SAE3.01 - Mettre en oeuvre un système de transmission

Ce compte rendu présente la mise en œuvre d'un système de transmission utilisant deux plateformes ADALM Pluto SDR sous MATLAB. L'objectif de ce projet est d'établir une communication sans fil entre un émetteur et un récepteur SDR (Software-Defined Radio), en exploitant les capacités de modulation, de transmission et de réception offertes par ces dispositifs.

Dans un premier temps, nous décrirons l'architecture du système, incluant la configuration matérielle et logicielle des ADALM Pluto. Ensuite, nous détaillerons la conception et l'implémentation du signal de transmission sous MATLAB, en précisant les paramètres de modulation et les protocoles utilisés. Enfin, une analyse des performances du système sera effectuée, en mettant en avant les défis rencontrés et les solutions adoptées pour optimiser la transmission.

Ce projet s'inscrit dans une démarche d'apprentissage et d'expérimentation des communications numériques par SDR, avec pour finalité d'approfondir la compréhension des transmissions radio et du traitement du signal en environnement MATLAB.

SOMMAIRE

1 - Présentation du hardware et software utilisé

A - ADALM Pluto SDR

L'ADALM Pluto SDR (Software Defined Radio) est un dispositif électronique compact et polyvalent conçu pour l'apprentissage et l'expérimentation des communications sans fil. Il permet d'émettre et de recevoir des signaux radio sur une large gamme de fréquences (généralement de 325 MHz à 3,8 GHz), tout en offrant la possibilité de traiter ces signaux via un ordinateur. Contrairement aux radios traditionnelles basées sur du matériel fixe, l'ADALM Pluto SDR s'appuie sur des logiciels pour configurer des protocoles de communication variés, tels que la modulation, la transmission numérique ou encore l'analyse du spectre. Il est particulièrement apprécié dans les domaines de la recherche, de l'enseignement et du prototypage, car il permet d'expérimenter facilement des concepts liés aux télécommunications, aux réseaux sans fil ou encore aux systèmes radar, le tout avec un simple câble USB et un logiciel comme MATLAB ou GNU Radio.

B - MATLAB et Simulink

Et dans notre cas, c'est MATLAB que nous utilisons, un environnement de programmation et un logiciel de calcul numérique largement utilisé pour l'analyse de données, la modélisation, la simulation et le développement d'algorithmes. Il se distingue par son langage de script simple et intuitif, particulièrement adapté aux opérations mathématiques complexes, à la visualisation de données et au traitement de signaux.

Simulink, quant à lui, est un environnement complémentaire intégré à MATLAB, dédié à la modélisation et à la simulation de systèmes dynamiques à l'aide de diagrammes de blocs. Il permet de concevoir des systèmes complexes (comme des chaînes de communication, des systèmes de contrôle ou des circuits électroniques) de manière graphique, en connectant des blocs fonctionnels représentant des opérations mathématiques, des sources de signaux ou des dispositifs physiques. Ensemble, MATLAB et Simulink offrent une plateforme puissante pour le prototypage rapide, le test d'algorithmes et la simulation de systèmes dans des domaines variés tels que l'ingénierie, la robotique ou encore les télécommunications pour ce projet.

2 - Analyse du modèle du transmetteur

A - Le fichier de configuration

Voici le contenu du fichier de configuration :

```
function SimParams = plutoradioqpsktransmitter_init
%% General simulation parameters
SimParams.Rsym = 0.2e6;         % Symbol rate in Hertz
SimParams.ModulationOrder = 4;    % QPSK alphabet size
SimParams.Interpolation = 2;      % Interpolation factor
SimParams.Decimation = 1;         % Decimation factor
SimParams.Tsym = 1/SimParams.Rsym;  % Symbol time in sec
SimParams.Fs   = SimParams.Rsym * SimParams.Interpolation; % Sample rate
%% Frame Specifications
SimParams.BarkerCode    = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];   % Bipolar Barker Code
SimParams.BarkerLength   = length(SimParams.BarkerCode);
SimParams.HeaderLength   = SimParams.BarkerLength * 2;            % Duplicate 2 Barker codes to be as a header
SimParams.Message       = 'Je suis un etudiant de RT-2025 Bilel Rohan';
SimParams.MessageLength   = length(SimParams.Message) + 5;
SimParams.NumberOfMessage = 100;                          % Number of messages in a frame
SimParams.PayloadLength   = SimParams.NumberOfMessage * SimParams.MessageLength * 7; % 7 bits per characters
SimParams.FrameSize     = (SimParams.HeaderLength + SimParams.PayloadLength) ...
  / log2(SimParams.ModulationOrder);                      % Frame size in symbols
SimParams.FrameTime     = SimParams.Tsym*SimParams.FrameSize;
%% Tx parameters
SimParams.RolloffFactor    = 0.5;                         % Rolloff Factor of Raised Cosine Filter
SimParams.ScramblerBase    = 2;
SimParams.ScramblerPolynomial        = [1 1 1 0 1];
SimParams.ScramblerInitialConditions   = [0 0 0 0];
SimParams.RaisedCosineFilterSpan = 10; % Filter span of Raised Cosine Tx Rx filters (in symbols)
%% Message generation
msgSet = zeros(100 * SimParams.MessageLength, 1);
for msgCnt = 0 : 99
  msgSet(msgCnt * SimParams.MessageLength + (1 : SimParams.MessageLength)) = ...
    sprintf('%s %03d\n', SimParams.Message, msgCnt);
end
bits = de2bi(msgSet, 7, 'left-msb')';
SimParams.MessageBits = bits(:);
% Pluto transmitter parameters
SimParams.PlutoCenterFrequency    = 868e6;
SimParams.PlutoGain           = 0;
SimParams.PlutoFrontEndSampleRate   = SimParams.Fs;
SimParams.PlutoFrameLength       = SimParams.Interpolation * SimParams.FrameSize;
% Simulation Parameters
SimParams.FrameTime = SimParams.PlutoFrameLength/SimParams.PlutoFrontEndSampleRate;
SimParams.StopTime  = 1000;
```

Il est facile de donner le contenu du fichier, mais que fait-il ?

Il est possible de le séparer en les 6 sections distinctes suivante :

- La définition des paramètres généraux de la simulation

Cette section définit les paramètres de base de la simulation du système de transmission QPSK. Le taux de symboles (Rsym) est fixé à 200 kHz, ce qui détermine la vitesse de transmission des données. L'ordre de modulation est de 4, correspondant à la modulation QPSK (Quadrature Phase Shift Keying). L'interpolation est définie à 2, ce qui augmente la fréquence d'échantillonnage (Fs), tandis que la décimation est à 1, signifiant qu'aucune réduction d'échantillons n'est appliquée. Le temps de symbole (Tsym) est calculé comme l'inverse du taux de symboles.

- La spécification de la trame

Cette section structure les trames de données à transmettre. Un code Barker bipolaire de longueur 13 est utilisé pour faciliter la synchronisation. L'en-tête de la trame est formé en dupliquant deux fois ce code Barker. Le message à transmettre est personnalisé : "Je suis un etudiant de RT-2025 Bilel Rohan", répété 100 fois avec des numéros incrémentés (de 000 à 099). Chaque caractère est codé sur 7 bits. La taille totale de la trame (FrameSize) est calculée en fonction de la longueur de l'en-tête, du message, et du schéma de modulation utilisé.

- La définition des paramètres du transmetteur

Cette partie configure les paramètres pour la transmission du signal. Un filtre en cosinus surélevé est utilisé pour réduire les interférences inter-symboles, avec un facteur de roll-off (RolloffFactor) de 0,5. Le scrambler, défini par une base de 2 et un polynôme spécifique, sert à disperser les séquences de bits répétitives afin d'améliorer la robustesse du signal. Le filtre a une portée de 10 symboles, assurant un compromis entre complexité et performance.

- La génération des messages

Cette section configure les paramètres propres à l'ADALM Pluto SDR. La fréquence centrale de transmission est fixée à 868 MHz, utilisée couramment dans les communications sans fil (par exemple, les réseaux IoT). Le gain de transmission est à 0 dB, ce qui signifie que le signal est envoyé sans amplification supplémentaire. La fréquence d'échantillonnage du Pluto correspond à celle de la simulation pour garantir la cohérence des données. La longueur des trames est ajustée en fonction du facteur d'interpolation.

- La définition des paramètres spécifiques à l'ADALM Pluto SDR

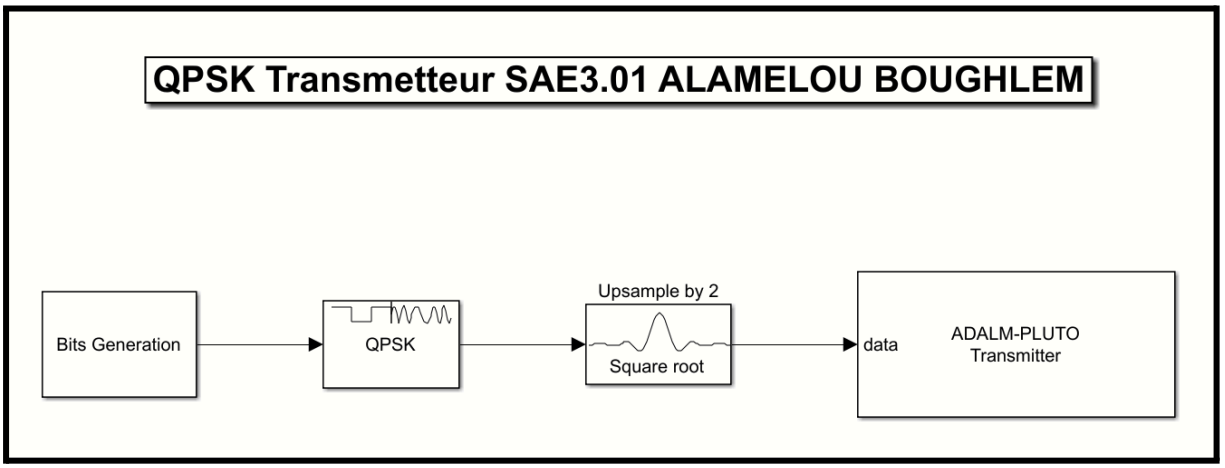
Cette partie configure les paramètres propres au matériel SDR (Software Defined Radio) ADALM Pluto. Elle définit la fréquence centrale de réception (PlutoCenterFrequency) à 868 MHz, une bande couramment utilisée pour des applications sans fil. Le gain du récepteur (PlutoGain) est fixé à 30 dB pour optimiser la réception du signal. La fréquence d'échantillonnage à l'entrée du Pluto est synchronisée avec celle de la simulation (PlutoFrontEndSampleRate), et la longueur des trames est ajustée en conséquence.

- Les paramètres de simulation

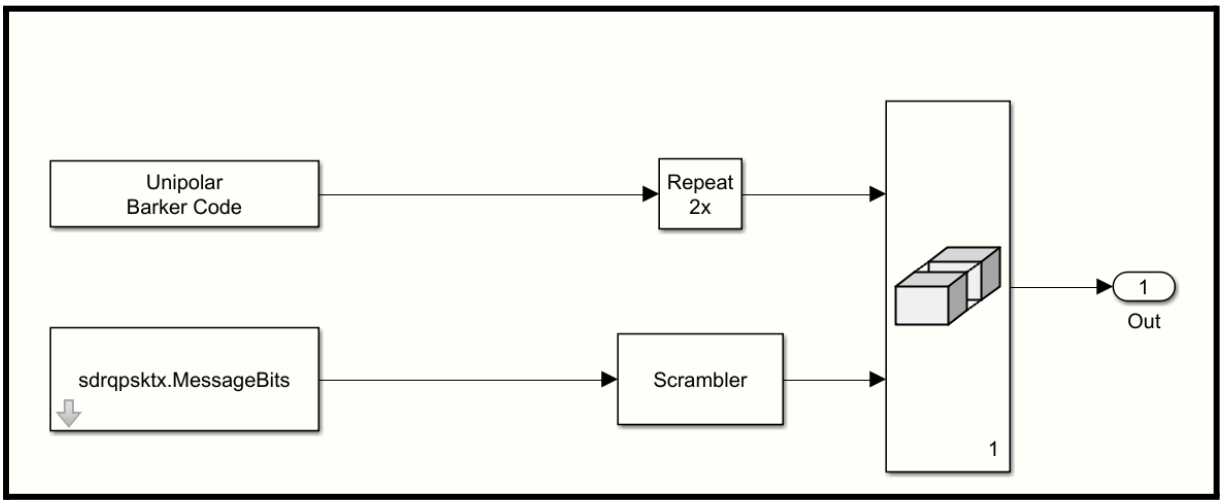
Enfin, la durée de la simulation est définie par le paramètre StopTime, fixé à 1000 secondes. Ce temps de simulation permet de transmettre un grand nombre de trames, ce qui est utile pour analyser les performances globales du système, notamment la stabilité de la transmission et la robustesse face aux erreurs. La durée de chaque trame est calculée en fonction de la longueur des trames et de la fréquence d'échantillonnage du Pluto.

