Onderwerp: Direct-Sequence Spread Spectrum

Leerdoelen

Aan het einde van dit onderwerp ben je in staat om:

- corellatie van twee signalen binnen GNU Radio uit te voeren en te begrijpen wat dit betekent;
- een datasignaal te verspreiden over een grotere bandbreedte met een bepaalde code;
- dit datasignaal te ontvangen en te decoderen.

Voorbereiding op de les

- Lees de hoofdstukken van PySDR zoals vermeld in de planning op de wiki
- Bekijk de introductie videos op Brightspace.

Beoordeling

Maak als groep een screencast waarin jullie de uitwerking van elke opdracht van dit onderwerp onderbouwen. Zorg ervoor dat beide personen aan woord komen en dat je aan de volgende punten aandacht besteedt:

1. Leg uit wat je in python hebt gedaan
2. Onderbouw voor de GNU Radio Companion schemas met name de instellingen van de blokken die je hebt gebruikt. Waarom heb je deze instellingen gekozen?
3. Laat zien dat het werkt, dit mag een korte opname met je telefoon zijn waarin in duidelijk is dat jouw schema worden gebruikt en dat jouw PRN/Masker wordt gebruikt en ontvangen.

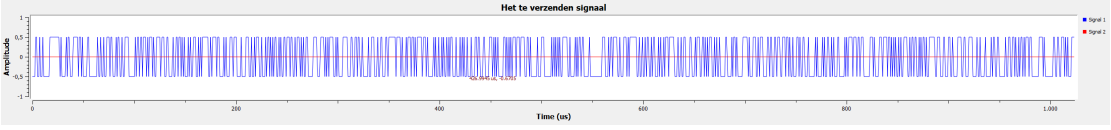
De screencast wordt beoordeeld volgens de rubric en beschrijving die op brightspace is geplaatst. Hou de deadlines van de cursushandleiding aan.

Opdrachten 26 pt

```
from gnuradio import digital
from math import ceil, log

## Mogelijke maskers met hoge autocorrelatie
#   Masker (degree 9)
#   0b100001000
#   0b100010110
#   0b100101100
#   0b100110111
#   0b100111011
#   0b101101101
#   0b110001001
#   0b110011000
#   0b110110000
#   0b110110101
#   0b111000010
#   0b111000111
#   0b111110001
#   0b111110100

#   Masker (degree 10)
#   0b1000000100
#   0b1000001101
#   0b1000110111
#   0b1010000110
#   0b1010001100
#   0b1010010001
#   0b1010011000
#   0b1011110010
#   0b1011111101
#   0b1100001001
#   0b1100111111
#   0b1110100110
#   0b1110110001
#   0b1111111100

def krijg_code(mask, seed):
    object = digital.glfsr(mask,seed)
    code = []
```

```
    for _ in range(2**ceil(log(object.mask(),2))-1):
        code.append(object.next_bit())
    return [2*(x-0.5) for x in code]

#bijvoorbeeld
masker = 0b100001000
code = krijg_code(masker,1)
print(code)
```

Listing 1: Python script om in radioconda omgeving uit te voeren. Zie `glfsr.py`



Figuur 1: Het te verzenden signaal

Opdracht 1: Voorbereiding [7 pt]

Eerst bouwen we de zender. Om een geschikte PRN te genereren, met goede correlatie-eigenschappen, gebruiken we het *GLFSR* blok. Voer de volgende stappen uit:

- Genereer met een Vector Source jouw data en zet het om naar een geschikt floating point signaal zoals in de video is besproken.
- Geneer ook een pseudo-willekeurige code met het *GLFSR* blok. Kies hiervoor een willekeurige maskerwaarde uit (dus niet allemaal de eerste kiezen) listing 1 met een degree van 9 of 10. De degree bepaalt direct de lengte van de code ($2^{degree} - 1$), en dus ook het vermogen van de correlatiepiek. De seed mag 1 blijven.
- Maak een variabele aan voor het repeat blok van de codebits en noem deze spcb voor *samples per codebit*. Zorg er nu voor dat elke databit wordt vermenigvuldigd met alle codebits. Voorbeeld: als spcb=2 en de lengte van de code is 255 dan heeft de totale code dus 2*255=510 samples. Je databit-samples moeten dan 510 maal herhaald worden. Maak zoveel mogelijk gebruik van variabelen om alles in te stellen, zodat het later gemakkelijk aangepast kan worden.
- De uitgang van de vermenigvuldiging zou dus een continue stroom aan codebits moeten zijn. Dit is het signaal wat later naar de Pluto wordt gestuurd, voor nu gaan we eerst alles testen zonder Pluto's. Zet het floating-point-signaal alvast

