JMAP Posteriori SNR Speech Enhancement Simulink Implementation for Hardware

By Justin Williams at Flat Earth Inc

# **INTRODUCTION**

The purpose of this document is to outline changes made to the source code of the JMAP Posteriori SNR speech enhancement algorithm, as well as to explain the process of creating an HDL compatible Simulink counterpart. The modified source code for the speech enhancement process can be found in Appendix A. The source code was written/modified in MATLAB R2019A. The contents of this document can be divided into smaller sections that will cover; block descriptions of the frame-based speech enhancement algorithm, MATLAB and Simulink implementations and analysis, and lastly documented changes to the source code from Dallas.

# **TABLE OF CONTENTS**

# **1.0 PROCESS**

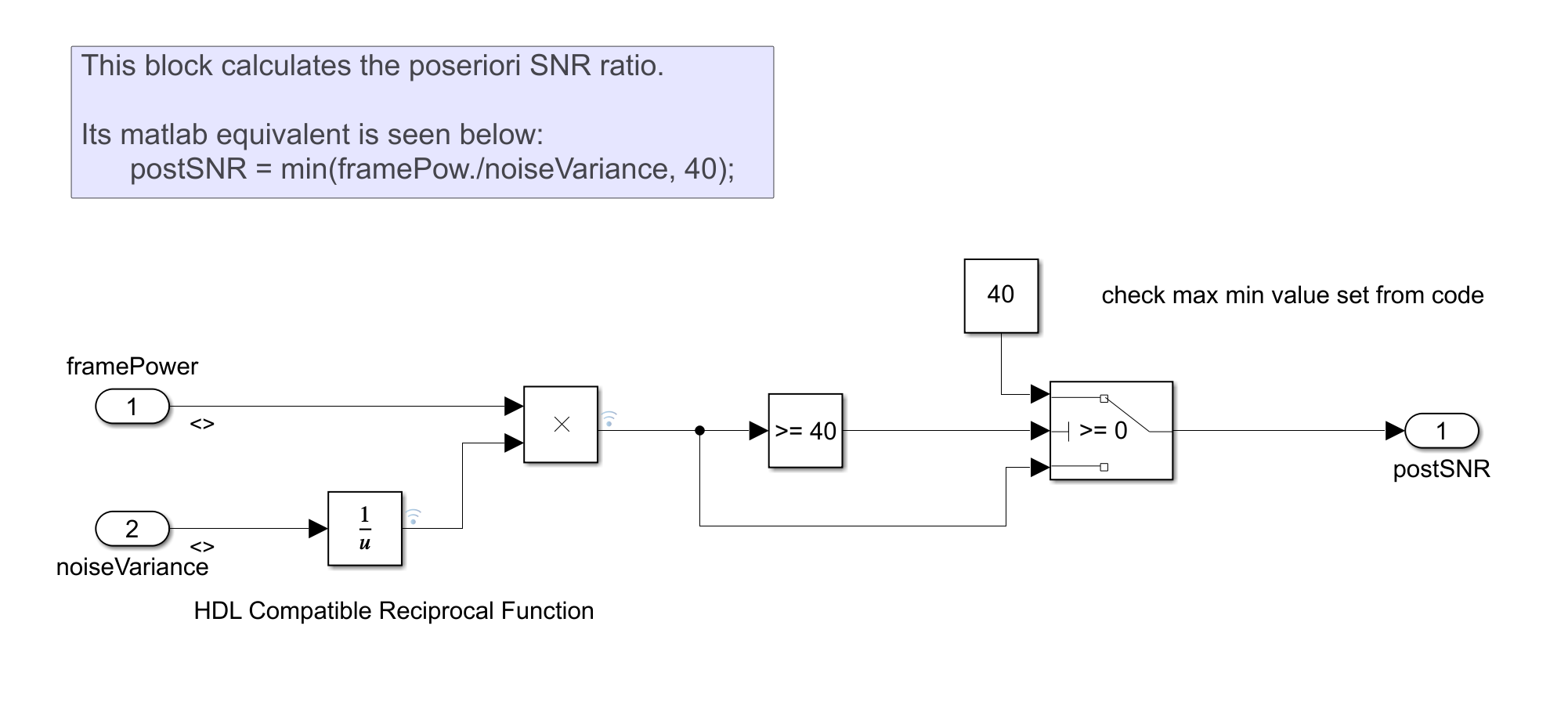
The Simulink blocks were designed by taking large chunks of the algorithm in Appendix A. The Simulink model consists of 5 main blocks where each block contains sub-systems for the proper calculations. Each block has then been isolated and verified by creating a range of random data that takes on the range of values that each variable could normally assume. The outputs of the blocks are then compared to the outputs of the calculations done in MATLAB scripts.