B - Le générateur de bits

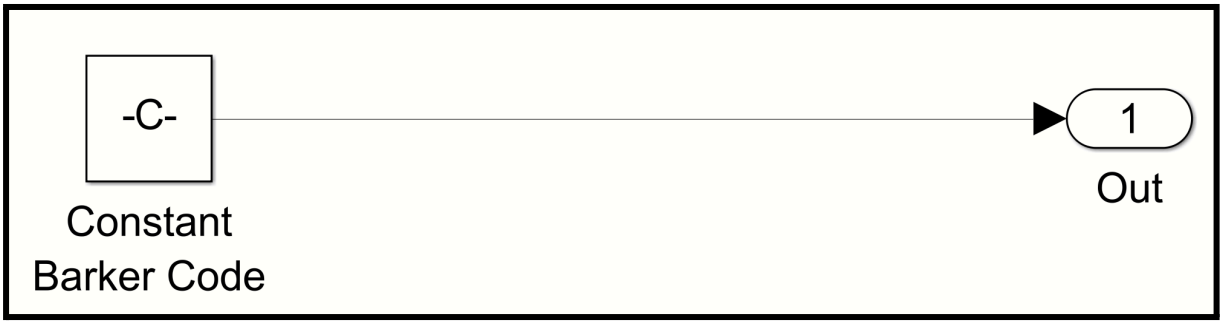
Le premier bloc majeur de notre transmetteur est le générateur de bits :



Et lui même analysé plus en détail ressemble à ceci :

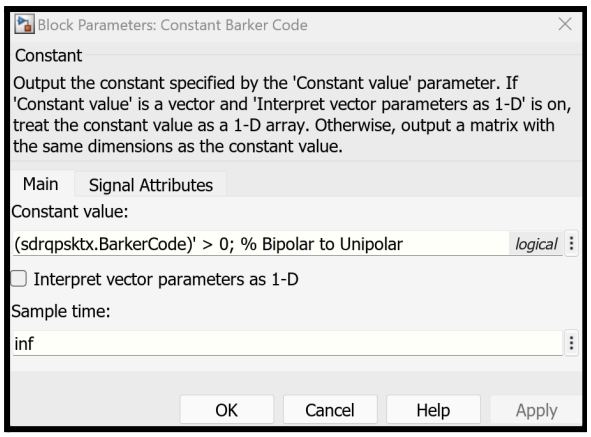


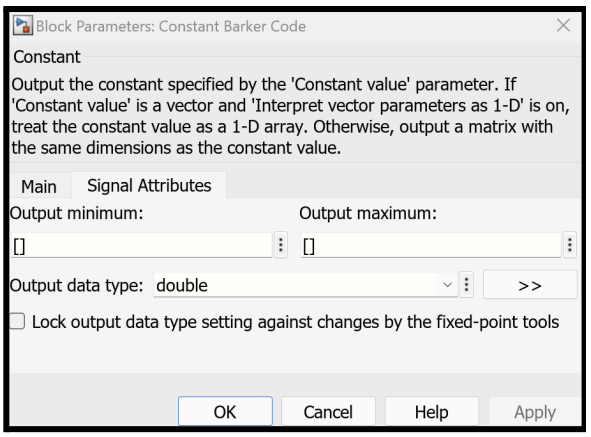
Il est donc composé de six blocs mineurs que nous allons l'un après l'autre en commençant par le Barker Code Unipolaire lui même composé de ceux deux blocs :



Tout d'abord, expliquons ce qu'est un Barker Code : C'est une séquence finie de valeurs digitales ayant le parfait taux d'autocorrélation, il est utilisé pour synchroniser le transmetteur et le receveur d'un flux binaire, sachant que le taux d'autocorrélation est la mesure de la similarité d'un signal avec lui-même après un certain décalage dans le temps, pour s'assurer le transmetteur et le receveur aient une synchronisation optimisée.

Voici les paramètres du bloc de la constante du Barker Code :

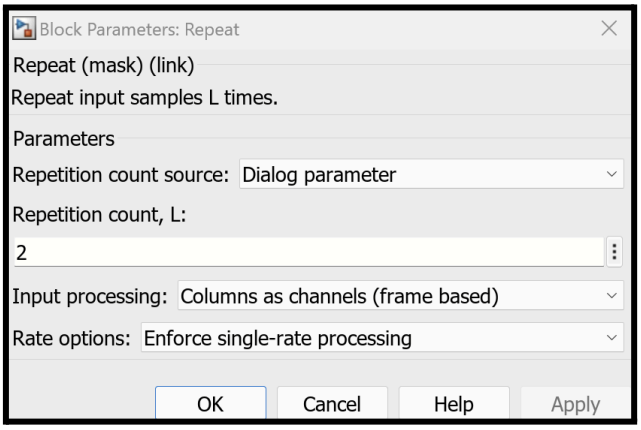




Cela va permettre de convertir le Barker Code bipolaire (car composé de "+1" et de "-1") en un Barker Code unipolaire (car composé de "+1" et de "0"). Le type de ces données sera en double.

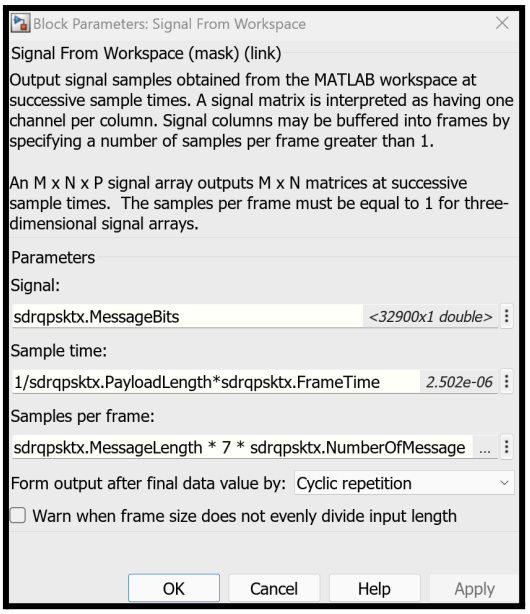
Et le petit bloc Out va permettre de récupérer ce Barker Code et de le mettre en entrée du bloc Unipolar Barker Code.

Le prochain bloc est un répéteur :



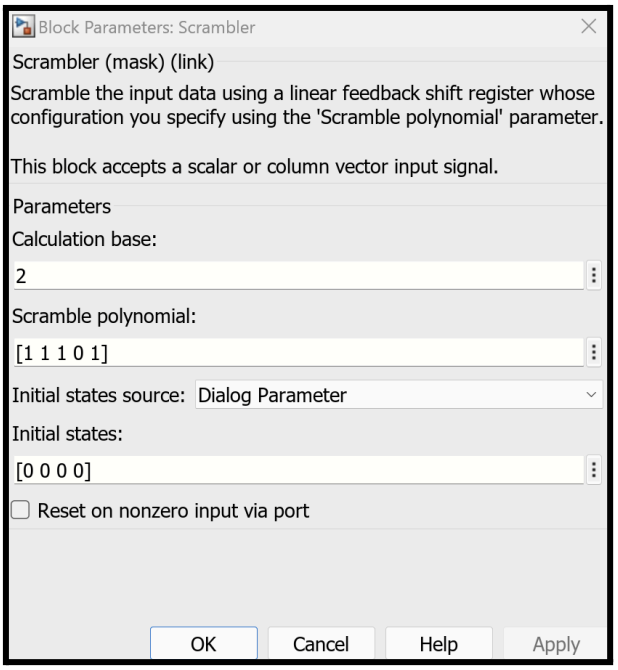
Pourquoi répéter deux fois le Barker Code ? Car cela permet de maintenir une fréquence d'échantillonnage constante à travers le système, en évitant les changements de taux entre les différentes parties du signal. Cette répétition peut améliorer la synchronisation du signal, réduire les erreurs de détection, et renforcer la robustesse face au bruit ou aux interférences, en offrant plus de redondance pour la détection correcte du signal dans des conditions difficiles.

La sortie de ce répéteur ira tout droit vers un bloc de concaténation de matrice à deux entrées dont nous parlerons plus tard, nous avons donc la première entrée mais qu'en est il de la seconde, en revenant à la base, il y a autre bloc sans prédécesseur qui est le suivant :



Il permet de prendre des échantillons à des instants successifs depuis l'espace de travail MATLAB et dans notre cas le fichier de configuration contenant les propriétés du message que l'on veut transmettre, les interprète selon leur organisation en colonnes, et peut les regrouper par trames pour un traitement plus efficace.

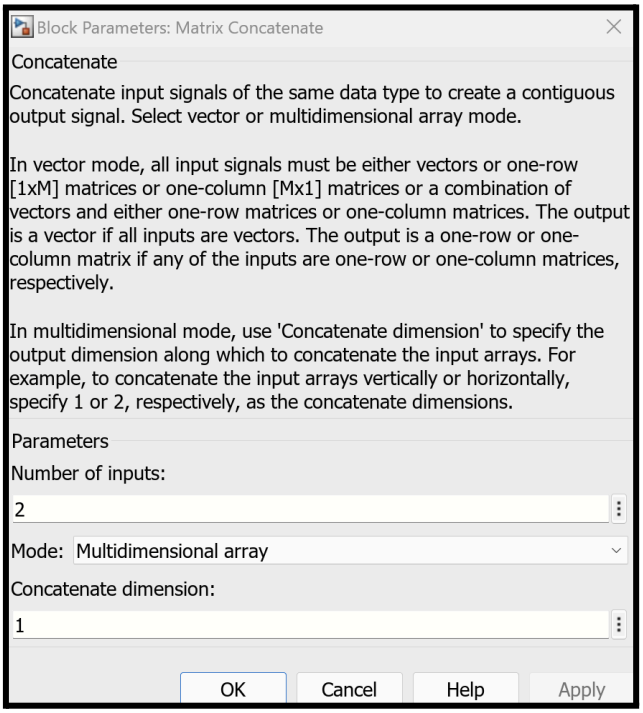
La sortie de ce bloc est dirigé vers un bloc Scrambler :



Il va brouiller le signal, brisant ainsi les motifs répétitifs ce qui permet de le rendre plus aléatoire et ainsi moins exploitable par un attaquant. De plus, ces motifs peuvent créer des soucis de modulation et/ou de filtrage en étant trop concentrés sur la même fréquence, le scrambler va donc permettre de les "étaler" sur la bande passante disponible.

Et la sortie du Scrambler sera la deuxième entrée du bloc de concaténation de matrices évoqué précédemment.

Analysons donc ce bloc plus en détails :



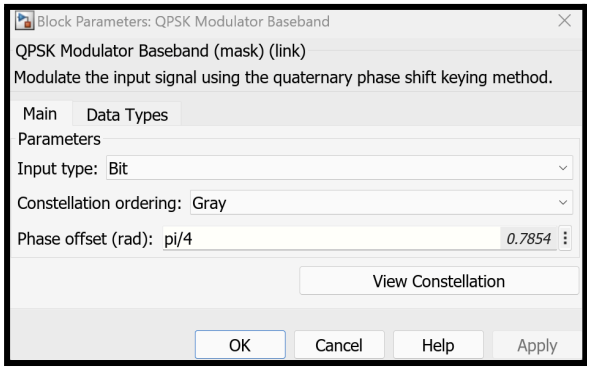
Nous voulons transmettre nos données en un seul signal, pourtant, nous avons déjà deux signaux ayant des dimensions mathématiques différentes, le premier étant une matrice de taille [13x1] répétée une seconde fois pour en obtenir une de taille [26x1], et l'autre étant de taille [32900x1], cette différences de dimension empêche la transmission en un seul signal, c'est pour cela que nous utilisons une concaténation de matrices afin d'assembler les deux signaux en un seul de dimension [32926x1].

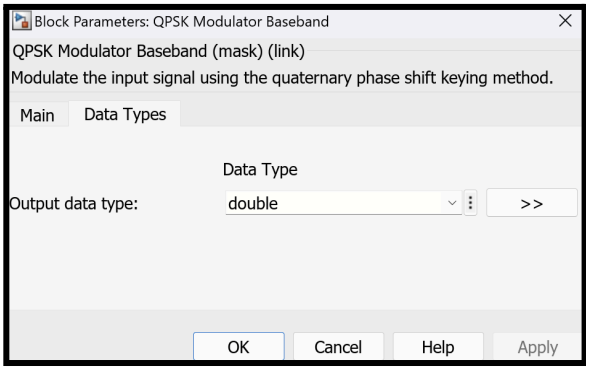
Et le dernier petit bloc Out permettra de faire de la sortie du bloc de concaténation de bits la sortie du premier bloc majeur qui était le générateur de bits.

C - Le modulateur QPSK

Le prochain bloc majeur est donc le modulateur QPSK (Quadrature Phase Shift Keying ou Modulation en Quadrature de Phase) en mode Baseband, un dispositif qui convertit un flux de bits numériques en un signal analogique modulé en phase, en utilisant quatre états de phase distincts (0°, 90°, 180° et 270° qui sont donc 0, π/2, π et 3π/2) pour représenter deux bits par symbole puisque c'est la valeur que l'on a choisi. En mode baseband, le signal modulé est transmis à des fréquences proches de zéro, ce qui signifie que le signal est centré autour de la fréquence de base, sans porteuse élevée. Chaque paire de bits est mappée à l'un des quatre symboles de phase, ce qui permet une utilisation plus efficace de la bande passante par rapport à d'autres modulations comme BPSK (Binary Phase Shift Keying ou encore Modulation de Phase Binaire). Le modulateur génère deux signaux porteurs, l'un modulé en phase (I, in-phase) et l'autre en quadrature (Q, quadrature), et les combine pour former un signal composite qui est prêt à être transmis sur le canal.