om naar een complex signaal. Als het klopt zie je in het tijdsdomein iets als figuur 1.

Nu starten we in hetzelfde schema met de ontvangstschakeling. Dit bevat dus de correlator. De correlatie doen we met een FIR filter. De coëfficiënten van het FIR filter zijn de bits van de PRN-code, maar dan in omgekeerde volgorde.

- Om deze coëfficiënten te kunnen verkrijgen kun je het pythonscript van listing 1 (`glfsr.py`) in gnuradio uitvoeren. Maak hiervoor een tijdelijk gnuradio schema aan, voeg een *Python Snippet* blok toe en plak hierin de inhoud van listing 1. Gebruik hetzelfde masker als eerder gekozen voor het GLSFR blok, en geef dit mee aan de `krijg_code()` functie binnen het *Python Snippet* blok. Run het schema. Kopieer de gegenereerde code en stop het in een *Variable* in je DSSS schema, genaamd `code`.
- Maak nog een *Variable* aan genaamd `filter_code` en plak hierin de python-snippet: `list(reversed([i for i in code for _ in range(spcb)]))`. De filter-code bevat nu dus de code-bits maar dan omgedraaid en elke bit herhaald met het aantal samples per codebit. Met een code van bijvoorbeeld `[-1,1,1]` en een `spcb=2` dan wordt `filter_code == [1,1,1,1,-1,-1]`.
- Gebruik een *Decimating FIR Filter* en geef bij `taps` de variabele `filter_code` op. Hiermee hebben we een correlator gemaakt die dus het ingangssignaal correleert met de GLSFR code.

Sluit de uitgang van opdracht 1 aan op de ingang van de correlator (FIR Filter). Als je alles goed hebt gedaan zou je nu in het tijddomein correlatiepieken moeten zien die veel groter zijn dan alle andere samples zoals in figuur 2 (exclusief de `bit` tags). Als elke databit uit 510 samples zou bestaan, dan zie je dus 1 piek per 510 samples. Het kan helpen om dan de time-sink meer samples te laten weergeven dan de standaard 1024. N.B. de correlatiepieken zouden even hoog moeten zijn als de lengte (in samples) van de code. Als dit niet het geval is gaat er iets fout.

Dit is de basis van DSSS. Nu is deze situatie natuurlijk niet zoals de werkelijkheid, want er zijn 0 effecten meegenomen. We gaan nu kijken wat er gebeurd als de signaal-ruisverhouding niet perfect is, en wanneer een frequentieverschil wordt toegevoegd.

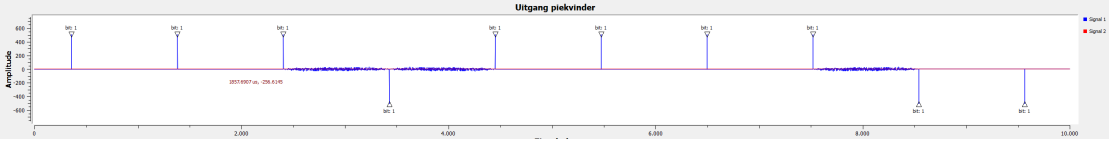
- Voordat het signaal van opdracht 1 naar de correlator gaat willen we dus wat ruis kunnen toevoegen. Maak gebruik van een *Add* en *Fast Noise Source* om ruis toe te voegen aan het signaal. Koppel de ruisamplitude aan een slider zodat je het real-time kunt aanpassen. Tot welke signaalruisverhouding kun je nog

correlatiepieken onderscheiden van de ruis? Onderbouw dit in de video met een berekening[a].

