar vergelijken op slechts één eigenschap. De eigenschap die we gaan bestuderen is de benodigde tijd om één output sample te produceren nadat een nieuwe input sample is verkregen.

We maken hierbij gebruik van kennis die de Elektrotechniek studenten bij het vak [DIS10](#) hebben verworven. Ook zonder deze kennis zijn de opdrachten uit te voeren, maar als je meer verdieping zoekt, dan kun je die vinden in deze [practicumopdracht](#) van DIS10.

² Afkomstig uit: Toktam Taghavi, Andy Pimentel en Mojtaba Sabeghi. “VMODEX: A Novel Visualization Tool for Rapid Analysis of Heuristic-Based Multi-Objective Design Space Exploration of Heterogeneous MPSoC Architectures”. In: *Simulation Modelling Practice and Theory* 22 (mrt 2012).

³ Zie eventueel: https://en.wikipedia.org/wiki/Pareto_efficiency.

We maken bij het implementeren van het filter gebruik van fixed-point getallen in plaats van floating-point getallen. Informatie over fixed-point getallen kun je [hier](#) vinden.

Je gaat deze week leren hoe je:

- een audio-sigitaal kunt filteren met behulp van het DE1-SoC board door een FIR-filter te implementeren in C-code dat uitgevoerd wordt op een Nios II processor;
- de uitvoersnelheid van deze implementatie kunt bepalen met behulp van een hardware performance counter die opgenomen is in het Nios systeem.

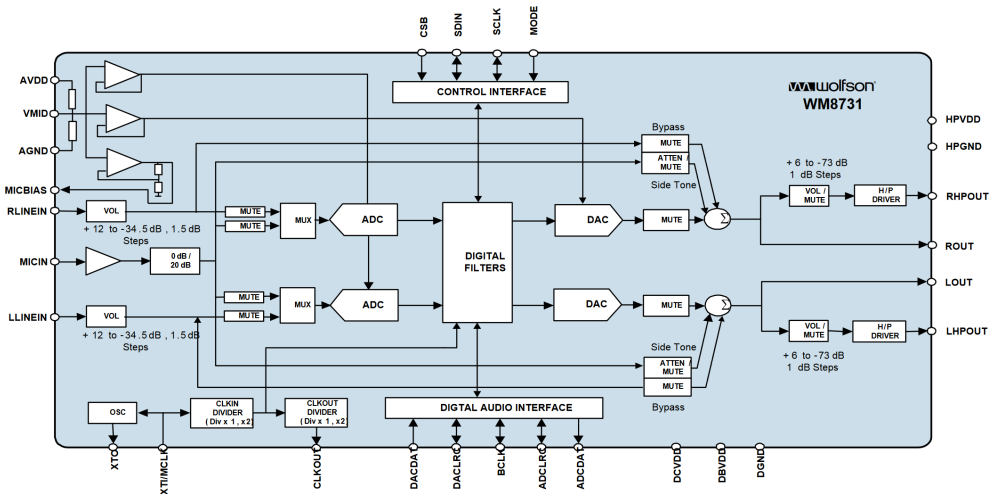
Je zou ook:

- een audio-sigitaal kunnen filteren m.b.v. het DE1-SoC board door een FIR-filter te implementeren in hardware door middel van VHDL;
- een audio-sigitaal kunnen filteren met behulp van het DE1-SoC board door een FIR-filter te implementeren in de hard core ARM processor met behulp van een programmeertaal naar keuze onder Linux;
- deze bovenstaande implementaties met elkaar kunnen vergelijken op uitvoersnelheid en andere criteria.

Om audio-signalen te bewerken met het DE1-SoC board kun je gebruik maken van de op het board aanwezige WM8731 codec (coder-decoder)⁴. Deze codec is een 24-bit stereo-audiocodec die kan werken met een bemonsteringsfrequentie van maximaal 96 ksp/s. De twee belangrijkste componenten binnen een codec zijn de analoog-digitaalomzetter (ADC) en de digitaal-analoogomzetter (DAC). Zoals te zien is in [figuur 2](#) bevat de WM8731 codec niet alleen twee ADC's en twee DAC's, maar ook verschillende versterkers, mixers en schakelaars.