Voici les paramètres en question :



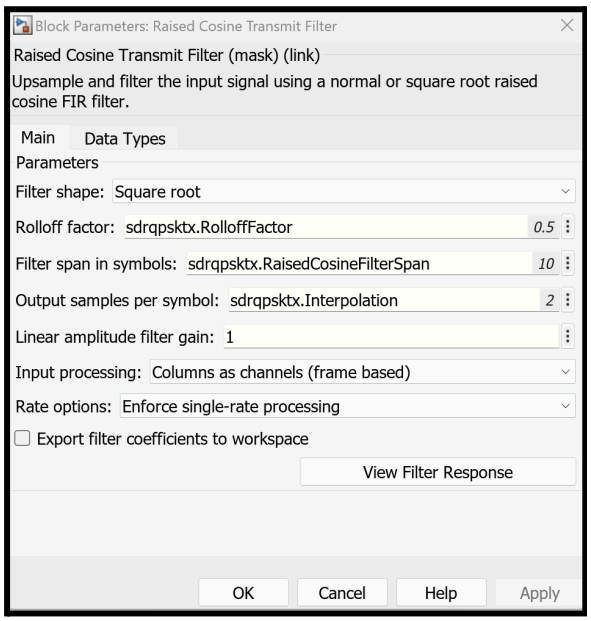


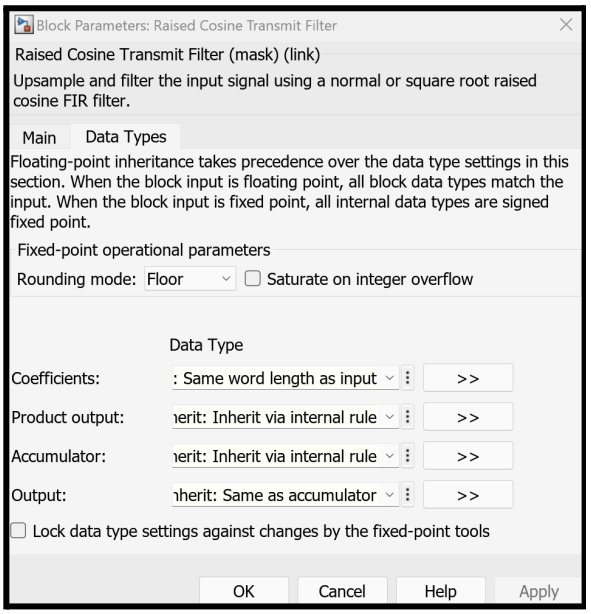
Il y a donc une entrée binaire arrangée en constellation de Gray ce qui signifie que l'ordre des éléments de deux bits n'est pas 00, 01, 10 et 11 qui est l'ordre croissant mais 00, 01, 11 et 10, un ordre dans lequel pour passer d'un élément à l'autre, il ne faut changer qu'un bit (à l'opposé de l'ordre croissant avec 01 et 10 par exemple). Ainsi, si une erreur se produit dans le signal reçu, elle est plus probablement limitée à une seule erreur de bit, ce qui améliore la robustesse de la transmission.

Et une chose qui a été dite avant n'est pas tout à fait exacte, les quatres états de phase ne sont pas 0°, 90°, 180° et 270° qui sont donc 0, π/2, π et 3π/2 mais bien 45°, 135°, 225° et 315° qui sont donc π/4, 3π/4, 5π/4 et 7π/4 car nous définissons un offset valant π/4. Un détail qui n'en est pas un est que la sortie des données sera de type double ce qui divise par deux la taille de la matrice du signal qui vaudra donc [16463x1] contrairement à celle d'entrée étant [32926x1].

D - Le filtre d'émission

Le prochain bloc majeur est le filtre en cosinus surélevé d'émission possédant les paramètres suivants :





Expliquons donc le fonctionnement du filtre :

Le filtre en racine de cosinus surélevé (Square Root Raised Cosine que l'on appellera SRRC) sert à façonner les impulsions que nous avons utilisées dans notre système de communication numérique pour minimiser l'interférence inter symboles et ainsi permettre d'optimiser l'efficacité spectrale.

Le filtre SRRC est la racine carrée d'un filtre en cosinus surélevé classique. Lorsqu'il est utilisé à la fois à l'émission et à la réception (technique de matched filtering), la combinaison des deux SRRC (transmetteur + récepteur) donne un filtre en cosinus surélevé complet. Cela permet de répartir l'effort de réduction de l'interférence inter symbole entre les deux extrémités de la chaîne de communication, tout en maintenant une complexité de traitement réduite.

Le facteur de roll-off détermine la largeur de la transition entre la bande passante utile et les fréquences hors bande. Un facteur de 0,5 signifie que la bande passante totale du signal sera 1,5 fois la fréquence de Nyquist. Ce compromis offre une bonne balance entre l'efficacité spectrale (réduction de la largeur de bande) et la robustesse face à l'interférence inter symbole. Plus le facteur est faible, plus la transition est abrupte, mais cela complique la mise en œuvre du filtre.

Le filter span in symbols détermine la longueur totale du filtre en termes de symboles. Ici, le filtre s'étend sur 10 symboles, ce qui permet d'assurer une bonne précision dans la forme des impulsions et une atténuation efficace des lobes secondaires. Plus le span est grand, plus le filtre est efficace pour réduire l'interférence inter symbole, mais cela augmente également la complexité computationnelle.

Le paramètre output samples per symbol = 2 signifie que chaque symbole modulé sera représenté par deux échantillons. Cela est souvent utilisé pour faciliter le suréchantillonnage, le filtrage numérique, et le traitement du signal, en particulier dans les systèmes SDR. Un suréchantillonnage aide également à simplifier le filtrage analogique en aval.

Le linear amplitude filter gain fixé à 1 indique que le filtre ne modifie pas l'amplitude globale du signal. Cela garantit que le signal reste à son niveau d'énergie initial après le filtrage, évitant ainsi toute distorsion de l'échelle d'amplitude.

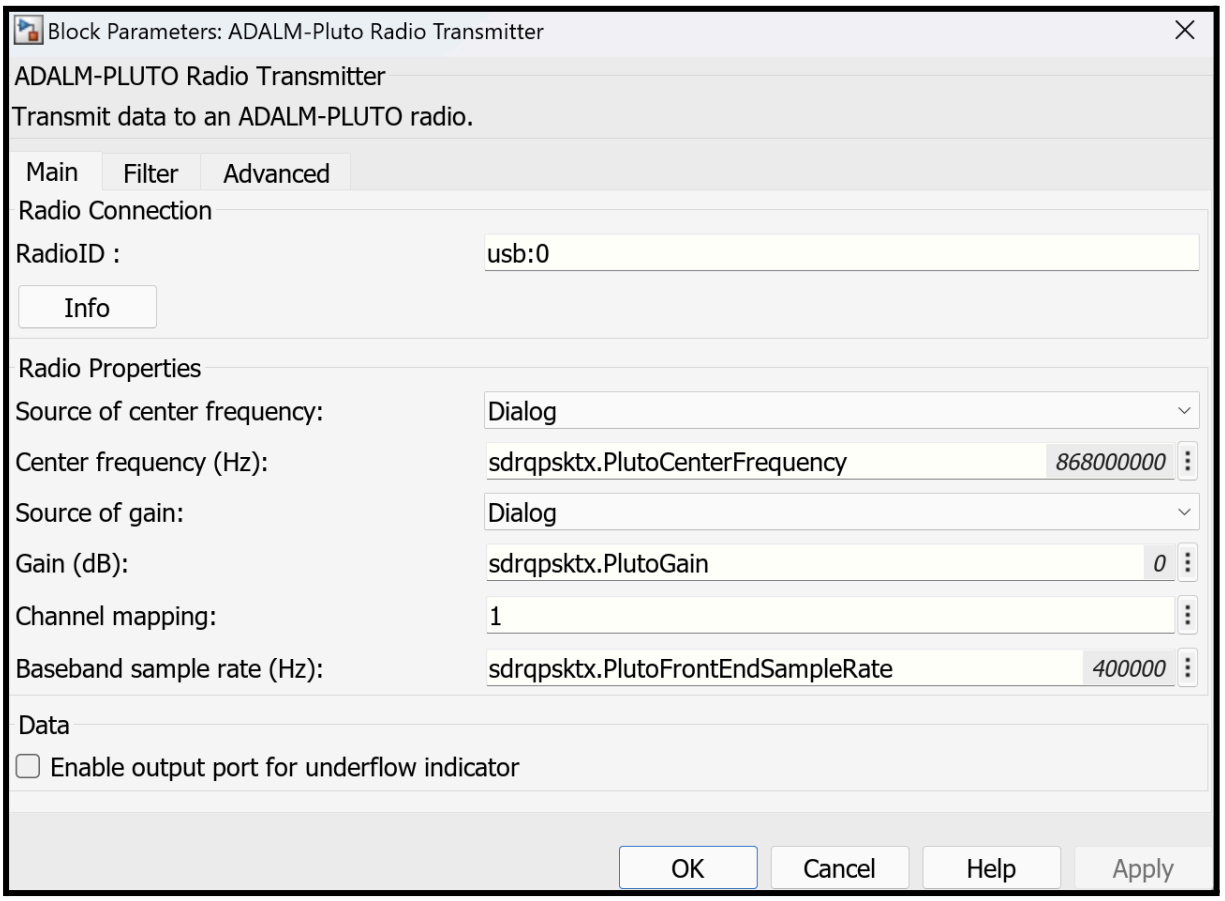
Le paramètre input processing : columns as channels signifie que chaque colonne de la matrice d'entrée est traitée comme un canal de données distinct. Cela est particulièrement utile pour le traitement de signaux multicanaux où plusieurs flux de données sont traités en parallèle.

Avec l'option enforce single-rate processing, le filtre est configuré pour maintenir le même taux d'échantillonnage à l'entrée et à la sortie. Cela est utile dans des systèmes où la cohérence du débit d'échantillonnage est cruciale pour l'alignement temporel ou la compatibilité avec d'autres blocs de traitement du signal.

Le paramètre data type : floor indique que les valeurs des données filtrées sont arrondies à l'entier inférieur. C'est pour des raisons de simplification numérique, et nous avons donc un filtre optimisé pour notre transmission…

E - Le transmetteur ADALM Pluto SDR

Nous pouvons donc analyser le dernier bloc majeur du transmetteur qui est logiquement le transmetteur radio ADALM Pluto :



Nous décidons donc de définir manuellement plutôt que dynamiquement la fréquence centrale et le gain de l'ADALM Pluto (en définissant la source de ces paramètres comme provenant du dialogue).

Et pour rappel, nous avions défini comme variables dans le fichier de configuration notre fréquence centrale étant égale à 868MHz et notre fréquence d'échantillonnage étant égale à 400kHz.

Enfin, le paramètre channel mapping nous permet de choisir le canal RF que nous allons utiliser et nous décidons de choisir le premier canal disponible sans compliquer la tâche plus que cela…

3 - Analyse du modèle du receveur

A - Le fichier de configuration

Voici donc le second fichier de configuration, celui du receveur :

```
function SimParams = plutoradioqpskreceiver_init
%% General simulation parameters
SimParams.Rsym = 0.2e6;         % Symbol rate in Hertz
SimParams.ModulationOrder = 4;     % QPSK alphabet size
SimParams.Interpolation = 2;      % Interpolation factor
SimParams.Decimation = 1;        % Decimation factor
SimParams.Tsym = 1/SimParams.Rsym;  % Symbol time in sec
SimParams.Fs   = SimParams.Rsym * SimParams.Interpolation; % Sample rate
%% Frame Specifications
SimParams.BarkerCode     = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];    % Bipolar Barker Code
SimParams.BarkerLength    = length(SimParams.BarkerCode);
SimParams.HeaderLength    = SimParams.BarkerLength * 2;             % Duplicate 2 Barker codes to be as a header
SimParams.Message       = 'Je suis un etudiant de RT-2025 Bilel Rohan';
SimParams.MessageLength   = length(SimParams.Message) + 5;
SimParams.NumberOfMessage = 100;                          % Number of messages in a frame
SimParams.PayloadLength   = SimParams.NumberOfMessage * SimParams.MessageLength * 7; % 7 bits per characters
SimParams.FrameSize      = (SimParams.HeaderLength + SimParams.PayloadLength) ...
  / log2(SimParams.ModulationOrder);                      % Frame size in symbols
SimParams.FrameTime      = SimParams.Tsym*SimParams.FrameSize;
%% Rx parameters
SimParams.RolloffFactor    = 0.5;              % Rolloff Factor of Raised Cosine Filter
SimParams.ScramblerBase    = 2;
SimParams.ScramblerPolynomial        = [1 1 1 0 1];
SimParams.ScramblerInitialConditions   = [0 0 0 0];
SimParams.RaisedCosineFilterSpan = 10;            % Filter span of Raised Cosine Tx Rx filters (in symbols)
SimParams.DesiredPower            = 2;        % AGC desired output power (in watts)
SimParams.AveragingLength          = 50;       % AGC averaging length
SimParams.MaxPowerGain             = 60;       % AGC maximum output power gain
SimParams.MaximumFrequencyOffset      = 6e3;
% Look into model for details for details of PLL parameter choice.
% Refer equation 7.30 of "Digital Communications - A Discrete-Time Approach" by Michael Rice.
K = 1;
A = 1/sqrt(2);
SimParams.PhaseRecoveryLoopBandwidth   = 0.01;      % Normalized loop bandwidth for fine frequency compensation
SimParams.PhaseRecoveryDampingFactor   = 1;        % Damping Factor for fine frequency compensation
SimParams.TimingRecoveryLoopBandwidth  = 0.01;      % Normalized loop bandwidth for timing recovery
SimParams.TimingRecoveryDampingFactor  = 1;        % Damping Factor for timing recovery
% K_p for Timing Recovery PLL, determined by 2KA^2*2.7 (for binary PAM),
% QPSK could be treated as two individual binary PAM,
% 2.7 is for raised cosine filter with roll-off factor 0.5
SimParams.TimingErrorDetectorGain     = 2.7*2*K*A^2+2.7*2*K*A^2;
SimParams.PreambleDetectionThreshold   = 0.8;
%% Message generation and BER calculation parameters
msgSet = zeros(100 * SimParams.MessageLength, 1);
for msgCnt = 0 : 99
```

```
  msgSet(msgCnt * SimParams.MessageLength + (1 : SimParams.MessageLength)) = ...
    sprintf('%s %03d\n', SimParams.Message, msgCnt);
end
bits = de2bi(msgSet, 7, 'left-msb')';
SimParams.MessageBits = bits(:);
% For BER calculation masks
SimParams.BerMask = zeros(SimParams.NumberOfMessage * length(SimParams.Message) * 7, 1);
for i = 1 : SimParams.NumberOfMessage
  SimParams.BerMask( (i-1) * length(SimParams.Message) * 7 + ( 1: length(SimParams.Message) * 7) ) = ...
    (i-1) * SimParams.MessageLength * 7 + (1: length(SimParams.Message) * 7);
end
% Pluto receiver parameters
SimParams.PlutoCenterFrequency    = 868e6;
SimParams.PlutoGain               = 30;
SimParams.PlutoFrontEndSampleRate  = SimParams.Fs;
SimParams.PlutoFrameLength        = SimParams.Interpolation * SimParams.FrameSize;
% Experiment parameters
SimParams.PlutoFrameTime = SimParams.PlutoFrameLength / SimParams.PlutoFrontEndSampleRate;
SimParams.StopTime = 10;
```

