KU LEUVEN

Faculteit Industriële Ingenieurswetenschappen
Systeemontwerp met HDL
S. Verslype

Verslag spectrogram

Academiejaar 2019 – 2020

Robin Nollet, Sebastian Vantomme, Ine Vanderhaeghe

Verslag: bouwen van een spectrogram op een FPGA met behulp van de VHDL-taal

Inhoudsopgave

1 Probleemstelling: een audiospectrogram

Doorheen het semester hebben we ons met ons groepje bezig gehouden met het bouwen van een audiospectrogram.

1.1 Wat is een spectrogram?

Het kan handig zijn om snel de frequentieinhoud van een signaal te zien, naar analyse toe. Gezien een signaal ook vaak veranderlijk is in de tijd, kan ook de frequentieinhoud veranderen in functie van de tijd. Er bestaat een methode om de frequentieinhoud van een signaal weer te geven in functie van de tijd. Dit heet een *Spectrogram*. Een spectrogram kan goed samengevat worden volgens de volgende definitie: het is een visuele representatie van de energie in elke frequentie uitgezet in de tijd. Een spectrogram heeft verschillende toepassingen. Enkele voorbeelden hiervan is bv. in de audio, of in de analyse van licht. Dit project focust specifiek op het analyseren van audiosignalen, hoorbaar voor het menselijk oor. Hiervoor nemen we standaard een range van $20Hz$ tot $20kHz$. In de audiowereld worden deze frequenties vaak de *pitch* genoemd van het signaal.

Mogelijke toepassingen van een spectrogram die hoorbare audiosignalen worden hieronder opgesomt:

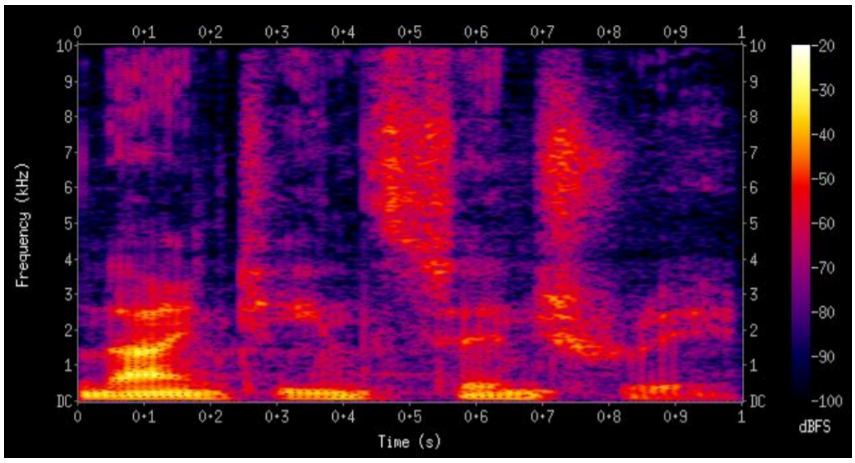
- analyse van Muziek(-instrumenten)
- analyse van Spraak
- analyse van Dierengeluiden
- analyse van RF modulatietechnieken

1.2 Hoe wordt een spectrogram voorgesteld?

De grafische voorstelling van een spectrogram zou eigenlijk in 3 dimensies moeten gegeven worden. Dit is echter wat lastig om dan op één oogopslag informatie te kunnen uit aflezen. Daarom worden in de literatuur vaak de volgende afspraken gevolgd:

- De tijd wordt op de horizontale as uitgezet;
- De frequentie wordt op de verticale as uitgezet;
- De sterkte van de frequentie wordt vaak weergegeven met behulp van kleurintensiteit of een kleurgradiënt.

In figuur 1 is een typische grafische voorstelling gegeven.



Figuur 1: Een typische grafische voorstelling van een spectrogram

1.3 Gebruikte methoden om een spectrogram te implementeren

In de literatuur kunnen 2 veelgebruikte methoden voor de implementatie van een Spectrogram gevonden worden.

1. Met behulp van de FFT: van het ingenomen signaal wordt de *Fast Fourier Transform (FFT)* genomen, wat een versnelde versie is van de *Discrete Fourier Transform (DFT)*. Dit toont de frequentieinhoud van het signaal. Een belangrijke eigenschap van deze transformatie is dat ze een discrete ingang (de audio) en een discrete uitgang (de frequentieinhoud van dit signaal) heeft.

