ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Software Defined Radio
2020-2021

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

10. Data Operations.

Data Operations.

ARTESIS PLANTIJN HOGESCHOOL ANTWERPEN

Intro

- Tegenwoordig is de grote meerderheid van alle draadloze communicatie digitaal. Zelfs wanneer het om spraak of beeld gaat wordt deze meestal eerst omgezet in een of andere digitaal formaat.
- Transmissie en receptie zal dan ook veelvuldig gebruik maken van digitale modulatie types zoals de eerder besproken n-PSK, QAM, .. en spread spectrum technologiën.
- Dit creëert dan ook de noodzaak eveneens dergelijke bewerkingen te kunnen doen in de Software Defined Radio aanpak.
- In GRC zijn daarom eene hele reeks aan blocks voorhanden die ons kunnen helpen alle nodige bewerkingen te doen op data in een flowgraph

Data Operations.

Data types

- Des zover hadden we het voornamelijk of de data-types float en complex daar dit de types zijn die gebruikt worden bij de RF samples.

- Bij de te versturen of ontvangen data zijn logischerwijze andere types mogelijk.

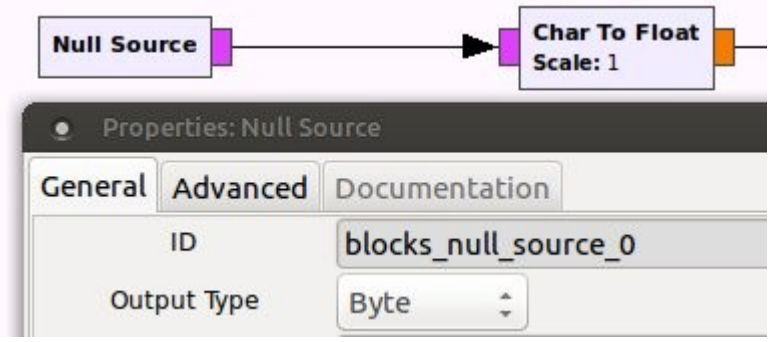
  - **Integer:** Deze kunnen 8, 16, 32, of 64 bits breed zijn.
    Elke hiervan heeft zijn eigen kleur in GRC
    Eveneens hebben ze een bijbehorende naam:

Data Operations.

Data types

- **Byte:** Een byte is 8 bits groot. Binnen GRC is een byte dan ook compatibel met **Char** uit de vorige reeks.

Men kan dus in en uitgangen van blocks zonder foutmelding verbinden wanneer de een byte en de andere char gebruikt (zie figuur)



Data Operations.

Data types

- **Byte:** Wat dan het verschil tussen char en byte?

  De oorsprong komt enigsinds voort uit de onderliggende programmeertaal C. In C bestaat ‘byte’ immers niet en is een 8-bits datatype steeds een ‘Char’

  Men gebruikt Char om te verwijzen naar het feit dat de binaire waarde verwijst naar een teken uit de ASCII tabel. Elk binair getal komt overeen met één teken uit deze tabel. Een bineare waarde blijft uiteraard wel in basis een getal. Diezelfde bineare waarde kan men net zo omzetten in een decimale waarde.

  Een Char is in GRC een ‘signed 8bit int’. Dit wil zeggen dat de decimale waarde ervan loopt van -128 tot +127

Data Operations.

Data types

- **Byte:** De benaming slaat dus meer op de toepassing ervan.

  Een consequentie is wel dat wanneer men bytes gaat wegschrijven naar een bestand met de file-sink dat deze als ASCII tekst terug te vinden zijn wanneer men het bestand opened met een tekst editor.

  Voorwaarde is uiteraard dan wel dat de bineaire waarde een zinvolle opeenvolging van ASCII waardes is.

Data Operations.

Data types

- Een belangrijk concept wanneer men werkt met data in een SDR context is het verschil tussen de continue stroom aan samples en de data vorm van de oorspronkelijke data en/of het bestemmingsformaat

- Een begrijpbaar voorbeeld hier is wanneer men een reeks van datapacket wil versturen. Op het packet-niveau spreken we bevoorbeeld van header-bytes en van packet-lengte.