Analysons donc ce second fichier.

Il est possible de le séparer en les 6 sections distinctes suivante :

- La définition des paramètres généraux de la simulation

Cette section définit les paramètres de base pour la simulation du système de communication. Elle inclut le taux de symboles (Rsym), qui détermine la vitesse de transmission des données, ainsi que l'ordre de modulation (ModulationOrder), fixé à 4 pour une modulation QPSK (Quadrature Phase Shift Keying). Des facteurs d'interpolation et de décimation sont utilisés pour ajuster la fréquence d'échantillonnage (Fs), essentielle pour garantir une compatibilité entre les signaux numériques et les capacités matérielles de l'ADALM Pluto SDR.

- La spécification de la trame

Ici sont définis les paramètres relatifs à la structure des trames de données. Un code de synchronisation Barker (BarkerCode) est utilisé pour faciliter la détection des débuts de trame. Chaque trame contient un en-tête formé par deux séquences Barker ainsi qu'un ensemble de messages de type "Je suis un etudiant de RT-2025 Bilel Rohan 000\n" à "Je suis un etudiant de RT-2025 Bilel Rohan 099\n". La taille totale de la trame (FrameSize) dépend du nombre de messages, de la longueur des données et du schéma de modulation choisi.

- La définition des paramètres du receveur

Cette partie configure les paramètres de réception pour assurer la qualité du signal. Des filtres en cosinus surélevé sont utilisés pour réduire l'interférence inter-symboles grâce à un facteur de roll-off (RolloffFactor) de 0,5. Le scrambler empêche l'apparition de motifs répétitifs dans les données, facilitant ainsi la synchronisation. L'AGC (Automatic Gain Control) ajuste automatiquement le gain pour stabiliser la puissance du signal reçu. Enfin, des boucles à verrouillage de phase (PLL) sont paramétrées pour corriger les décalages de fréquence et de phase.

- La génération des messages et le calcul du Taux d'Erreur Binaire

Cette section génère les messages à transmettre et prépare le système pour l'évaluation des performances. Cent messages distincts sont créés et encodés en binaire (7 bits par caractère). Un masque de bits (BerMask) est utilisé pour comparer les bits transmis et reçus, permettant ainsi de calculer le taux d'erreur binaire (BER), un indicateur clé de la fiabilité du système de communication.

- La définition des paramètres spécifiques à l'ADALM Pluto SDR

Cette partie configure les paramètres propres au matériel SDR (Software Defined Radio) ADALM Pluto. Elle définit la fréquence centrale de réception (PlutoCenterFrequency) à 868 MHz, une bande couramment utilisée pour des applications sans fil. Le gain du récepteur (PlutoGain) est fixé à 30 dB pour optimiser la réception du signal. La fréquence d'échantillonnage à l'entrée du Pluto est synchronisée avec celle de la simulation (PlutoFrontEndSampleRate), et la longueur des trames est ajustée en conséquence.

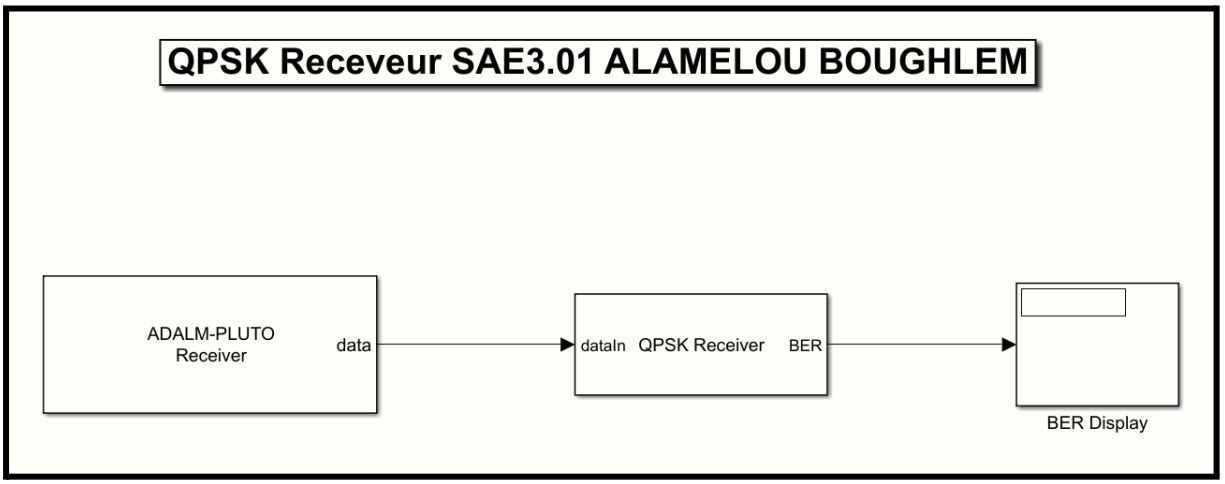
- Les paramètres de simulation

Enfin, la simulation est configurée pour s'exécuter pendant une durée spécifique définie par le paramètre StopTime, ici fixé à 10 secondes. Cette durée est suffisante pour permettre la transmission et la réception de plusieurs trames, ce qui facilite l'analyse des performances du système en termes de qualité de signal et de fiabilité des données.

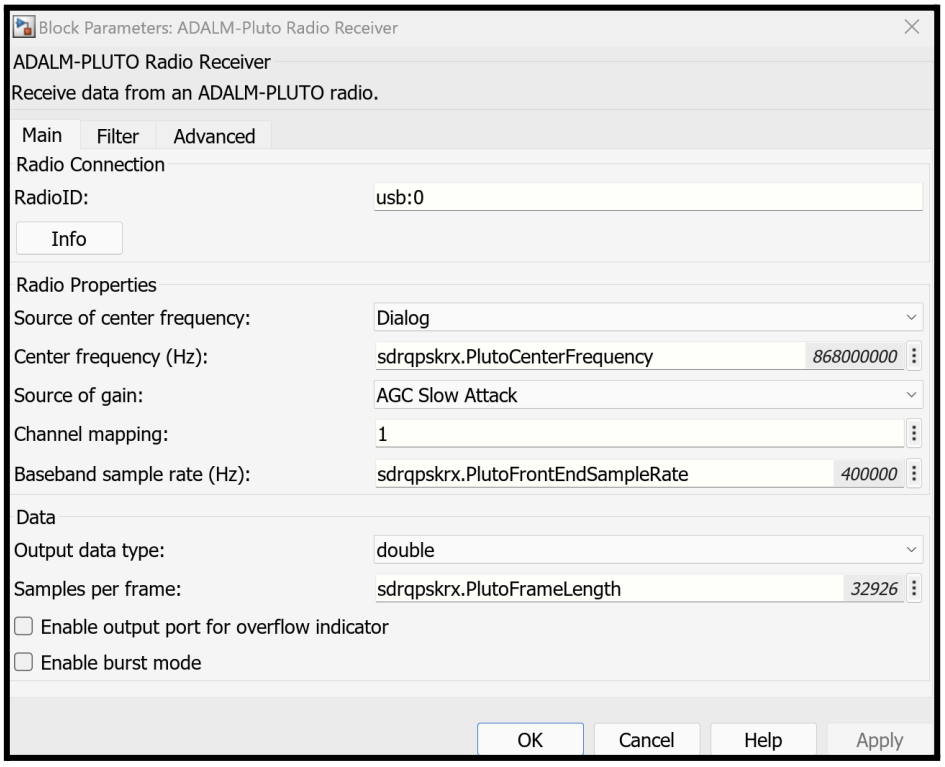
A noter que cela peut sembler étrange que le receveur connaisse déjà le message mais c'est nécessaire pour le calcul du Taux d'Erreur Binaire pour tester le fonctionnement de notre système de transmission.

B - Le receveur ADALM Pluto SDR

Le premier bloc majeur de notre receveur est le receveur ADALM Pluto :



Voici donc les paramètres du receveur ADALM Pluto :



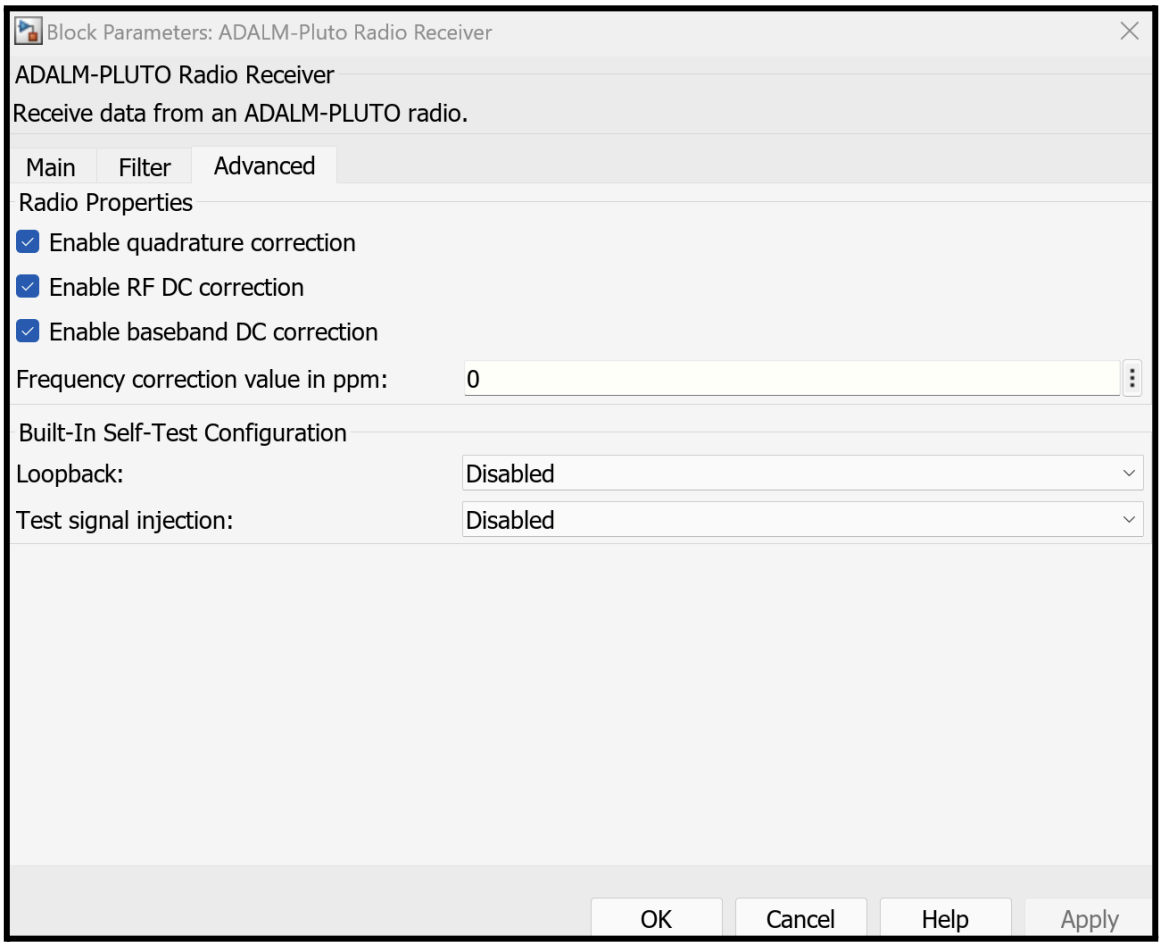
Quasiment tous les paramètres sont les mêmes donc les explications se trouvent dans cette partie du document.