# **2.0 BLOCK ISOLATION & VERIFICATION**

This section is dedicated to breaking down the individual blocks and their results.

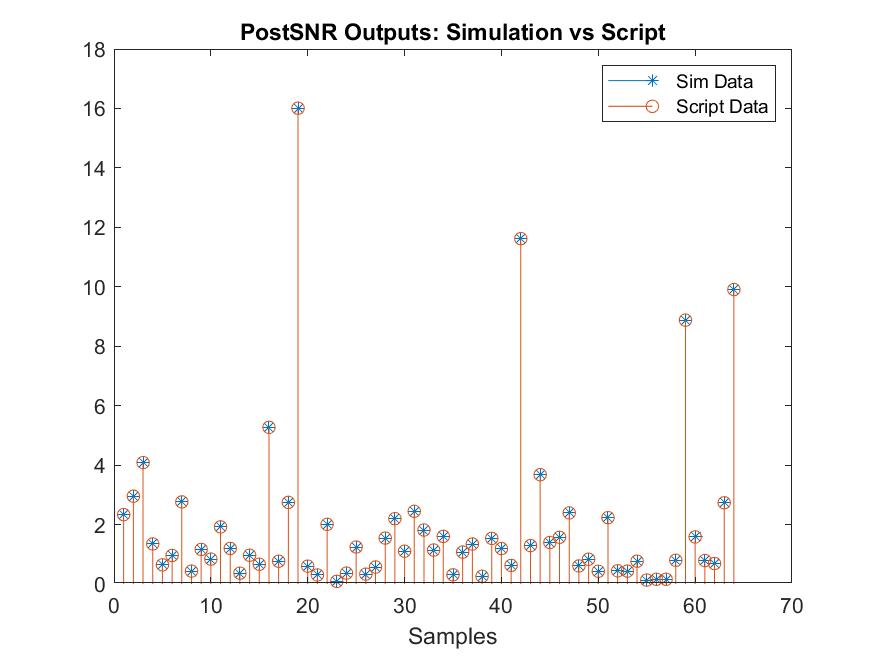
## **2.1 Posteriori SNR Block Verification**

The posteriori SNR ration can be calculated by taking the frame power and dividing it by the frames noise variance. An absolute maximum is inserted if it is very high at a specific sample. A block implementation of this can be seen below.



**Figure 1: Posteriori SNR Simulink Block**

It was found that the min/max HDL Simulink blocks were not working as expected, so a compare-to-constant and switch block were implemented in its place. Here, if the minimum value is above 40, 40 dB is inserted. By creating random values that take on acceptable ranges within the Dallas JMAP model, we can compare the outputs of this block to its MATLAB counterpart.