2. Met behulp van een filterbank: hier wordt een reeks banddoorlaatfilters geïmplementeerd, die allemaal op hetzelfde signaal werken. Door na iedere filter dan te gaan kijken wat de energie van het signaal is, kan dit als een maat gebruikt worden voor de frequentieinhoud van ons ingangssignaal.

Naar implementatie toe heeft men dan een keuze, dit hangt af welke resolutie men wilt in de frequentie. Wil men een grote resolutie in de frequentie, gaat men eerder kiezen voor de aanpak met de FFT. Als men met een lage frequentieresolutie zich tevreden kan nemen, wordt er vaak geopteerd voor de implementatie met die meerdere filters.
Wij hebben hier gekozen voor de implementatie met een FFT, omdat deze een stuk flexibeler is in implementatie.

1.4 Afwerkingsgraad

Onze opdracht bestaat er uit om zelf een real-time spectrogram te maken van een audiosignaal dat aangeboden wordt door een FPGA. Dit spectrogram moet gevisualiseerd worden op een extern scherm via VGA of HDMI.

Als eerste moet er dus een analoog audiosignaal kunnen binnengelezen worden via de audiocodec chip. Deze zorgt zelf voor de digitalisatie van het signaal en geeft dit door aan de FPGA. Er moet dus communicatie mogelijk zijn tussen de codec chip en de FPGA.
Daarna moet het spectrum bepaald kunnen worden. Er worden een aantal samples genomen op het gedigitaliseerde audiosignaal. Daarop wordt een FFT uitgevoerd. Hierbij kan gewerkt worden met aansluitende of overlappende blokken van de samples. Als met overlapping gewerkt wordt zou de resolutie fijner zijn, maar het zou ook complexer zijn.
Ten laatste moet de verzamelde data na de FFT op een 2D map komen die het spectrogram voorsteld. Eens het volledige scherm gevuld is, moet de oudste meting verwijderd worden en schuift het spectrogram een tijdseenheid op via een FIFO.

In het volledige systeem moet rekening gehouden worden met de timing en mogelijke asynchrone signalen.
Er zal tijdens de evaluatie van het project met verschillende testsignalen gecontroleerd worden als het project juist ontworpen is. Voorbeelden van testsignalen zijn FM modulatie, frequentie sweep, blokgolf, DTMF signalen, ...

Er zijn enkele minimum vereisten in dit project:

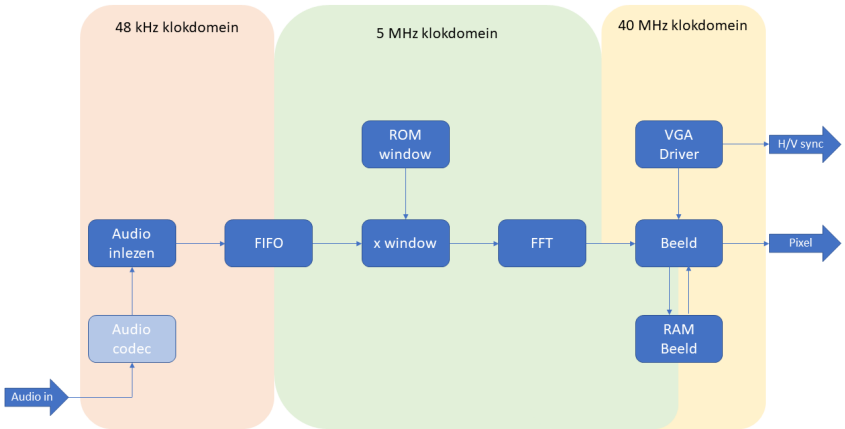
- Er wordt gewerkt met een FPGA bord naar keuze (Zedboard, Zybo, PYNQ, ...)
- Interface met audiocodec chip
- Real-time FFT op audio input
  - 1 kanaal (mono)
  - Aansluitende blokken van N samples
- Opstellen en updaten van spectrogram
- VGA/HDMI interface logica
- Beschikbaar stellen van spectrogram op een extern beeldscherm

Daarnaast zijn nog enkele mogelijkheden tot uitbreiding:

- Stereo spectrogram
- FFT op overlappende tijdssignalen
- 3D spectrogram
  - Amplitude als kleurintensiteit
- Full HD
- Audio loopback
  - Signaal terug naar buiten via line-out of headphone

2 Oplossingsstructuur

Voor dergelijke probleemstelling is de eerste stap om een *pipeline* op te stellen, een stappenplan van wat er met de data moet gebeuren van input tot aan output. Bij deze opdracht hebben we de pipeline toegepast gevonden in figuur 2.