Les deux paramètres qui changent (ou sont nouveaux) sont les suivants :

La source du gain ne sera plus configurée par le dialogue et donc manuellement mais par attaque lente AGC, AGC pour Automatic Gain Control, signifiant que l'ADALM Pluto ajuste automatiquement le gain de réception pour maintenir un signal constant, mais il le fait progressivement afin d'assurer une grande stabilité et de réduire les risques de distorsion, particulièrement utile dans des environnements de transmission relativement stables.

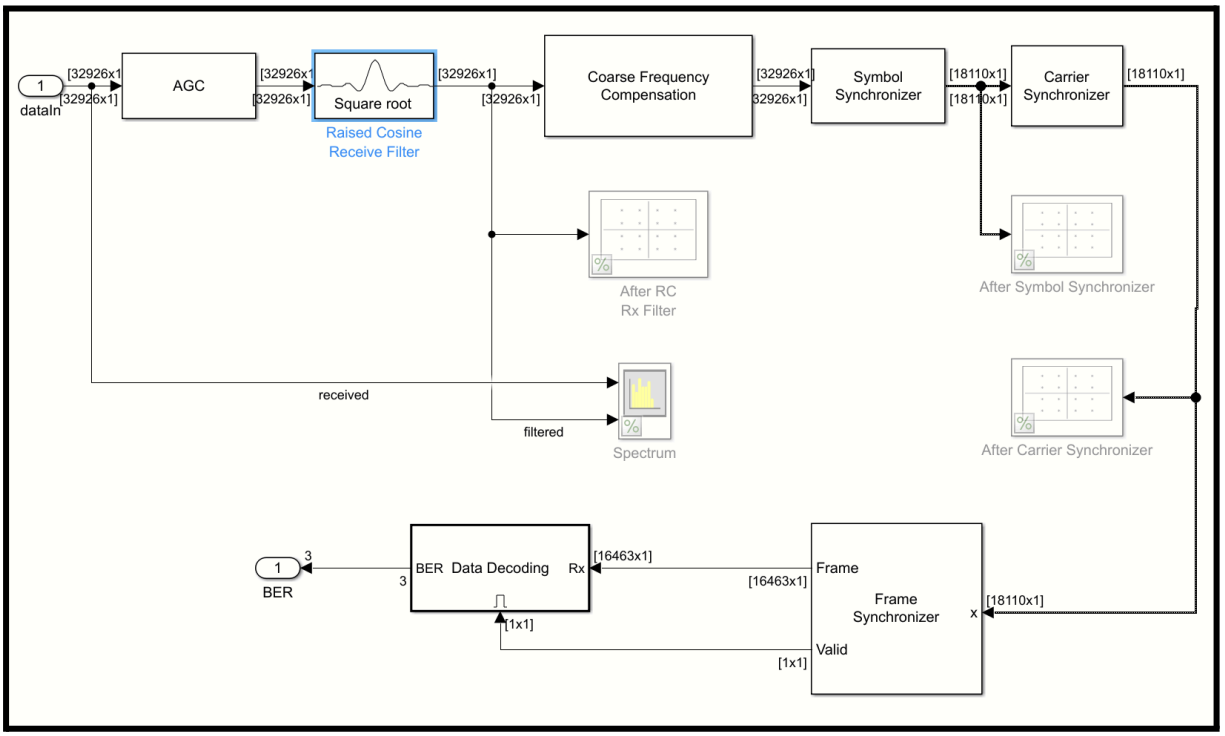
Le nombre d'échantillons par trame en sortie sera égal 32926, une valeur rappelant la taille de matrice [32926x1] des données transmises ce qui est logique.

De plus, nous activons toutes les corrections de nos données possibles :



C - Le démodulateur QPSK

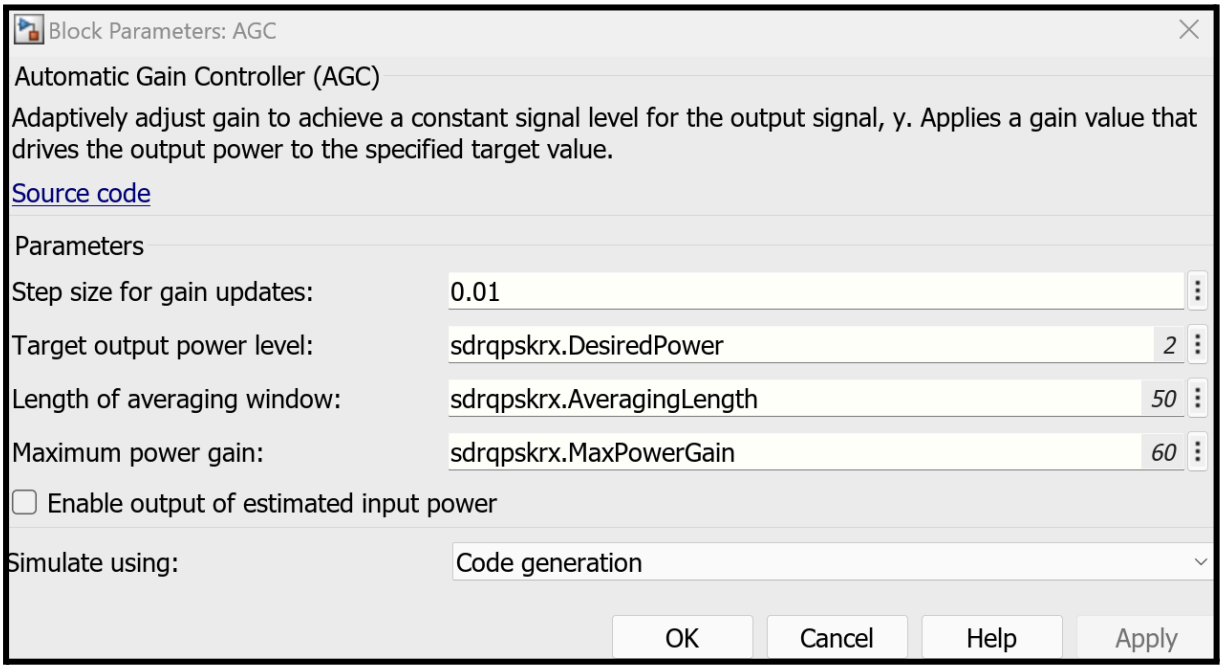
Le prochain bloc majeur est le démodulateur QPSK, composé des blocs mineurs suivants :



Le bloc dataIn va donc prendre comme entrée la sortie du bloc de l'ADALM Pluto qui fera office d'entrée pour le prochain bloc mineur qui est le contrôleur de gain automatique (AGC pour Automatic Gain Control).

Ce bloc sert à ajuster automatiquement le gain du signal d'entrée pour maintenir un niveau de puissance de sortie constant. C'est une anticipation aux différentes fluctuations de la puissance du signal reçu dues aux différentes atténuations, interférences…

Il est configuré comme ceci :



Le premier paramètre définit la vitesse d'ajustement du gain par l'AGC à chaque itération. Une valeur de 0,01 signifie que l'AGC modifie le gain très progressivement pour éviter des variations trop brusques du signal. Cela permet une correction douce des fluctuations de puissance sans introduire d'instabilité ni de distorsion excessive. Une valeur trop élevée rendrait l'AGC plus réactif, mais au risque d'oscillations et de bruit supplémentaire sur le signal.

Le second paramètre définit le niveau de puissance cible pour la sortie du bloc AGC. Autrement dit, le gain sera ajusté pour que la puissance moyenne du signal de sortie soit maintenue autour de 2 Watts. Cela garantit un signal stable pour les étapes suivantes du traitement, comme la démodulation. Ce réglage est essentiel pour éviter que le signal soit trop faible (risque de perte d'information) ou trop fort (risque de saturation).

Le troisième paramètre contrôle la fenêtre d'observation sur laquelle l'AGC estime la puissance du signal avant d'ajuster le gain. Avec une longueur de 50 échantillons, l'AGC lisse les variations sur un période plus longue, ce qui réduit les fluctuations rapides et améliore la stabilité du signal. Une valeur trop faible rendrait l'AGC plus réactif mais pourrait entraîner des ajustements instables du gain, tandis qu'une valeur trop grande le rendrait trop lent à réagir aux changements soudains de la puissance du signal. Ce paramètre ressemble au premier…

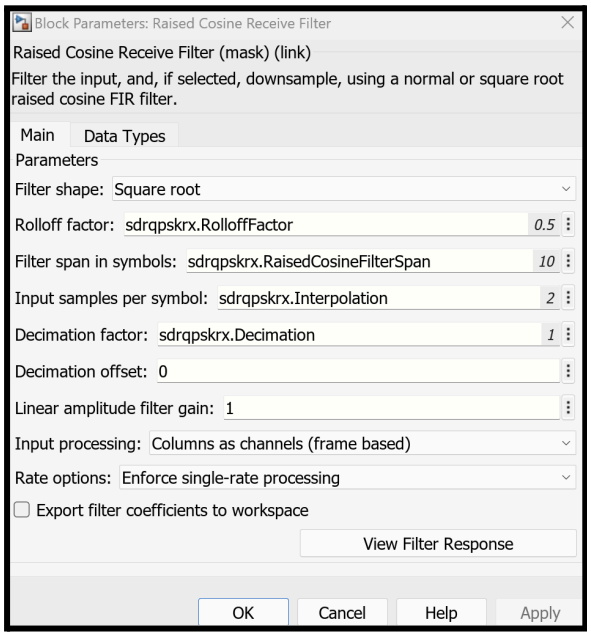
Le quatrième paramètre définit la limite maximale du gain que l'AGC peut appliquer au signal. Une valeur de 60 dB signifie que l'AGC ne peut pas amplifier le signal de plus de 10**6 fois sa puissance d'origine. Cela empêche l'AGC de compenser des signaux trop faibles en les amplifiant excessivement, ce qui pourrait aussi augmenter le bruit et détériorer la qualité du signal reçu.

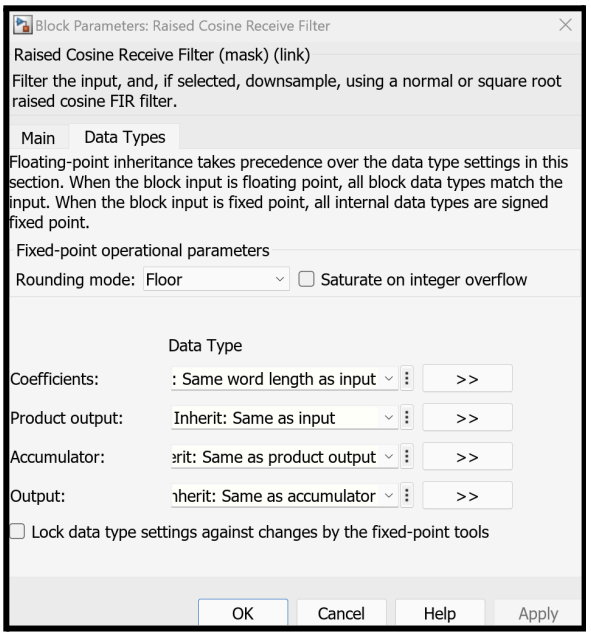
Le cinquième et sixième paramètres sont plus propres à Matlab qu'à un AGC physique :

- Avec le cinquième déterminant si l'AGC fournit ou non une mesure de la puissance du signal d'entrée en sortie. En le réglant sur "No", l'AGC ajuste le gain mais ne fournit pas d'informations sur la puissance mesurée du signal reçu. Activer cette option pourrait être utile pour du débogage ou pour analyser les variations du signal, mais elle n'est pas nécessaire pour une transmission standard.

- Et le sixième spécifiant le mode d'exécution de la simulation dans Simulink. En choisissant "Code Generation", l'AGC est simulé à l'aide de code compilé plutôt que d'une interprétation en temps réel. Cela améliore la vitesse d'exécution, ce qui est particulièrement utile pour des simulations (relativement) complexes comme la nôtre.