De versterkingsfactoren worden gespecificeerd in decibel (dB), zoals te zien is in [figuur 2](#). Dit is een logaritmische grootheid die vaak wordt gebruikt in de elektrotechniek. De versterking (Engels: gain) in dB (G_{dB}) van een versterker kan

⁴ Zie: https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/WM8731.pdf.



Figuur 2: Vereenvoudigd blokschema van de codec.

als volgt worden berekend:

$$G_{dB} = 20 \cdot \log_{10} \frac{V_{out}}{V_{in}} \quad (1)$$

Hierin zijn V_{in} en V_{out} de ingangs- respectievelijk de uitgangsspanning van de versterker.

Het gebruik van een logaritmische schaal voor de versterking van audiosignalen is logisch, omdat de gevoeligheid van het menselijk oor voor geluidsdruk ook op een logaritmische schaal werkt.

De codec is via twee seriële bussen verbonden met de FPGA: een I²C bus⁵ en een I²S bus⁶. De I²C bus wordt gebruikt om de codec te configureren en aan te sturen (via de control interface, zie bovenaan [figuur 2](#)) en de I²S bus wordt gebruikt

⁵ I²C staat voor Inter-Integrated Circuit, meer informatie is te vinden op: <https://en.wikipedia.org/wiki/I%C2%B2C>.

⁶ I²S staat voor Inter-IC Sound, meer informatie is te vinden op: <https://en.wikipedia.org/wiki/I%C2%B2S>.

om de audio samples heen en weer te sturen (via de digital audio interface, zie onderaan [figuur 2](#)).

In de DE1-SoC User Manual is in [tabel 3-12](#) gegeven met welke pinnen van de FPGA de codec is verbonden. Merk op dat de klokingang van de codec (XTI/MCLK) verbonden is met de FPGA. De FPGA zal dus het kloksignaal voor de codec moeten genereren.

In het Intel University Program (UP) zijn drie IP-blokken opgenomen die gebruikt kunnen worden om de codec te gebruiken vanuit de FPGA:

- De *Audio Clock for DE-series Boards*, zie https://ftp.intel.com/public/pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Clocks/Altera_UP_Clocks.pdf kun je gebruiken om het kloksignaal voor de codec te genereren.
- De *Audio and Video Config*, zie https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf kun je gebruiken om de codec via de I²C-bus te configureren.
- De *Audio*, zie https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Audio_Video/Audio.pdf kun je gebruiken om audio-samples in te lezen en uit te sturen, via de I²S bus.

4.1 Download⁷ en bestudeer de documentatie van de hierboven genoemde IP-blokken zodat je begrijpt hoe je de codec vanuit de FPGA kunt gebruiken.

4.2 Om er voor te zorgen dat je programma's op de Nios II kunt uitvoeren en debuggen moet je de MSEL DIP switches weer terugplaatsen in hun oorspronkelijke positie. Zie [figuur 3-1](#) in de DE1-SoC User Manual.

⁷ In sommige browsers (b.v. Chrome) wordt het document in je download-directory geplaatst als je op de bovenstaande ftp-links klikt. In sommige andere browsers (b.v. Firefox) wordt het document in je browser geopend en kun je het van daaruit downloaden.

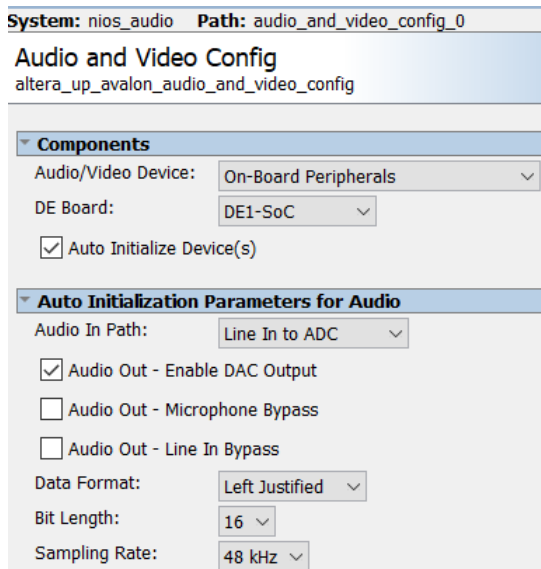
4.3 Implementeer een systeem m.b.v. Platform Designer met daarin de volgende componenten:

- *Audio Clock for DE-series Boards.*

Gebruik de 50 MHz klok `clk_0` als clock input en genereer een output clock van 12,288 MHz.