Figuur 2: toegepaste pipeline

Zoals te zien valt op de figuur is het werk grotendeels op te splitsen in 3 grote delen, namelijk aan de hand van het klokdomein. Er hoeft dan gewoon via een dual-port-RAM een overschakeling te worden gemaakt tussen deze verschillende klokdomeinen. We hebben het werk opgesplitst in de volgende delen:

1. Audio inlezen. Kort samengevat is dit het inlezen van de audio via de audiocodec en stockeren in de FIFO.
2. Fourier nemen. Dit is de data uit de FIFO halen, er een window op toepassen en er daarna de fouriertransformatie op loslaten. De output van de FFT wordt dan doorgegeven aan Beeld.
3. Grafisch weergeven. Hier wordt de data gegeven van de FFT opgeslagen in een RAM, die dan ook wordt uitgelezen aan een andere kloksnelheid. Zo wordt de data visueel op het scherm gerepresenteerd.

2.1 Audio inlezen

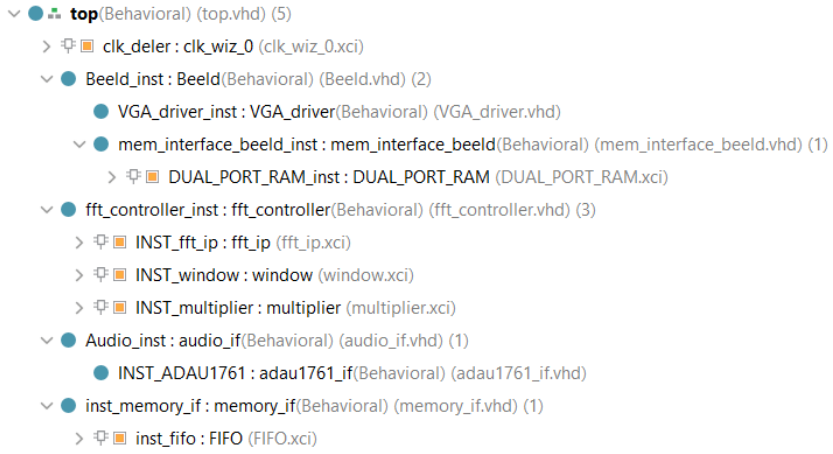
Om de audio binnen te lezen, zijn er 2 belangrijke bestanden: *adau1761* en *audio_if*. Deze bestanden lezen de data binnen en genereren daar de juiste clock signalen voor. In het top level wordt daar dan een audio loopback van gemaakt om de data die binnengelezen werd ook weer naar buiten te sturen. De data die binnengelezen werd wordt ook naar een FIFO gestuurd, zodat deze op de juiste momenten binnengelezen kan worden in de fft controller.

2.2 Fourier nemen

2.3 Grafisch weergeven

Om de berekende resultaten dan ook grafisch weer te geven, hebben we gebruik gemaakt van VGA. Om dit wat beter beheersbaar te houden is de architectuur opgesplitst in *VGA_Driver* en *mem_interface_Beeld*. *VGA_Driver* heeft als functie om Hsync en Vsync te genereren. Daarnaast genereert het ook een X en Y, wat de huidige positie is van de pixel die moet worden doorgestuurd, tesamen met een flag die hoog komt als men in het actieve gebied zit van het beeld (dus niet in de porch of syncpulses).
mem_interface_Beeld zorgt met de gegeven X en Y waarde dat de juiste pixel naar buiten komt. Ook zorgt het voor de overschakeling tussen de verschillende klokdomeinen.

3 Implementatie



Figuur 3: Hierarchie van de implementatie

3.1 Audio inlezen

Er kan gebruik gemaakt worden van de audiocodec die aanwezig is op het Zedboard. Dit is de *adau1761*. Hiervoor moest een driver geschreven worden. Voor ons project hebben we een driver kunnen gebruiken die we gekregen hadden van Dhr. Verslype. Deze driver krijgt zowel serieel als parallelle data binnen. De samples van het linker en rechter kanaal zijn 24 bit breed. Deze worden verwerkt in de driver en er wordt een linker en rechter output gegenereerd die 24 bit breed zijn en een seriële data output.