Le prochain bloc mineur est un filtre en cosinus surélevé de réception, et puisque déjà abordé et expliqué pour le transmetteur, nous n'expliquerons que les paramètres changés :



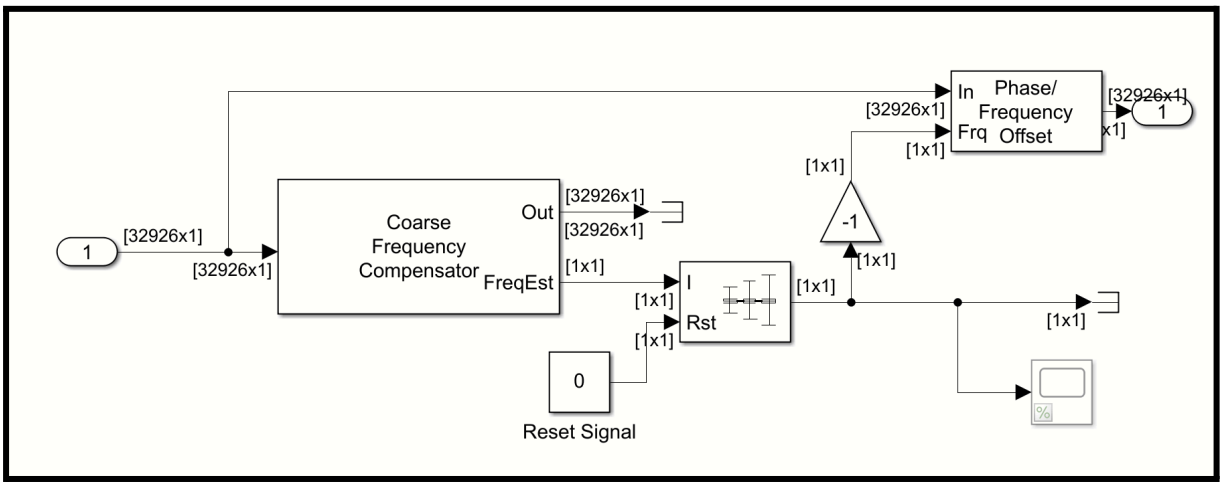


Si tous les paramètres sont quasiment tous les mêmes, celui qui est différent Decimation Offset, un paramètre propre à notre filtre en cosinus surélevé de réception

Le Décimation Offset = 0 dans notre filtre en cosinus surélevé côté récepteur signifie que l'échantillonnage du signal reçu est aligné avec les instants optimaux de décision du récepteur, c'est-à-dire au centre des symboles transmis. Ce paramètre est crucial car, comme vu précédemment, un filtre en cosinus surélevé sert à limiter l'interférence entre symboles (ISI) en concentrant l'énergie du signal autour des instants optimaux. Une valeur de 0 indique que le récepteur ne décale pas l'instant d'échantillonnage, assurant une récupération efficace des symboles et minimisant les erreurs. Un décalage non nul pourrait être utilisé pour compenser une désynchronisation éventuelle, mais ici, cela signifie que la synchronisation est supposée parfaite dès la réception..

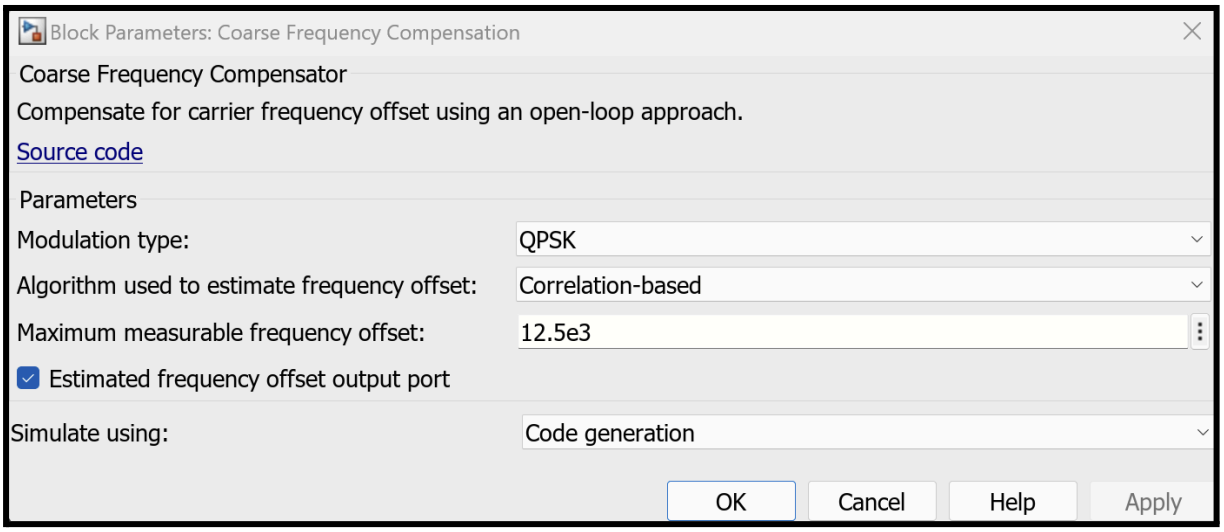
Le prochain bloc mineur est le Coarse Frequency Compensation (ou compensation grossière de fréquence) qui sert à corriger rapidement les erreurs de fréquence entre l'émetteur et le récepteur.

Il existe un bloc nommé Coarse Frequency Compensator dans la librairie de Simulink mais ce bloc ne peut pas être configuré de manière assez précise comparé au bloc que nous avons créé, composé de la multitude des plus petits blocs suivants :



Nous pouvons donc voir le bloc Coarse Frequency Compensation comme étant le bloc coarse frequency compensator mais avec des blocs permettant d'apporter des modifications plus précises au signal que nous allons analyser…

Il n'est plus nécessaire d'expliquer les blocs de ports d'entrée et de sortie. Analysons donc directement le bloc Coarse Frequency Compensator :



Le premier paramètre indique donc l'utilisation de notre bloc comme étant adapté à une QPSK ce qui est logique puisqu'étant le type de modulation que nous avons choisi pour notre transmission.

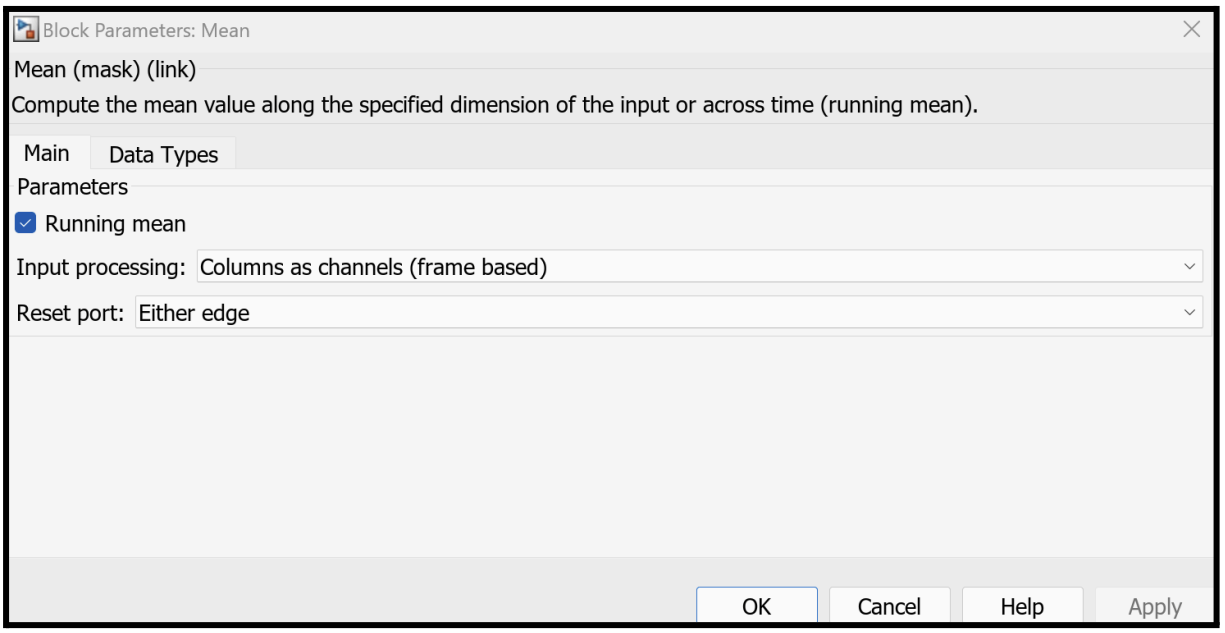
Le second paramètre indique que le décalage de fréquence est estimé à l'aide d'un algorithme basé sur la corrélation. Cet algorithme compare les motifs répétitifs du signal reçu (comme des séquences de formation ou des préambules connus) pour détecter la vitesse à laquelle la phase du signal dérive dans le temps. Il est particulièrement efficace pour des signaux modulés comme le QPSK, où la structure des symboles permet d'exploiter cette corrélation pour une estimation rapide du décalage de fréquence.

Le troisième paramètre définit la plage maximale de fréquence que le compensateur peut corriger. Ici, une erreur de fréquence jusqu'à ±12,5 kHz peut être détectée et corrigée. Cela signifie que si l'erreur dépasse cette valeur, le compensateur ne pourra pas l'estimer correctement, et une partie du décalage persistera, nécessitant des corrections supplémentaires en aval (comme une compensation fine via une boucle PLL).

Le quatrième paramètre que nous activons permet au bloc de fournir en sortie la valeur estimée du décalage de fréquence en plus du signal corrigé. Cela permet de visualiser et analyser la dérive de fréquence, ce qui est utile pour le diagnostic du système et pour un éventuel ajustement dynamique de la réception. Cette information peut être utilisée pour affiner d'autres parties du traitement du signal.

Enfin, le cinquième paramètre indique que la simulation est exécutée via du code compilé plutôt que via une interprétation en temps réel. C'est une configuration que nous avons également vu sur l'AGC.

L'estimation en fréquence sortant du bloc servira, avec une constante valant 0, à calculer une moyenne au moyen du bloc mean :

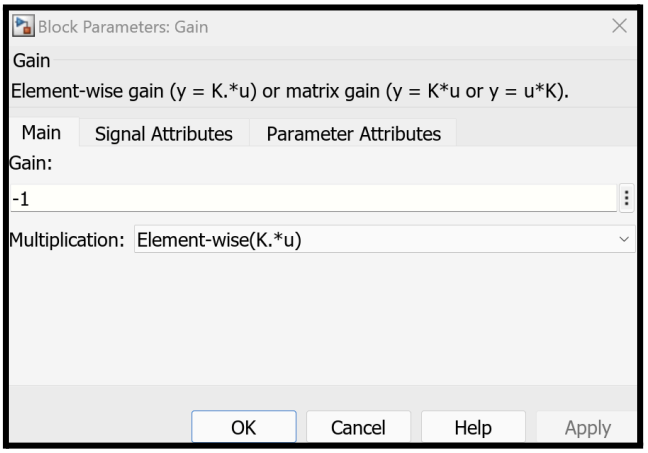


Le premier paramètre précise que le bloc effectue une moyenne glissante, ce qui signifie que la moyenne est recalculée dynamiquement à chaque nouvel échantillon reçu. Cela permet de lisser les variations de l'estimation de fréquence et de réduire les fluctuations dues au bruit.

Le second paramètre indique que les données sont traitées colonne par colonne, chaque colonne étant considérée comme un canal distinct. En mode Frame-Based, le bloc traite plusieurs échantillons en même temps, ce qui améliore l'efficacité du calcul et permet une mise à jour plus rapide de la moyenne.

Enfin, le troisième bloc précise que la moyenne peut être réinitialisée dès qu'un signal externe change d'état (passage de 0 à 1 ou de 1 à 0). Cela permet de redémarrer le calcul lorsqu'un événement important est détecté, comme un changement soudain du décalage de fréquence.

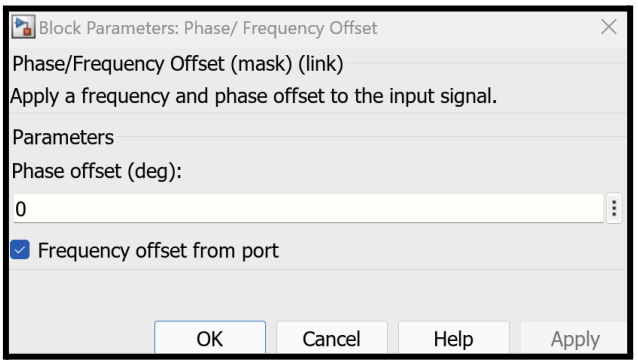
Cette moyenne subira une modification de type multiplicative par -1 dans le bloc suivant :



Cela inverse simplement le signe de la moyenne…

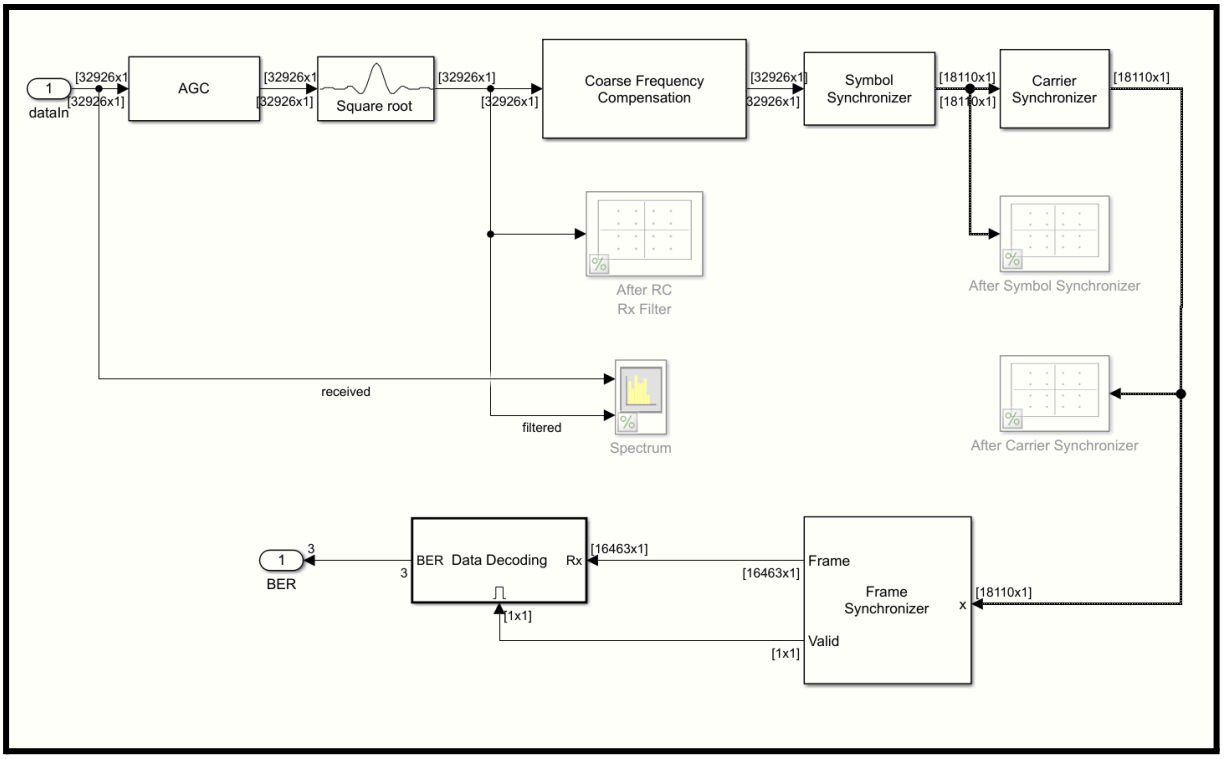
Et le signal (avant d'entrer dans le bloc Coarse Frequency Compensator) sera une entrée du bloc Phase/Frequency Offset, l'autre entrée étant la moyenne inversée de la fréquence estimée en sortie du Coarse Frequency Compensator.