- *Audio and Video Config.*

Configureer deze component zoals weergegeven in [figuur 3](#). De codec wordt nu vanuit deze component geconfigureerd en het is niet meer nodig om dit vanuit software te doen.



Figuur 3: De gewenste configuratie van de Audio and Video Config component.

- *Audio.*

Configureer deze component met een memory mapped interface en 16-bit data width.

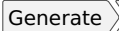

- *Nios II/f processor.*


We kiezen in dit geval *niet* voor een Nios II Classic processor maar

voor een tweede generatie Nios II processor. Dit doen we omdat de processor anders niet in staat is om de samplefrequentie van 48 ksp/s bij te houden. Voor deze processor is een licentie nodig als je het board standalone wilt gebruiken. Omdat we dat niet willen, is het niet erg dat we niet over een licentie beschikken.

- *On-Chip Memory*.
128 kB RAM geheugen.
- *Performance Counter*.
Deze component kun je gebruiken om de executietijd van een stuk code te meten. Zie https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/Embedded_Peripherals_IP_User_Guide_18_1.pdf#page=429
- *JTAG-UART*.
Om berichten op het console af te kunnen drukken.
- Twee *PIOs*.
Zodat je de 10 leds kunt aansturen en de 10 schakelaars kunt inschakelen.
- *System ID Peripheral*
Hiermee kan de software m.b.v. het BSP controleren of de juiste hardware in de FPGA is geladen.

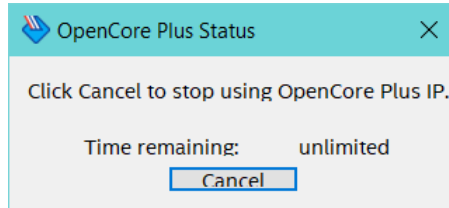
De top-level entity is gegeven in [listing 1](#).

Gebruik de menu-optie   in Platform Designer om het gegenereerde systeem te instantiëren in het top-level .vhd-bestand. Zorg dat de juiste verbindingen worden gemaakt en dat de pinnen correct worden toegewezen. Laad deze hardware vervolgens op het DE1-SoC board. Er verschijnt een window zoals weergegeven in [figuur 4](#).

Dit komt omdat we geen licentie voor de betreffende Nios II processor hebben. Zolang je *niet* op  drukt kun je deze hardware echter gewoon gebruiken.

```
ENTITY audio IS
  PORT (
    CLOCK_50 : IN STD_LOGIC;
    KEY : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
    SW : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
    LEDR : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    AUD_ADCDAT : IN STD_LOGIC;
    AUD_ADCLRCK : INOUT STD_LOGIC;
    AUD_BCLK : INOUT STD_LOGIC;
    AUD_DACDAT : OUT STD_LOGIC;
    AUD_DACLCK : INOUT STD_LOGIC;
    AUD_XCK : OUT STD_LOGIC;
    FPGA_I2C_SDAT : INOUT STD_LOGIC;
    FPGA_I2C_SCLK : OUT STD_LOGIC
  );
END audio;
```

Listing 1: De top-level entity van het audio systeem.



Figuur 4: Niet op klikken!

4.4 Maak een nieuw project met BSP aan in de Nios II Software Build Tools for Eclipse. Voer het programma uit dat gegeven is in [listing 2](#).