  Wanneer meerdere packets echter naar een modulator gestuurd worden dan is dit voor de modulator zelf een onafgebroken reeds aan samples, die afhankelijk van het modulatie-type dan zelfs nog verder opgedeeld kunnen/moeten worden in afzonderlijke bits...

Data Operations.

Data types

- Nemen we als voorbeeld een BPSK modulator dan stuurt deze modulator bit per bit door.

  Ergens in ons flow-diagramma gaan we dus dit begrip van een ‘data-packet’ moeten introduceren…

  - **Vector:** Binnen GRC gebruiken we hiervoor de data **vector**

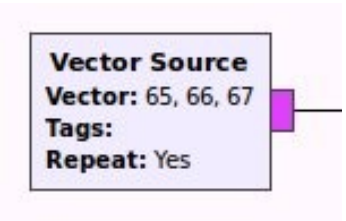
    Een vector is eigenlijk niet meer dan een verzameling van elementen van een bepaald type. Met ander woorden een array.

    Gezien python de achterliggende taal zijn er geen arrays maar worden de elementen van een vector opgeven als tuple of list

Data Operations.

Data Vector

- Een aantal blocken in GRC zijn voorhand om te werken met Vectoren.
- Vector Source: Dit block creëert een vector met element van het opgegeven datatype. Dit kan dus net zo goed Char, float als complex zijn.



In de figuur hiernaast staat als voorbeeld een Vector Source met data-type ‘byte’.

De opgegeven vector is hier de tuple (65,66,67)
Als char bekeken bevat deze vector dus de ASCII tekst ‘ABC’.

Data Operations.

Data Vector

- Vector Source: Een belangrijk begrip wanneer men spreekt over vectoren is de **vector-lengte**.

  Dit is hoeveel individuele elementen er verpakt worden in één vector. Deze vector lengte moet men dus ook opgeven in de vector source.

- Belangrijk hierbij om te begrijpen is dat wanneer men een vectorlengte opgeeft van 1 men eigenlijk een gewone sample-stroom heeft.

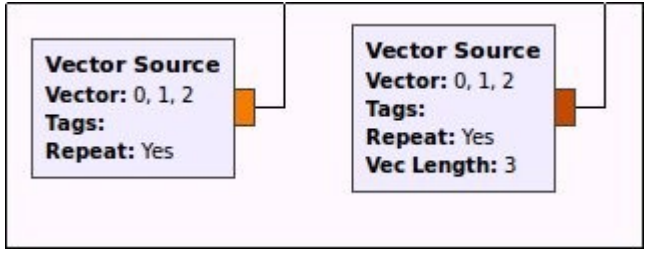
  Men kan stellen dat GRC eigenlijk altijd met vectoren werkt, doch dan wel vectoren met een lengte van één enkel element...

Data Operations.

Data Vector

- Vectoren -wanneer hun lengte groter is dan één- hebben ook een merkbaar kleuren verschil in GRC.

  Daar waar een vector-lengte van een, logischewijze, dezelfde kleur heeft van het data type heeft een vector met grotere lengte altijd een donkere versie van diezelfde kleur.



In het voorbeeld hiernaast ziet men het verschil tussen een vector-source met vectorlengte 1 en vectorlengte 3:

Beiden hebben echter hetzelfde type,
In dit geval float

Data Operations.

Data Vector

- Vectoren kunnen eens ze méér dan één element bevatten niet meer zomaar naar ander blocks verzonden worden. Blocks verwachten generiek een sample stroom, vectoren worden in de flowgraph echter als groepje data doorgegeven.
- Er zal m.a.w. nood zijn aan conversie
- Vector to stream : het vector tot steam block zal de elementen van een vector terug opsplitsen en doorsturen als gewone sample stroom

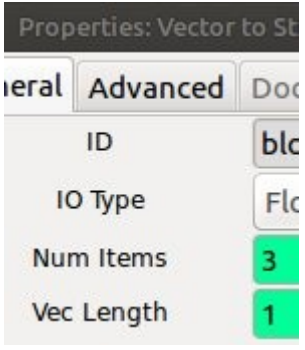
In het voorbeeld hiernaast verwacht het block vectoren van 3 elementen lang.