Le bloc Phase/Frequency Offset applique une compensation de phase et de fréquence au signal d'entrée. En prenant comme second paramètre l'inverse de l'estimation de fréquence issue du Coarse Frequency Compensator, il annule le décalage de fréquence estimé. Concrètement, si le compensateur a détecté un décalage de fréquence Δf, alors le bloc applique une correction de -Δf, ramenant ainsi le signal à sa fréquence nominale. Cela permet d'aligner correctement le signal reçu pour une démodulation précise, en éliminant les erreurs de fréquence qui pourraient fausser l'interprétation des symboles.



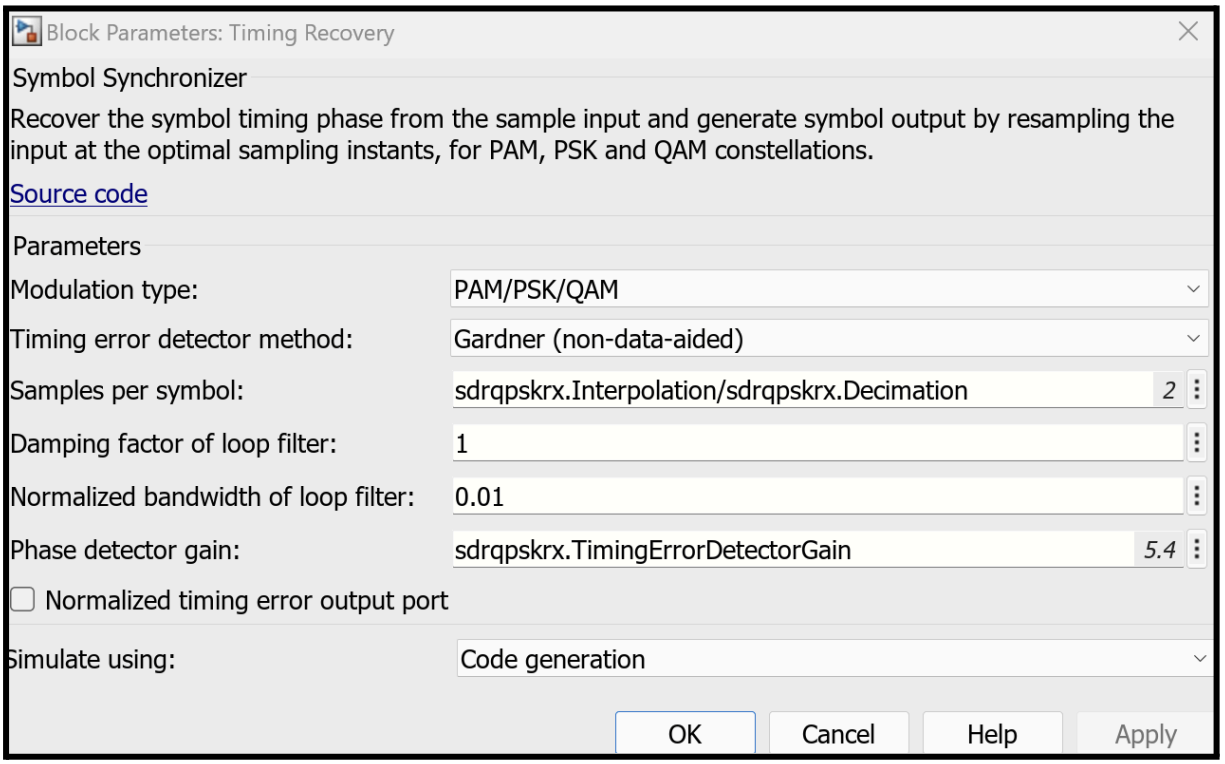
A noter que ces blocs sont des terminateurs de signaux permettant de le diriger vers une sortie nulle :

Et ce sera tout pour le bloc Coarse Frequency Compensation, qui pour rappel, était lui même composé de plus petits blocs, continuons donc avec le prochain bloc mineur étant le synchroniseur de symboles :



Un synchroniseur de symboles sert à aligner correctement les instants d'échantillonnage d'un signal reçu afin d'extraire les symboles de manière optimale. Dans un système de communication numérique, après la transmission, le signal peut subir des déphasages, des dérives de fréquence, ou des distorsions temporelles qui rendent difficile la récupération précise des symboles.

Et voici ses configurations :



Le premier paramètre nous permet de choisir le type de modulation du synchroniseur de symboles qui, bien sûr, est une PSK donc nous choisissons cette modulation.

Le second paramètre indique que le détecteur d'erreur de synchronisation est basé sur l'algorithme Gardner, qui est non-data-aided (NDA). Cela signifie qu'il ne nécessite pas de séquence de formation connue pour fonctionner. Il utilise la relation entre les échantillons milieu de symbole et les échantillons inter-symboles pour estimer et corriger le retard ou l'avance de l'horloge d'échantillonnage. Cet algorithme est particulièrement efficace pour des signaux en bande de base avec un taux d'échantillonnage élevé.

Le troisième paramètre indique que chaque symbole est représenté par 2 échantillons après l'échantillonnage. Un taux de 2 échantillons par symbole est un bon compromis entre précision et complexité, permettant au synchroniseur d'interpoler correctement les instants optimaux pour la prise d'échantillons sans ajouter trop de charge computationnelle.

Le quatrième paramètre est le facteur d'amortissement (damping factor) qui contrôle la réponse de la boucle PLL (Phase-Locked Loop) du synchroniseur. Une valeur de 0,01 signifie que le filtre est faiblement amorti, ce qui le rend très réactif aux erreurs de synchronisation mais peut aussi causer des oscillations. Un facteur plus élevé rendrait la réponse plus lisse mais plus lente à converger.

Le cinquième paramètre définit le gain du détecteur de phase, qui influence la vitesse de correction des erreurs de synchronisation. Une valeur de 5,4 signifie que les ajustements seront relativement rapides, permettant au système de s'adapter plus efficacement aux variations du signal, mais un gain trop élevé pourrait aussi introduire de l'instabilité.

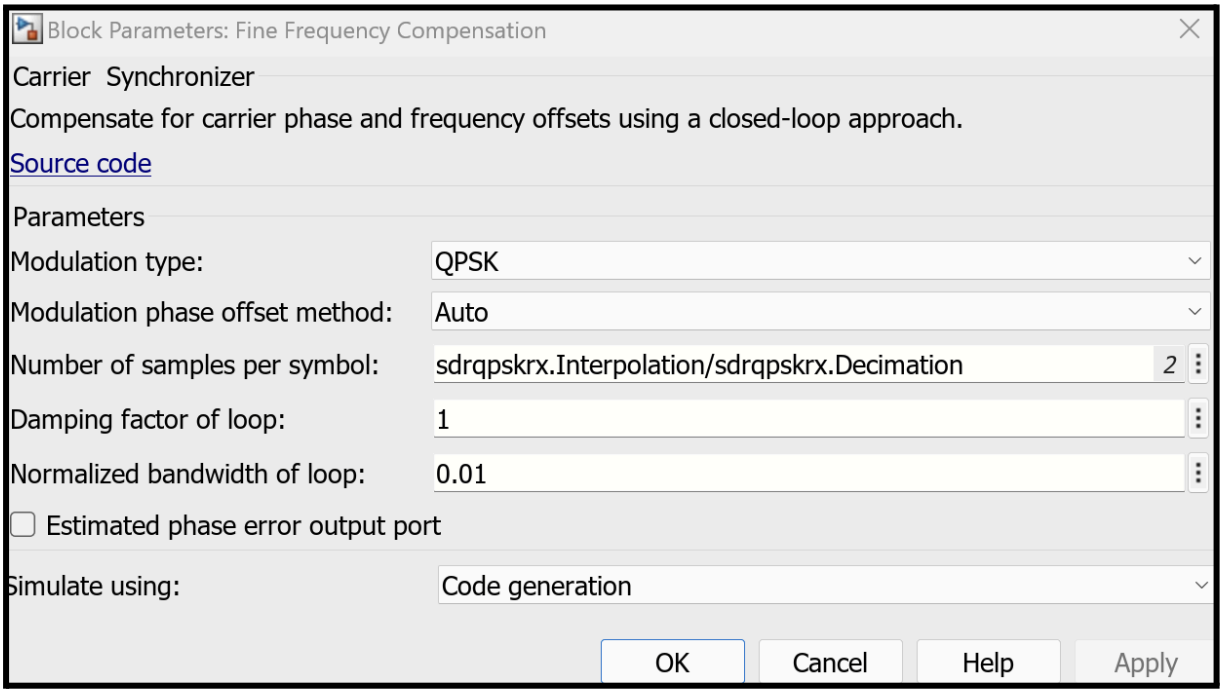
Le sixième paramètre est le choix de si oui ou non, on désactive la sortie du taux d'erreur de synchronisation normalisé. Nous choisissons de le désactiver ce qui signifie que le bloc ne fournit pas de mesure explicite de l'erreur de synchronisation, mais ajuste directement l'horloge d'échantillonnage en interne.

Enfin, le dernier paramètre indique que l'exécution du synchroniseur se fait via du code compilé au lieu d'une exécution interprétée en temps réel. Un paramètre déjà expliqué pour d'autres blocs…

Une dernière information importante sur ce bloc est que le signal subit une réduction de sa taille avec une dimensions passant de [32926x1] à [18110x1] après passage dans le synchroniseur de symboles qui s'explique par l'ajustement du taux d'échantillonnage effectué par le bloc. Initialement, le signal est suréchantillonné avec 2 échantillons par symbole, mais le synchroniseur corrige dynamiquement l'alignement des symboles et réajuste l'horloge d'échantillonnage, supprimant les échantillons redondants ou mal positionnés. Cette réduction de la longueur du signal correspond donc à une correction du timing et à une conversion vers un échantillonnage optimal, éliminant ainsi les variations temporelles et garantissant une récupération plus précise des symboles transmis.

Le prochain bloc mineur est un Carrier Synchronizer qui corrige le décalage de fréquence et de phase entre l'oscillateur local du récepteur et la porteuse du signal reçu. Il compense ces erreurs en ajustant dynamiquement la phase et la fréquence du signal, permettant une démodulation précise et évitant la rotation des symboles dans la constellation

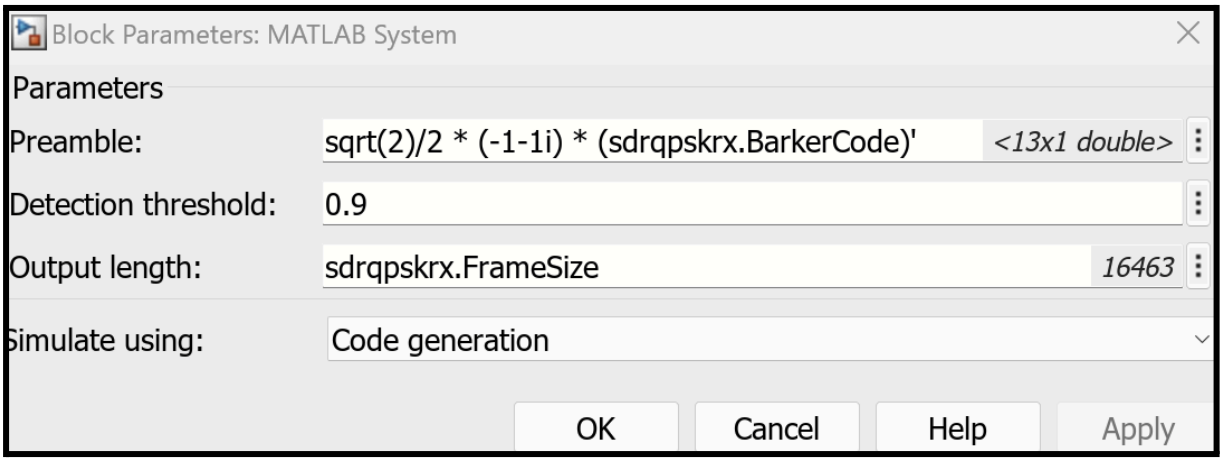
Voici donc ses paramètres :



Tous les paramètres sauf la méthode d'offset de phase de la modulation, sont les mêmes que le bloc précédent et sont donc expliqués plus haut, la méthode d'offset de phase de la modulation étant différente, expliquons la :

Elle signifie que le Carrier Synchronizer détermine automatiquement le décalage de phase initial de la modulation et l'ajuste pour assurer une synchronisation correcte. Dans le cas de modulations comme le QPSK, où la phase initiale peut varier en fonction de l'émetteur, cette option permet au synchroniseur de s'adapter sans nécessiter une configuration manuelle. Cela garantit que les symboles restent alignés avec la constellation correcte, évitant ainsi des rotations indésirables qui pourraient compromettre la démodulation des données

Le prochain bloc mineur est le synchroniseur de trames qui sert à détecter et aligner les trames de données dans notre signal. Dans une transmission numérique, chaque trame commence généralement par une préambule connu (comme un code de Barker) qui permet au récepteur d'identifier le début d'un message valide. Une fois détecté, le synchroniseur extrait la trame complète et l'envoie pour traitement. Ce bloc est essentiel pour éviter la perte de données due à un décalage temporel ou des erreurs de synchronisation.