Als het goed is wordt het audiosignaal dat je op de blauwe audio jack connector aanbiedt, met 48 *ksps* gesampled door de codec en vervolgens weer naar buiten gestuurd m.b.v. de codec via de groene audio jack connector, gedurende 30 s. De hoogste 10 bits van de amplitude van het signaal worden weergegeven op de rode leds.


```
#include <altera_up_avalon_audio.h>
#include <altera_avalon_pio_regs.h>
#include <sys/alt_stdio.h>
#include <stdlib.h>
#include <system.h>

int main(void) {
    alt_up_audio_dev *audio_dev = ←
    ← alt_up_audio_open_dev("/dev/audio_0");
    if (audio_dev == NULL) {
        alt_printf("Error: could not open audio ←
    ← device\n");
        return -1;
    } else
        alt_printf("Opened audio device\n");

    const int run_time_in_seconds = 30;
    const int run_time_in_samples = ←
    ← run_time_in_seconds * 48000;
    int sample_count = 0;
    do {
        int fifospace_right = ←
    ← alt_up_audio_read_fifo_avail(audio_dev, ←
    ← ALT_UP_AUDIO_RIGHT);
        if (fifospace_right > 0) { // check if data is ←
    ← available
            sample_count++;
            // read audio buffer
            unsigned int r_buf = ←
    ← alt_up_audio_read_fifo_head(audio_dev, ←
    ← ALT_UP_AUDIO_RIGHT);
            IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, ←
    ← abs((short)r_buf)>>5); // light up the leds
            // write audio buffer
            alt_up_audio_write_fifo_head(audio_dev, ←
    ← r_buf, ALT_UP_AUDIO_RIGHT);
        }
    }
```

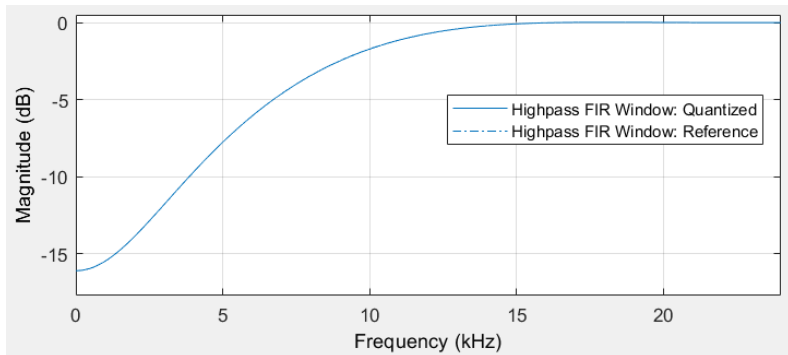
```

    int fifospace_left = ←
    ↪ alt_up_audio_read_fifo_avail(audio_dev, ←
    ↪ ALT_UP_AUDIO_LEFT);
    if (fifospace_left > 0) { // check if data is ←
    ↪ available
        unsigned int l_buf = ←
    ↪ alt_up_audio_read_fifo_head(audio_dev, ←
    ↪ ALT_UP_AUDIO_LEFT);
        alt_up_audio_write_fifo_head(audio_dev, ←
    ↪ l_buf, ALT_UP_AUDIO_LEFT);
    }
    } while (sample_count < run_time_in_samples);
    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0); // ←
    ↪ switch off the leds
}

```

Listing 2: Het programma `audio_loop.c`

4.5 We willen nu het signaal op het rechter audiokanaal filteren. We willen alleen de hele hoge tonen behouden en een eenvoudig filter implementeren dat is gegeven in [figuur 5](#).



Figuur 5: De overdrachtskarakteristiek van het gewenste hoogdoorlaatfilter.

We implementeren dit filter als een zogenoemd FIR (Finite Impuls Response) filter. De formule voor een FIR-filter is:

$$y[n] = \sum_{k=0}^N b_k \cdot x[n-k] \quad (2)$$

Waarin $y[n]$ de output samples zijn, b_k de filtercoëfficiënten zijn, $x[n]$ de input samples zijn en N de orde van het filter is.

Met behulp van MATLAB Filter Designer zijn de filtercoëfficiënten bepaald. Er is in Filter Designer gekozen voor een Highpass FIR-filter met een Hamming Window waarbij de doorlaatband geschaald is naar 0 dB. De parameters van dit filter zijn als volgt: $F_s = 48$ kHz, $F_c = 12$ kHz, $N = 8$ en fixed-point coëfficiënten in Q1.15 notatie⁸.

Meer informatie over FIR-filters kun je, indien gewenst, die vinden in deze [practicumopdracht](#). Informatie over fixed-point getallen kun je [hier](#) vinden.

De met MATLAB gegenereerde coëfficiënten kun je vinden in [listing 3](#).

```
#define N 8

// Q1.15: 16 bits fixed point fraction length is 15 bits
const short int B[N+1] = {
    0, -528, -2817, -6383, 24580, -6383, -2817, -528, 0
};
```

Listing 3: De coëfficiënten voor het filter uit [figuur 5](#) in Q1.15 notatie.