Data Operations.

Data Vector

- Vector to stream : De doel-stroom van dit block kan echter ook opnieuw een vector zijn maar dan met een andere vectorlengte.

Hiermee laat het block ook toe vectoren op zich om te zetten. In deze context is het belangrijk te begrijpen wat de properties betekenen in de config v/h block:



Num Items : lengte van de binnenkomende vector

Vec Length : lengte van de uitgaande vector

Vec Length = 1 is dus een normale sample stroom aan de uitgang. Een groter getal zal opnieuw een vector creëren

Data Operations.

Data Vector

- Logisch is ook de omgekeerde bewerking beschikbaar:
- Stream to vector : het stream to vector block creëert een vector op basis van de elementen uit de sample stroom volgende de opgegeven vectorlengte

- Gezien vectoren meerdere elementen bevatten zullen deze eveneen ook gelijktijdig doorgegeven worden in de flowgraph.

  Wil men de snelheid aanpassen met het throttle-block dan zal ook het trottleblock vectoren als elementen moeten nemen

Data Operations.

Data Vector

Het throttleblock heeft daartoe eveneens een ‘Vec Length’ parameter. Deze staat default op ‘1’ zoals we nu reeds kunnen begrijpen.

- Vector sink : Dit block wordt wel eens fout begrepen. Het doel ervan is beperkt.

Het hoofdzakelijk gebruik van het Vector sink block is als hulpmiddel voor het debuggen van code.

Na het runnen en stopzetten van de flowgraph zal de vector data die naar het block verzonden werdt beschikbaar zijn in het python object snk.data()

Binnen onze flowgraphs is het dus een block met weinig nut.

Data Operations.

Bits & Bytes

- Deszover maakten we enkel de stap van ‘bytes’ naar ‘packets’ en gelijkwaarig. Zoals eerder in het BPSK voorbeeld of bijvoorbeeld bij ON/OFF-keying hebben we echter met het versturen van individuele bits te doen.
- Afhankelijk van het modulatie type kunnen ook 2-bits in een keer verzonden worden (QPSK) of 3-bits per keer (8-PSK)
- Er is dus nood aan een omzetting van bytes naar bits alsook van bytes naar groepjes van bits. Bij ontvangst is uiteraard opnieuw de omgekeerde bewerking nodig.
- GRC kent evenwel géén ‘bit’ data-type, nevenmin een ‘n-bit’ data-type.

Data Operations.

Bits & Bytes

- Unpack K Bits : Het Unpack K bits neemt byte als type binnen en heeft eveneens ook byte als uitgaande type.

Deze uitgaande byte's hebben echter énkel op de plaats van de LSB een nuttige bit. De andere 7 bit's van de byte worden hierbij altijd op '0' geplaatst.
De decimale waarde van de uitgaande bytes zal dus altijd '0' of '1' zijn.

De op te geven 'K' parameter geeft aan hoeveel bits van de inkomende bytes gebruikt moeten worden. Geeft men 8 op dan zullen alle bits van de inkomende byte aan de uitgang verschijnen: bvb in=[0x0F] → out=[0,0,0,0,1,1,1,1]

één inkomende byte produceert hier dus 8 uitgaande bytes.

Data Operations.

Bits & Bytes

- Unpack K Bits : Geeft men voor K echter een kleiner getal op dan 8 dan worden énkel de onderste ‘K’ bits van de byte omgezet naar bits.

  Met ‘K’ gekijk aan 4 krijg men dus een omzetting van slechts de onderste 4 bits van de inkomende byte naar 4 uitgaande bytes met élk een relevante LSB bit

- Pack K Bits : Het Pack K Bits doet net het omgekeerde van dit vorige unpack bock. Het neemt bytes binnen doch kijkt énkel naar de LSB van deze bytes. Bij een K=8 worden 8 individule bits uit 8 opeenvolgende inkomende bytes verzamelt en als één volledige byte naar de uitgang gestuurd.

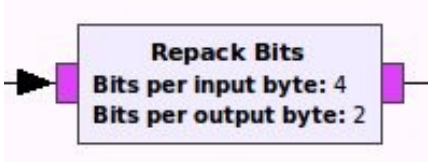
Data Operations.