- Voeg nu ook een frequentieverschuiving toe voor het signaal bij de correlator komt. Maak ook deze aanpasbaar met een slider. Bekijk tot welke frequentieverschuiving de correlatiepieken zichtbaar blijven. Het helpt om een *Complex to Mag* blok te gebruiken, omdat de amplitude van de correlatiesamples niet zo snel afneemt.

Een ander voordeel van DSSS is dat je dus een makkelijke indicator hebt van de mate van frequentieafwijking.

[a] Je weet het ruisvermogen en het signaalvermogen, de SNR kun je dus berekenen in dB. Zie eventueel het PySDR boek over SNR.



Figuur 2: Uitgang correlator zonder ruis

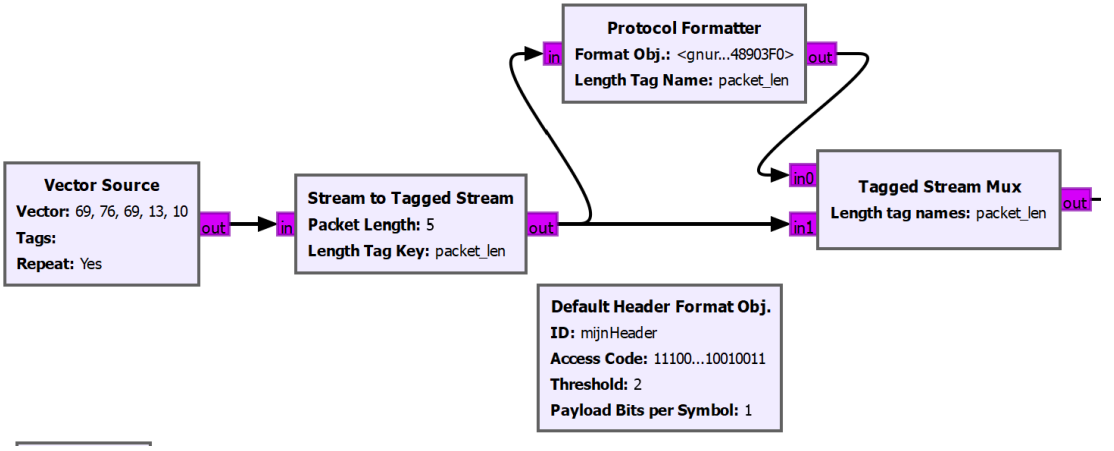
Opdracht 2: DSSS Naar docent [10 pt]

De docenten openen een schema wat tegelijk drie verschillende codes kan ontvangen op dezelfde frequentie. Het werkt op een draaggolf van 433,5 MHz en bekijkt een bandbreedte van 1 MHz. De docent gebruikt de laatste drie codes van 10 bits uit listing 1. Deze hebben een maximale kruiscorrelatie van 73, terwijl de autocorrelatie een maximum oplevert van 1023. Probeer je data op 1 van de drie uitgangen te laten zien.

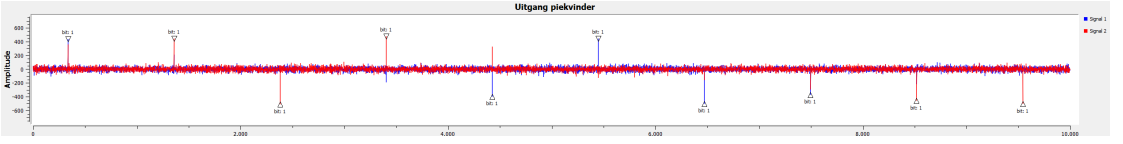
Er zullen een aantal blokjes in de chain toegevoegd moeten worden:

- Maak gebruik van een *Differential Encoder* zoals bij jouw PSK zender
- Maak gebruik van de blokjes uit figuur 3 om je data in een pakket te stoppen voordat het omgezet wordt naar bits. Lees eventueel de Frame Synchronisatie paragraaf voor extra uitleg over pakketten. De lengte van jouw pakket kun je even lang maken als jouw vector in de Vector Source. Als `Access Code` in het *Default Header Format Obj.* moet je `0xe15ae893` dus

11100001010111010111010001001 0011 gebruiken. De threshold en bits per symbool kun je beide op 1 zetten. De ID van dit blok gebruik je dan in het *Protocol Formatter* blok. Aan de ontvangende kant (bij de docent) wordt weer correlatie gebruikt d.m.v. het *Correlate Access Code - Tag* om deze access code te vinden.