De C-code die het FIR-filter implementeert is gegeven in [listing 4](#).

```
static short int buffer[N+1] = {0}; // buffer for ↵
↵ input samples
// read audio buffer
```

⁸ Zie [https://en.wikipedia.org/wiki/Q_\(number_format\)](https://en.wikipedia.org/wiki/Q_(number_format)).

```
unsigned int r_buf = ←
↳ alt_up_audio_read_fifo_head(audio_dev, ←
↳ ALT_UP_AUDIO_RIGHT);
// Add sample to buffer
buffer[0] = r_buf;
// Apply filter calculation (convolution)
int output = 0;
for (size_t k = 0; k <= N; k++)
{
    output += buffer[k] * B[k];
}
// Shift old samples to the back of the buffer
for (size_t i = N; i >= 1; i--)
{
    buffer[i] = buffer[i-1];
}
// write audio buffer
alt_up_audio_write_fifo_head(audio_dev, ←
↳ ((output>>14)+1)>>1, ALT_UP_AUDIO_RIGHT);
```

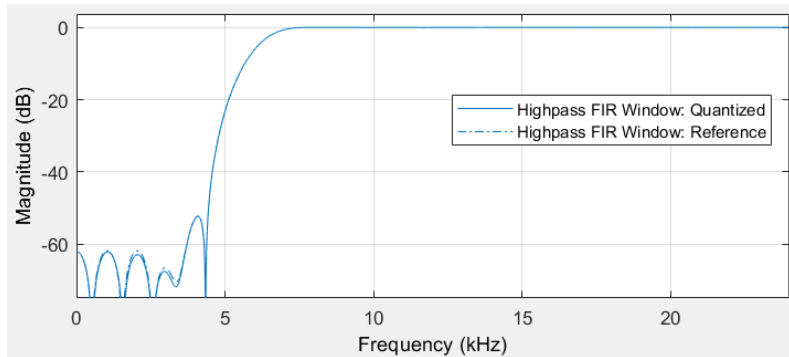
Listing 4: De implementatie van het filter uit [figuur 5](#).

Implementeer dit filter *alleen* voor het rechter audiokanaal. Als het goed is, dan kun je goed horen dat de lage tonen aan de rechterkant gedempt worden en links niet. Je kunt eventueel gebruik maken van je pc als functie-generator met behulp van het programma Soundcard Oscilloscope. Je kunt deze software downloaden vanaf: https://www.zeitnitz.eu/scope_en. De installatie wijst zichzelf. Je kunt ook een app gebruiken waarmee je jouw telefoon of tablet als functiegenerator kunt gebruiken⁹.

4.6 Als we de lagere tonen beter willen wegfilteren, dan moeten we het filter scherper maken door de orde te verhogen. De overdrachtskarakteristiek

⁹ Zelf gebruik ik <https://apps.apple.com/nl/app/audio-function-generator/id768229610> op mijn iPad.

voor hetzelfde type filter maar dan met $F_c = 6$ kHz en $N = 50$ is gegeven in [figuur 6](#).



Figuur 6: De overdrachtskarakteristiek van een scherper hoogdoorlaatfilter.

De met MATLAB gegenereerde coëfficiënten kun je vinden in [listing 5](#).

```
#define N 50

// Q1.15: 16 bits fixed point fraction length is 15 bits
const short int B[N+1] = {
    -24, 0, 30, 53, 48, 0, -79, -143, -127,
    0, 195, 338, 290, 0, -419, -711, -602, 0,
    877, 1520, 1344, 0, -2377, -5135, -7341, 24552, -7341,
    -5135, -2377, 0, 1344, 1520, 877, 0, -602, -711,
    -419, 0, 290, 338, 195, 0, -127, -143, -79,
    0, 48, 53, 30, 0, -24
};
```

Listing 5: De coëfficiënten voor het filter uit [figuur 6](#) in Q1.15 notatie.

Implementeer dit filter *alleen* voor het rechter audiokanaal. Je zult ontdekken dat dit niet het gewenste effect geeft. Ook het linker audiokanaal wordt vervormd. De Nios II processor is namelijk niet in staat om snel genoeg het volgende output sample te berekenen. Dit is niet verbazingwekkend als je

bedenkt dat de processor slechts $50\text{ MHz}/48\text{ kHz} = 1041$ clockcycles de tijd heeft om het volgende output sample te berekenen.