Bits & Bytes

- Pack K Bits : Kiest men in dit block een K = n en K is kleiner dan 8 dan worden slechts n bits verzamelt en als byte naar buiten gestuurd. De ongebruikte bits worden hierbij op ‘0’ geplaatst.

  Een K=4 is dus een manier om van een bitstroom individiuele HEX tekens te maken. Voor elke 4 inkomende bytes gaat slechts één byte buiten met een waarde tussen 0x0 en 0xF

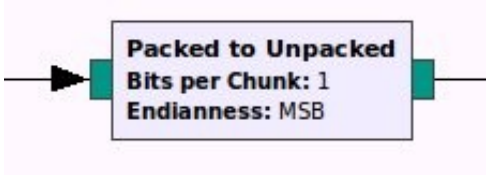
- Repack Bits : Dit block is als het ware het unpack and pack block na elkaar in één operatie. Het laat toe om bvb snel van bytes met 4 relevante bits naar 2 relevante bits te gaan



Data Operations.

Bits & Bytes

- Packed to unpacked :



Alle voorgaande operaties gebruiken byte als datatype
Het packed to uppacked block laat toe gelijkaardige operaties te doen met zowel byte, short als int.

Het output data-type is hierbij steeds gelijk aan het ingangsdatatype.

De parameter ‘bits per Chunk’ bepaalt hier hoeveel relevante bits er in de output byte’s, short’s of int’s zullen zitten.

Deze waarde op 1 geeft dus een enkelvoudige bitstroom als resultaat. Met het data type op byte verkrijgt men dezelfde werking als het ‘upack K bits’ block.

Data Operations.

Bits & Bytes

- Packed to unpacked : Dit block heeft hierboven nog enkele extra opties. Zo kan men de volgorde kiezen waarin de individuele bits geselecteerd worden uit het bron datatype. Deze keuze heeft men via de parameter ‘Endianness’.

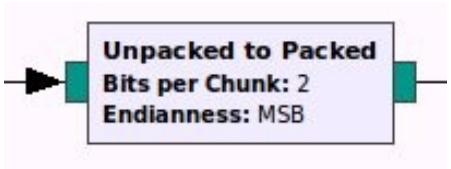
  Bij Endianness = MSB De omzetting zal starten met de meest significante bit en elke volgende output byte krijgt de volgende bit tot aan de LSB.

  Bij Endianness = LSB De omzetting zal starten met de minst significante bit en elke volgende output byte krijgt de volgende bit tot aan de MSB.

Data Operations.

Bits & Bytes

- Unpacked to packed : Opnieuw de omgekeerde operatie dit block.

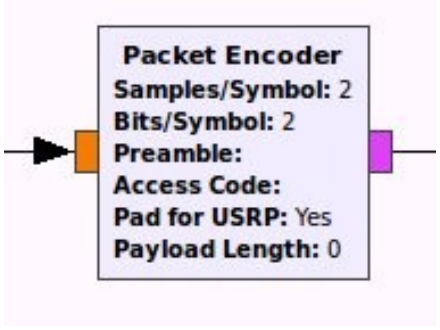


Van een bitreeks of groepjes bitreeks wordt terug een vol datatype element gemaakt.

Data Operations.

Bits & Bytes

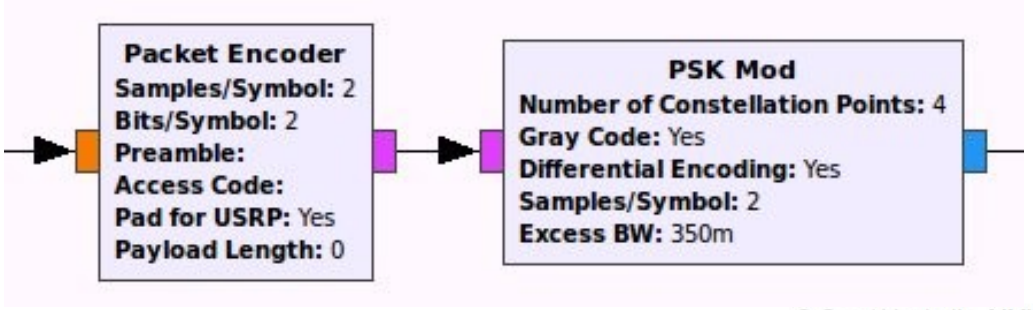
- Packet Encoder :



Het Packet Encoder block zet een sample stroom van het datatype Complex, float, Int, Short of Byte om in een reeks van bytes die exact het aantal nodige bit bevatten om een n-PSK, GMSK of QAM modulator aan te sturen.