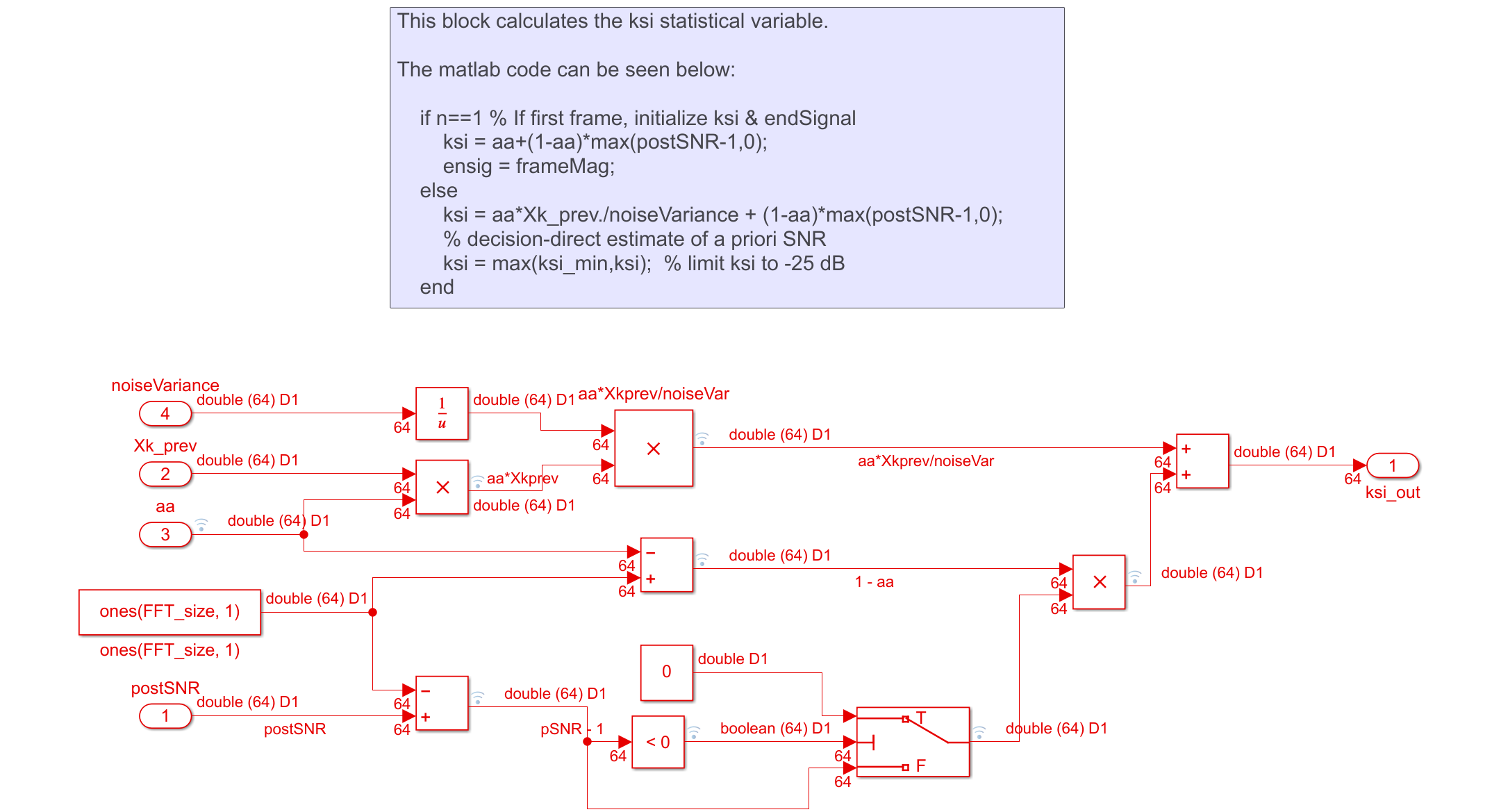


**Figure 2: Frame Outputs of Posteriori SNR from Matlab Script and Simulink Block**

A maximum error of 0.39% has occurred in this HDL compatible block. This calculation is repeated with each new frame of data.

## **2.2 KSI Block Calculation & Verification**

This block is dedicated to calculating *ksi*, which is used to determine the presence of voice activity within the signal. Our model takes care of the if-else statement by calculating the first frames ksi variable. Therefore, our block below seen in Figure 2 shows each iteration of ksi with each frame.