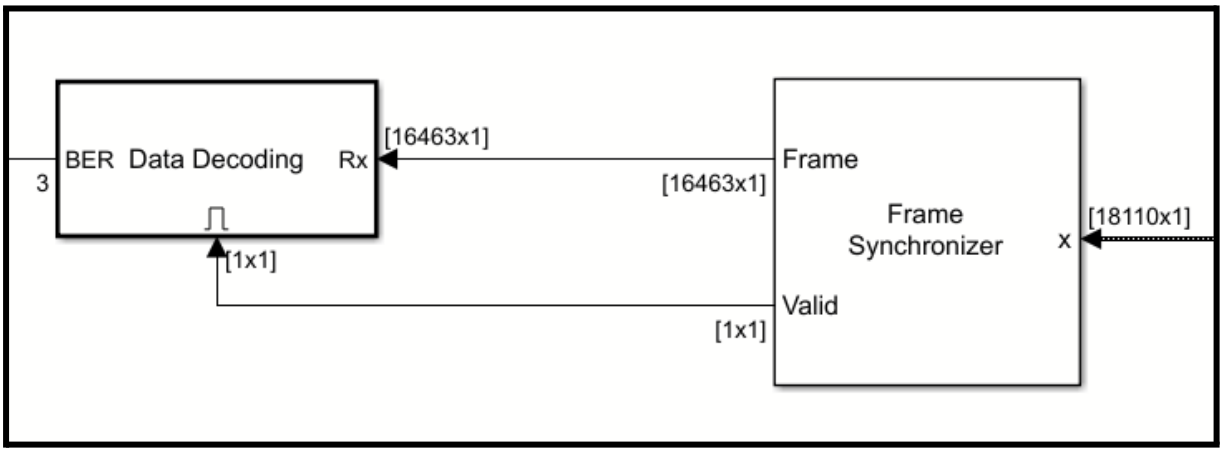


Le premier paramètre indique que le préambule est un signal de référence inséré au début de chaque trame pour faciliter sa détection. Ici, il est basé sur le code de Barker (une séquence bien connue pour la synchronisation), modulé en QPSK avec un facteur d'échelle sqrt(2)/2 pour normaliser la puissance. Ce préambule permet au synchroniseur de repérer précisément le début des trames dans le flux reçu.

Le second paramètre définit le niveau de corrélation minimal entre le signal reçu et le préambule pour considérer qu'une trame valide a été détectée. Une valeur de 0.9 signifie que la correspondance doit être très élevée (90 % de similarité), ce qui réduit les fausses détections mais peut aussi rendre la détection plus stricte, nécessitant un bon rapport signal/bruit (SNR).

Le troisième paramètre définit la longueur des trames extraites après détection du préambule. Une fois la synchronisation effectuée, le synchroniseur extrait exactement 16 463 échantillons pour former une trame complète. Cela permet d'assurer une cohérence dans le traitement des données, en garantissant que chaque trame contient toutes les informations nécessaires pour la démodulation et le décodage.

Le dernier paramètre à déjà été expliqué pour d'autres blocs et peut donc être relu en haut, nous allons donc expliquer le fonctionnement du dernier bloc du démodulateur QPSK qui est lié au bloc synchroniseur de trame de la manière suivante :



La trame sera logiquement l'entrée du signal principal du décodeur de données nommé Rx recevra la trame reçue par le synchroniseur de trames (depuis le port Frame vers le port Rx). L'information de la validité de la trame est transmise par le port Valid vers un second port du décodeur de données qui sert simplement à autoriser ou non la transmission et le décodage en fonction de la validité de la trame.

Le bloc décodeur de données permet de :

- Convertir les symboles reçus en bits. En modulation numérique comme la QPSK, chaque symbole représente 2 bits. Le décodeur associe donc chaque symbole détecté à sa correspondance binaire en utilisant un schéma de constellation (souvent Gray codé pour minimiser les erreurs). Cette étape est cruciale pour retrouver la séquence binaire originale envoyée par l'émetteur.

- Durant la transmission, le signal peut être altéré par le bruit, l'interférence ou l'atténuation du canal. Pour corriger ces erreurs, des codes correcteurs comme Viterbi (pour convolutionnel) ou Reed-Solomon sont utilisés. Le décodeur applique ces algorithmes pour détecter et corriger les erreurs, garantissant ainsi une récupération fiable des données malgré les perturbations.

- Pour aligner les trames, l'émetteur insère souvent un préambule (ex. Code de Barker), qui est utilisé par le Frame Synchronizer. Une fois la synchronisation assurée, ce préambule devient inutile et doit être retiré. Le décodeur supprime également les éventuels bits de contrôle ou marqueurs de début/fin de trame pour ne conserver que l'information utile.

- Après correction des erreurs et suppression des données auxiliaires, les bits obtenus sont convertis en données lisibles. Par exemple, si le message était en ASCII, chaque groupe de 7 ou 8 bits est traduit en caractères pour reconstruire le texte original. Cette conversion permet à l'utilisateur ou au système final de comprendre l'information transmise.

Et c'est la suppression du préambule qui explique le passage d'une taille de trame passant d'une matrice de dimension [18110x1] à une dimension [16463x1], et c'est ce bloc qui affichera le message.

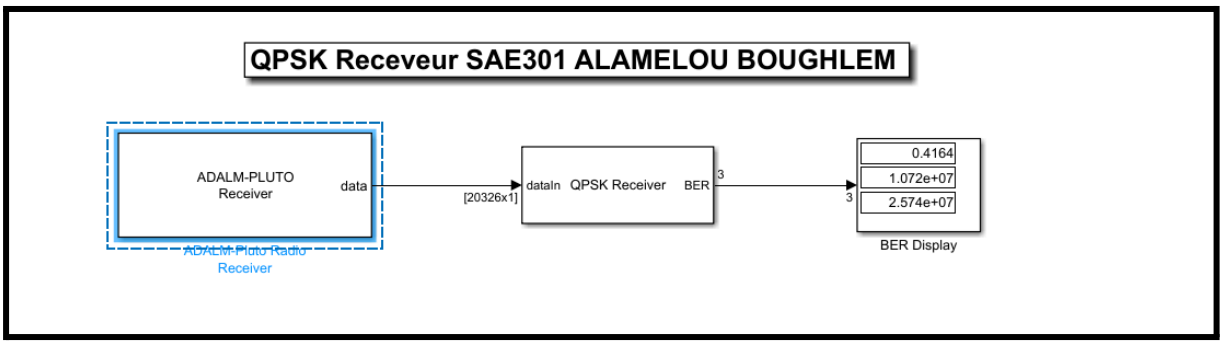
Il convient de noter que plusieurs fois dans le schéma, nous avons placé des analyseurs de spectre et des oscilloscopes qui servaient à reconnaître et essayer d'identifier la source d'erreur lorsque la topologie n'était pas encore fonctionnelle…

Et c'est tout pour l'explication du bloc mineur de l'encodeur de données qui était le dernier bloc mineur du bloc majeur du démodulateur QPSK…

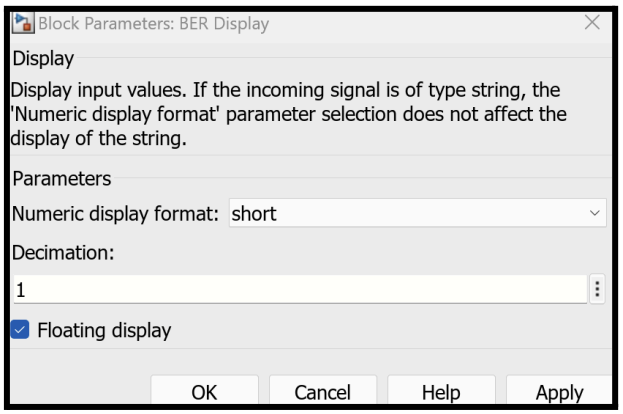
D - L'affichage du Taux d'Erreur Binaire

C'est donc le dernier bloc majeur du récepteur, l'affichage du Taux d'Erreur Binaire (TEB).

Le message sera affiché sur le bas de l'écran de Simulink par le décodeur de données et ce bloc servira à afficher le TEB, il prend en entrée trois valeurs étant le TEB calculé à partir du nombre de bits erronés et du nombre total de bits transmis, des informations fournies par le décodeur de données :



Il est configuré de cette manière :



Le paramètre Numeric Display Format détermine la façon dont les nombres seront affichés dans le bloc d'affichage. Le format "short" signifie que les nombres sont affichés avec une précision réduite, en utilisant 5 chiffres significatifs. Cela permet de rendre l'affichage plus compact tout en affichant une valeur approximative du signal, plutôt que d'afficher des valeurs longues et complexes.

Le second paramètre Decimation spécifie le facteur de décimation, c'est-à-dire la fréquence à laquelle les données sont affichées. Une décimation de 1 signifie que chaque échantillon entrant dans le bloc sera affiché sans modification de la fréquence d'échantillonnage. Autrement dit, l'affichage sera mis à jour pour chaque échantillon du signal d'entrée, sans réduction du taux d'affichage. Si la décimation était supérieure, seuls certains échantillons seraient affichés, réduisant ainsi la fréquence de rafraîchissement de l'affichage.

Le dernier paramètre Floating Display indique que les valeurs numériques affichées seront en notation scientifique (c'est-à-dire en "forme flottante"). Par défaut, l'affichage n'utilisera pas des valeurs décimales simples (ex : 3.01), et choisira des notations comme 3.01e+2. Cela permet de rendre les longs nombres plus faciles à lire.

Ce dernier paramètre d'affichage ne s'applique qu'aux nombres de bits (transmis et erronés) mais pas au TEB qui est une fraction décimale.

4 - Test du fonctionnement

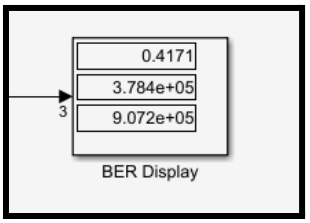
Nous pouvons donc tester le fonctionnement de notre système en tentant de transmettre la phrase suivante :

Je suis un etudiant de RT-2025 Bilel Rohan X

Avec X étant le numéro de la phrase qui est incrémenté pour chaque retransmission de la phrase.

Nous pouvons donc transmettre le message !

Et qu'en est-il de l'affichage du TEB ?



Nous avons donc 378 400 bits erronés pour un total de 907 200 bits transmis, ce qui nous donne un TEB de 0,4171. Un TEB important mais le principal est là, nous avons réussi à remplir le cahier des charges en transmettant nos données.

Dans ce projet de transmission d'une phrase avec deux ADALM Pluto SDR sur MATLAB, utilisant la modulation QPSK, nous avons abordé différentes étapes clés, de la génération du message à sa transmission et réception en passant par la synchronisation des trames et le décodage des données. L'objectif principal était de tester la fiabilité et l'efficacité du système de communication dans un environnement de simulation avec des SDR.

Le TEB observé lors de la transmission a atteint une valeur de 0,4171. Ce résultat est important car il reflète une performance relativement faible dans des conditions réelles de transmission, indiquant qu'une proportion significative de bits sont mal reçus ou corrompus par le bruit, les interférences ou d'autres imperfections du canal. Un TEB de 0,4171 met en évidence la nécessité d'améliorer la qualité de la communication, ce qui pourrait être réalisé en optimisant certains paramètres du système, comme la puissance du signal, l'utilisation de codes de correction d'erreurs supplémentaires ou la diminution de l'impact du bruit.

Pour améliorer ce TEB, plusieurs approches peuvent être envisagées. Par exemple, l'augmentation de la puissance du signal d'émission permettrait de mieux compenser les effets du bruit. Une autre solution serait d'optimiser le filtrage et la synchronisation du signal, notamment avec des techniques de synchronisation de porteuse et de timing avancées. L'utilisation de modulations à ordre plus élevé ou de codes correcteurs plus robustes pourrait également permettre de mieux contrer les erreurs. En somme, l'optimisation du système de transmission serait essentielle pour atteindre des performances proches de celles observées dans des conditions idéales ou en laboratoire…