Men kan hierbij direct het aantal Samples/Symbol ingeven alsook het aantal bits/Symbol

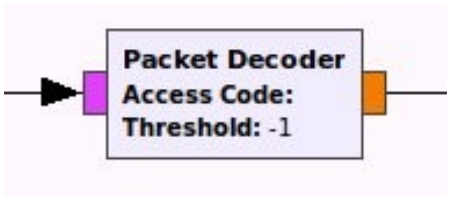
Merk op dat het aantal Constellatie Punten hierbij twee tot de macht van het aantal bits is.



Data Operations.

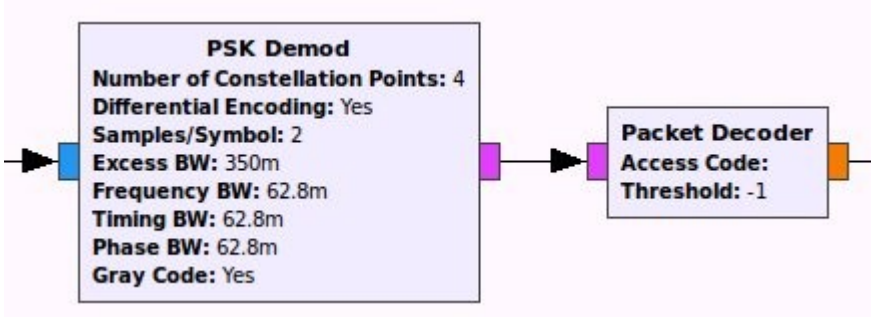
Bits & Bytes

- Packet Decoder :



Het Packet Decoder block zet de nuttige groepjes bits in een reeks bytes uit een n-PSK, GMSK of QAM modulator terug om in een sample stroom van het datatype Complex, float, Int, Short of Byte

Ook hier geeft men direct het aantal Samples/Symbol op, doch niet het aantal bits maar het aantal Constellatie Punten



Data Operations.

Sample reeksen

- Uit een reeks samples is het mogelijk dat niet alle data gewenst is.
  Ook hier heeft GRC enkele nuttig blokken voor.

- Skip Head : Het Skip Head block laat toe om de éérste sample uit een reeks te laten vallen en alle ‘n’ volgende samples te behouden. Dit is bvb nodig om dat de brondata packet is waarin er header-informatie zit dit louter informatief is of deze bijvoorbeeld sync data bevat.

In beide gevallen bestaat de kans dat deze sample niet verzonden dient te worden of niet in de ontvangstroom dient te zitten. Met het skip head block kunnen we deze sample dan uit de samplestroom halen.

Data Operations.

Sample reeksen

- Keep 1 in n : Het ‘keep 1 in n’ block is net het omgekeerde v/h skip head block.

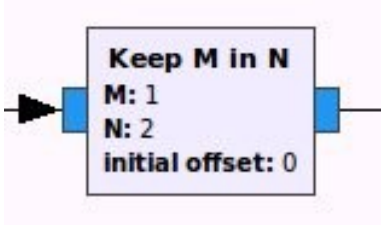
Hier behouden we énkel de éérste sample in een reeks van ‘n’ samples

Samen met het ‘skip head’ block is hiermee mogelijk van een sample reeks waarbij de éérste sample een ‘sync’ sample is deze op te splitsen in een ‘data’ uitgang met énkel de payload data (skip head) en een ‘sync’ uitgang met énkel sync data (keep 1 in n)

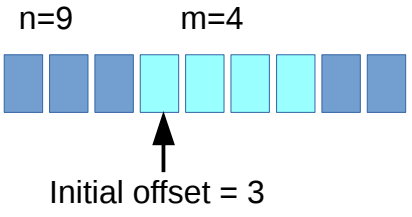
Data Operations.