Daarnaast moest een interface geschreven worden die de clock genereerd en de data die de driver binnenleest op een juiste manier door geeft naar het top level. Ook hier hebben we gebruik gemaakt van een interface die we gekregen hadden van Dhr. Verslype. Deze interface heeft een kloksignaal nodig van 100 MHz en 12,288 MHz. Deze genereren 3 nieuwe kloksignalen: de *master clock*, de *digital audio bit clock* en de *digital audio left-right clock*. Deze worden gebruikt in de adau1761 codec.

De configuratie van de audio interface werkt met het I^2C protocol, het doorgeven van audio data gebeurt via het I^2S protocol.

In het top level wordt daarna een audio loopback gemaakt. De data wordt ook doorgegeven aan een FIFO memory, zodat deze uitgelezen kan worden door de fft controller. De data wordt serieel doorgegeven, eerst moet deze dus geparallelliseerd worden en per 24 bits opgeslagen worden in de memory. Het memory is 2048 waarden groot, dus wanneer deze vol is wordt een flag hoog gezet zodat de fft controller weet dat alle waarden die in de memory staan geldig zijn.

3.2 Fourier nemen

3.3 Grafisch weergeven

Om de berekende resultaten dan ook grafisch weer te geven, hebben we gebruik gemaakt van VGA. Om dit wat beter beheersbaar te houden is de architectuur opgesplitst in *VGA_Driver* en *mem_interface_Beeld*.

VGA_Driver genereert Hsync en Vsync. Dit werkt vrij eenvoudig door 2 tellertjes bij te houden (in de code *Hteller* en *Vteller*). Deze tellertjes geven weer waar op welke horizontale lijn men zit, alsook de positie in de lijn. Zo wordt het scherm horizontaal van links naar rechts doorlopen, en per lijn van boven naar beneden. Het is vrij eenvoudig om zo de Hsync en Vsync pulses te genereren, door enkel gewoon te gaan kijken wat de waarde is van de respectievelijke tellers.
Extra uitgangen van dit blokje zijn VGA_X en VGA_Y, die de huidige positie is van de pixel die moet worden doorgestuurd. Erg eenvoudig, want hiervoor moeten we gewoon de tellertjes die Hsync en Vsync genereren naar buiten sturen. Ook wordt een flag gegenereert, die hoog komt als men in het actieve gebied zit van het beeld (dus niet in de porch of syncpulses).

mem_interface_Beeld zorgt voor de overschakeling tussen klokdomeinen en het beheer van de data bij uitlezen en schrijven. Hierin is de functionaliteit opgesplitst in 3 delen:

- Memory component: voor de overschakeling tussen de klokdomeinen hebben we gebruik gemaakt van een dual port RAM, een stuk geheugen die langs 1 kant met een bepaalde klok geschreven wordt, en langs de andere kant wordt uitgelezen met een andere klok. Qua adressering loopt dit geheugen in dezelfde richting als de VGA: eerst van links naar rechts horizontaal, daarna lijntje per lijntje verticaal. Daardoor heeft de linkerbovenhoek dus het laagste adres, en de rechterbovenhoek het hoogste adres. Om dit adres wat te gaan abstraheren, hebben we dit opgesplitst in een X en een Y. Het adres valt dan te berekenen via de volgende formule:

  $$A = X + lijnbreedte \times Y$$

  met X de horizontale pixelpositie, Y de verticale pixelpositie en A het adres.

- Writing to memory: De FFT heeft een uitgang van 2048 waarden. Gezien dit spectrum symmetrisch is, krijgt dit component slechts 1024 doorgestuurd. Een overzicht van wat ik door krijg vanuit de FFT:

  - new_entry_clk: de klok waarmee de data uit de FFT komt
  - new_entry: de data vanuit de FFT, serieel doorgegeven, laagste frequentie eerst
  - new_entry_valid: komt hoog als er data verstuurd wordt.
  - new_entry_counter: een teller die bijhoudt hoeveel data er al is doorgestuurd
  - new_entry_last: een vlag die hoog wordt als het laatste element over de datalijn komt

  De randen (assen e.d.) worden op geïnitialiseerd in dit geheugen. Deze mogen uiteraard niet overschreven worden, waardoor er een beperking is waar er geschreven mag worden in het geheugen. De data wordt geschreven in een circulaire buffer. Deze circulaire buffer ligt in een bepaald gebied van het scherm, zodat de randen niet overschreven kunnen worden. Door middel van constantes wordt bijgehouden waar deze circulaire buffer begint en eindigt.