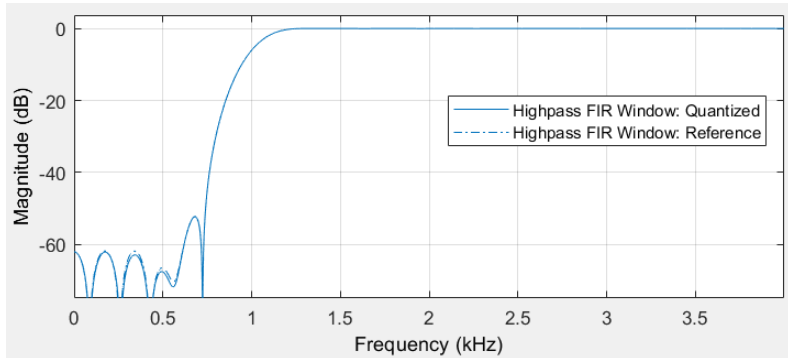
4.7 We kunnen wel scherpere filters implementeren als we de samplefrequentie verlagen. Dit zouden we kunnen doen door de hardware aan te passen. Maar het blokje *Audio and Video Config*, zie https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf kun je ook vanuit software configureren. Zoek in de datasheet van de WM8731 codec, zie https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/WM8731.pdf, op welk register je moet aanpassen om een samplefrequentie van 8 kHz in te stellen. Configureer dit register via de software met behulp van de volgende functies uit de BSP:

- `alt_up_av_config_open_dev;`
- `alt_up_av_config_write_audio_cfg_register;`
- `alt_up_av_config_read_ready.`

Als we de samplefrequentie verlagen, dan verlagen we ook de cut-off frequentie van het filter. De overdrachtskarakteristiek van het filter met de coëfficiënten uit [listing 5](#) bij een samplefrequentie van 8 kHz is gegeven in [figuur 7](#).

Als het goed is, functioneert het filter nu wel. Doordat de codec een anti-alias-filter gebruikt met een cut-off frequentie van $\frac{1}{2}F_s = 4\text{ kHz}$, worden alle signalen met een frequentie hoger dan 4 kHz op het rechter en linker kanaal geblokkeerd. Test met een functiegenerator dat een signaal van:

- 500 Hz alleen op het linkerkanaal hoorbaar is;
- 2 kHz zowel op het linker- als op her rechterkanaal hoorbaar is;
- 5 kHz op geen enkel kanaal hoorbaar is.



Figuur 7: De overdrachtskarakteristiek van een scherper hoogdoorlaatfilter met $F_c = 1$ kHz bij $F_s = 8$ kHz.

4.8 In de hardware hebben we een blokje genaamd *Performance Counter* opgenomen. Deze component kun je gebruiken om de executietijd van een stuk code te meten. Zie: https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/Embedded_Peripherals_IP_User_Guide_18_1.pdf#page=429

Maak gebruik van deze hardware en de BSP om te meten hoe lang de filterbewerking op het rechterkanaal en hoe lang het doorgeven van het linker kanaal duren. Bij een juiste implementatie verschijnt na 30 s het volgende performance report in de *Nios II Console*:

```
--Performance Counter Report--
Total Time: 29.9837 seconds (1499183661 clock-cycles)
+-----+-----+-----+-----+-----+
| Section      | %   | Time (sec)| Time (clocks)|Occurrences|
+-----+-----+-----+-----+-----+
|right        | 66.9| 20.06412| 1003205793| 240000|
+-----+-----+-----+-----+-----+
|left         | 1.44| 0.43200| 21600014| 240000|
+-----+-----+-----+-----+-----+
```

4.9 Deze opdracht is optioneel en mag ook als (deel van) een eindopdracht gebruikt worden.

Als je een scherp filter wilt implementeren bij een samplefrequentie van 48 kHz zul je hardware moeten gebruiken die sneller is dan de Nios II processor. Er zijn twee voor de hand liggende mogelijkheden:

- implementeer de filterberekening in VHDL;
- maak gebruik van de HPS om het programma uit te voeren.