Sample reeksen

- Keep m in n : Dit block is nog interessanter dan de vorige daar het toelaat uit een reeks van samples exact deze reeks samples te halen die men wil hebben. We behouden ‘m’ van de ‘n’ samples.



Een belangrijk parameter is hier de ‘initial offset’. Dit is de index vanaf waar de ‘m’ samples dienen geteld te worden. Dit laat toe zowel aan de start als het einde van een reeks samples te laten vallen en een stuk uit het midden te behouden.

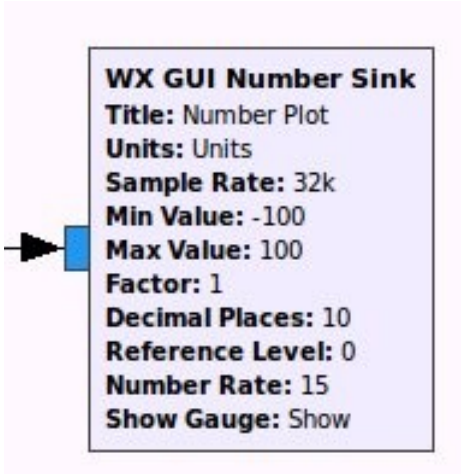


Dit is nuttig wanneer we bijvoorbeeld iets zoals een TCP/IP packet hebben waar we énkel de payload willen uit abstraheren

Data Operations.

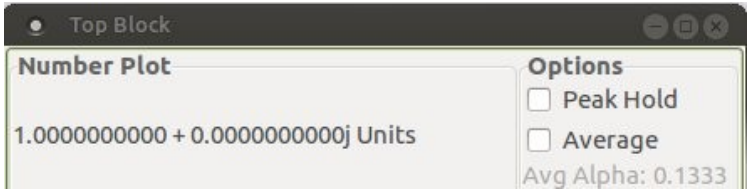
Visualisatie

- WX number sink :



bij het werken met data is het handig ook de numerieke waarde van een sample of sample reeks te kunnen visualiseren. Het WX GUI Number Sink block doet exact dat dit zowel van float:

als complex samples in het formaat a + bj:



ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

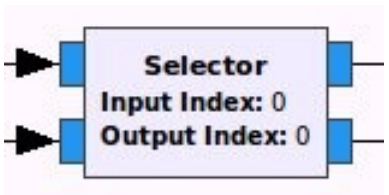
11. Routing

Routing

Stream routing

- In een SDR is het vaak nodig om met meerdere sample stromen te kunnen werken en deze dus ook te kunnen sturen van en naar de onderdelen die deze streams benutten of generen. Dit is wat we verstaan onder **routing**

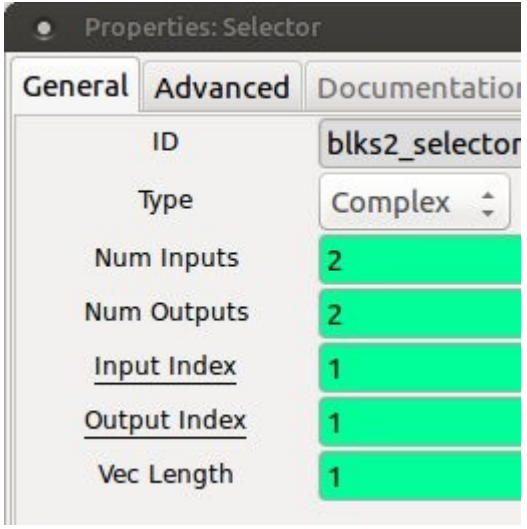
- Selector : Het selector block laat toe om enerzijds één sample stroom te kiezen uit een aantal ingangs sample stromen en anderzijds deze selectie naar één van meerdere mogelijke uitgangen te sturen.



Routing

Stream routing

- Selector : In de properties van dit block kan men kiezen hoeveel ingangen en hoeveel uitgangen het selector block heeft.