  Ieder "frame"dat we krijgen van de FFT is dus 1024 elementen groot. Deze zouden verticaal afgebeeld moeten worden. Spijtig genoeg is het aantal elementen een stuk hoger dan we behaald hebben in resolutie van het scherm, waardoor we wat hebben moeten "downsamplen": bv. een waarde houden, en een waarde wegsmijten.

  Het adres waarnaar geschreven moet worden is volgens de volgende formule bepaald:

  $$Y_{schrijf} = \frac{pixelbreedte \times new\_entry\_counter}{1024}$$

vervolgens kan het adres berekend worden:

$$A_{schrijf} = X_{schrijf} + lijnbreedte \times Y_{schrijf}$$

waar $X_{schrijf}$ de huidige positie is van de schrijfpositie in de circulaire buffer. Deze incrementeert na ieder frame, en wordt terug op de startpositie gezet als de eindpositie bereikt is.

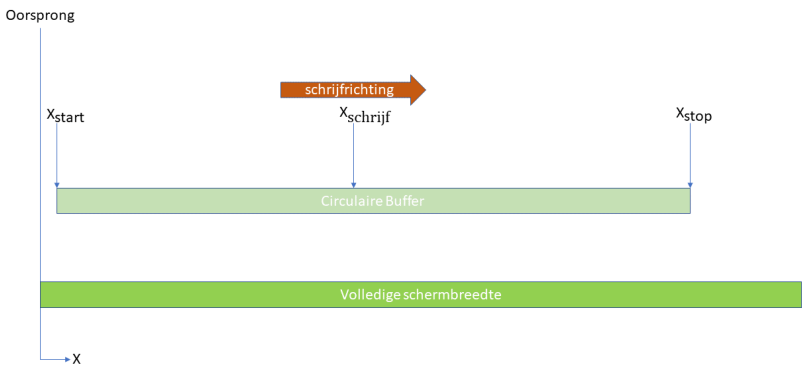
De data die doorgegeven wordt vanuit de FFT is 7 bits breed (range 0 - 127). De *color space* (= de kleuren die kunnen bereikt worden) van ons project is maar range 0 - 75. Daardoor moeten deze datawaarden herschaald worden:

$$D_{schrijf} = \frac{D \times 75}{127}$$

- Reading from memory: om de inhoud van dit geheugen dan ook te visualiseren zijn ook een aantal trucs toegepast. Zolang we niet in het circulair-buffer-gedeelte van het scherm zitten, is dit vrij eenvoudig. Het adres de te lezen pixel is gewoon als volgt:

$$A_{lees} = X_{VGA} + lijnbreedte \times Y_{VGA}$$

Maar als we in het circulair-buffer-gedeelte zitten, moeten we de adressering een tikkeltje anders doen. We definieren dan een nieuwe variabele X_{read}. Dit is de "virtuele"X-positie waar we moeten lezen in onze circulaire buffer.



Figuur 4: schematische representatie van de adressering

leesAdressering

Deze formules kunnen afgeleid worden uit de figuur 4

Als $X_{VGA} < X_{stop} - X_{schrijf} + X_{start}$ dan

$$X_{read} = X_{VGA} + X_{schrijf} - X_{start}$$

anders

$$X_{read} = X_{VGA} + X_{schrijf} - X_{stop} - 2$$

Bemerk: omdat $X_{schrijf}$ kan veranderen terwijl 1 frame van het scherm wordt opgebouwd, hebben we enkel gekeken in het begin van het scherm, en dan een vaste waarde genomen voor $X_{schrijf}$ de rest van dit frame. Om nu het effectieve adres te berekenen waar gelezen moet worden in het circulair-buffer-gedeelte, gebruiken we de volgende formule:

$$A_{lees} = X_{read} + lijnbreedte \times Y_{VGA}$$

De data die uit dit geheugen komt is in de range 0-75 en representeert de kleur van een pixel. Om deze waarde dan effectief om te zetten naar een kleur, wordt dit getal dan via onze *colorspace-vector* omgezet naar 4 bits per kleur (Rood, Groen en Blauw).

4 Besluit

Afsluitende tekst.