Figuur 3: Een pakket maken



Figuur 4: Uitgang piekvinder blok na toevoegen ruis en een frequentieverschuiving.

Opdracht 3: DSSS ontvangen [9 pt]

Natuurlijk is het leuk om de verzonden data weer terug te kunnen zien. Om dit te bereiken heeft de docent een scriptje geschreven wat je in een *Python Blok* kunt stoppen. Daarnaast voegen we een costas loop toe om eventuele frequentieverschuivingen te corrigeren. De code van het scriptje is te vinden in `piekvinder2.py`.

- Voeg een *Costas Loop* toe voor de correlator (FIR filter). Je kunt als bandbreedte voor nu `62e-3` gebruiken en een order van `2`. In een later lab bespreken we dit blokje uitgebreid. Dit blokje corrigeert eventuele frequentieverschuivingen.

- Kopieer de code van `piekvinder2.py` en gebruik het als code voor een *Python Blok*. Voer hiervoor de volgende stappen uit:
  - Open de instellingen van het python blok en klik op Open in Editor. Dit opent de code van het blok waarschijnlijk in Notepad (onder Windows) of een andere editor. Mac gebruikers kunnen even om hulp vragen als er geen editor te selecteren valt.
  - Vervang deze code met die van `piekvinder2.py` en sla het bestand op. Als dit is gelukt zou het blokje nu **Piekvinder en Decimator** moeten heten in gnuradio met 1 parameter.

  Het blokje geeft alleen uitgangssamples wanneer er pieken worden gevonden. De parameter is de decimatiefactor, dus hoeveel samples tussen de correlatiepieken zitten (=lengte van de code in samples).

  Sluit de ingang van het blok aan op de correlator uitgang, en bekijk de uitgang van het blok in een timesink. Hier zou je nu je datastroom terug moeten kunnen zien. Je kunt de uitgang van dit blok ook bekijken in een constellatiesink met bijvoorbeeld 8 punten. Zonder frequentieverschuiving zou je nu puntjes moeten zien bij ± 1.
- De uitgang van het python blokje is nu 1 sample per databit en praktisch een PSK signaal. Je kunt nu de standaard technieken (met *Differential Decoder* en *Correlate Access Code - Tag* blokken) gebruiken om dit naar een telnet server te sturen en te laten zien in een terminal.
- Test het nu met twee pluto's. Hiervoor moet je natuurlijk geen ruis meer toevoegen of de frequentie te verschuiven. Test het als eerste met een spcb van 1. Als dit niet werkt kun je het verhogen in kleine stappen.
- Test het nu met twee groepen, dus 4 plutos, op dezelfde centerfrequentie, maar met andere codes. Je kunt de kruiscorrelatie van twee codes controleren door de codes in thonny te zetten, en `np.correlate(code1,code2,mode=full)` te gebruiken. Hoe kleiner de kruiscorrelatie tussen de twee gekozen codes hoe beter. Zonder een hoge kruiscorrelatie zou je dus als ontvanger kunnen kiezen welke van de twee signalen je wilt bekijken.

Deze opdrachten waren grotendeels een stappenplan. In de screencast zul je voor elke opdracht goed moeten onderbouwen wat je exact hebt gedaan en waarom de blokken in het schema dat voor elkaar krijgen. Lees vooral ook het PySDR detectie hoofdstuk goed door

voor betere theoretische onderbouwing van wat je aan het doen bent voordat je de video maakt.

Woordenlijst

GRC GNU Radio Companion. 1