De parameter ‘input index’ kiest welke van de beschikbare ingangen geselecteerd wordt

De parameter ‘output index’ kiest vervolgens naar welke van de beschikbare uitgangen de stream gaat die met de inpux index gekozen werdt.

Routing

Stream routing

- Stream MUX : Het stream MUX block laat toe om de samples van twee of meer bronnen samen te voegen door ze te multiplexen.
- 

Multiplexen betekend dat afwisselend samples van de verschillende ingangen genomen worden en naar de uitgang gestuurd. Het aantal samples die telkens van elke bron genomen wordt geeft men op met de parameter ‘Lengths’.

Heeft de MUX bvb 2 ingangen en is Lengths = 1,1 dan wordt afwisselend één sample van de éne ingang genomen en één sample van de andere ingang.

Kiest men 1,9 dan zullen op elke 10 samples die aan de uitgang er één van de éérste ingang komen en 9 van de tweede ingang enz.

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

12. Tools

Tools

Flowgraph tools

- Ten slotte bekijken we nog enkele handige tools zowel in GRC, GNU radio, als voor SDR in het algemeen.

- Virtual sink :

De virtual sink en virtual source block combinatie is eigenlijk niets meer dan een manier om een verbinding tussen twee blocks te maken zonder dat de flowgraph als spageti gaat uitzien.

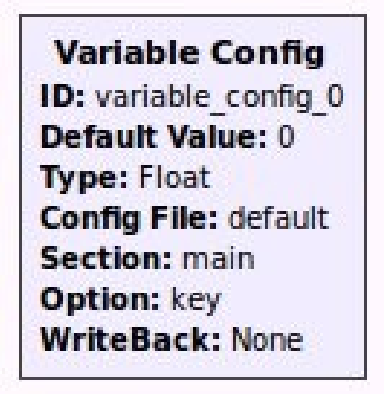
Virtual source :

Welke virtuele sink en source verbonnen is wordt bepaald door de opgegeven ‘Stream ID’.
Alle gelijke Stream-ID’s worden met elkaar verbonden

Tools

Flowgraph tools

- Variable config : Het variable config block laat toe om de waarde van een variable bij het opstarten van een bestand te halen. Wanneer de waarde van deze variable in runtime aangepast wordt zal deze aangepaste waarde vervolgens bij het afsluiten van de flowgraph weggeschreven worden in dit bestand.

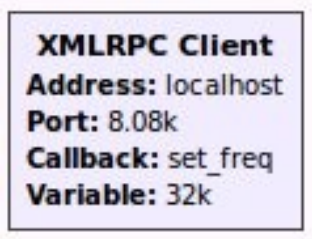


Dit laat m.a.w. de volgende keer de flowgraph op te starten met dezelfde instellingen als het ervoor stopte.

Tools

Flowgraph tools

- XMLRPC Client : Dit block creëert fantastisch extra mogelijkheden in GRC flowgraphs. Het XMLRPC Client block laat toe om de waarde van een variable in een flowgraph op afstand aan te passen via een TCP/IP verbinding.

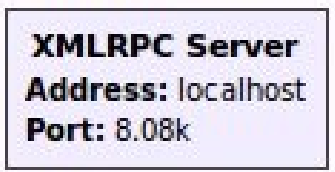


Dit gebeurd door middel van een callback functie. We kiezen de naam van deze functie zelf zodat bvb met een set van functies ons gehele flowgraph bestuurbaar wordt vanop afstand, dit door meerdere XMLRPC Client blocks in de flowgraph te integreren.

Tools

Flowgraph tools

- XMLRPC Server : Dit is het counter-part block van het XMLRPC Client block



Dit block creëert een XMLRPC server waarnaar het Client block kan connecten en de parameter uit de callback functie kan ontvangen.

Op deze wijze, bvb samen met de TCP sink en source blokken kunnen dus twee of meerdere flowgraphs op verschillende systemen op afstand volledig met elkaar geïntegreerd worden.

Tools

Filter Design

- GRC heeft ook een ingebakken Filter Design Tool, te vinden onder de ‘tools’ tab in hoofdmenu van GRC.
  Dit tool roept op een meer grafische manier achter-liggende python functies aan die taps van FIR en IIR filters kan berekenen.