**Figure 2: KSI Simulink Block**

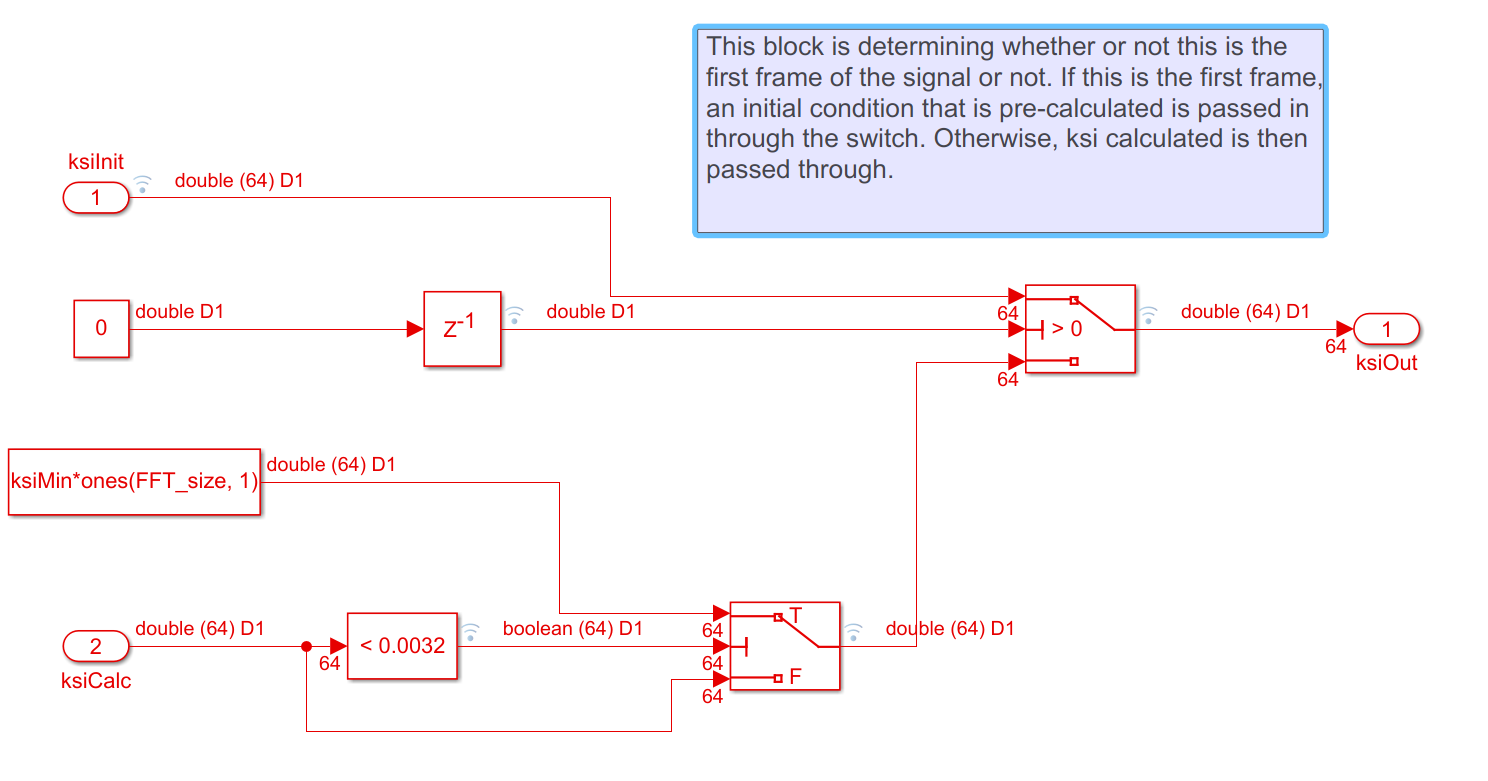
A maximum error of ksi for the Simulink model is 0.378%.

## **2.3 HW Block Verification**

The variable *hw* is used to calculate the high pass filter in order to attenuate the noise. It is derived from the equation below,

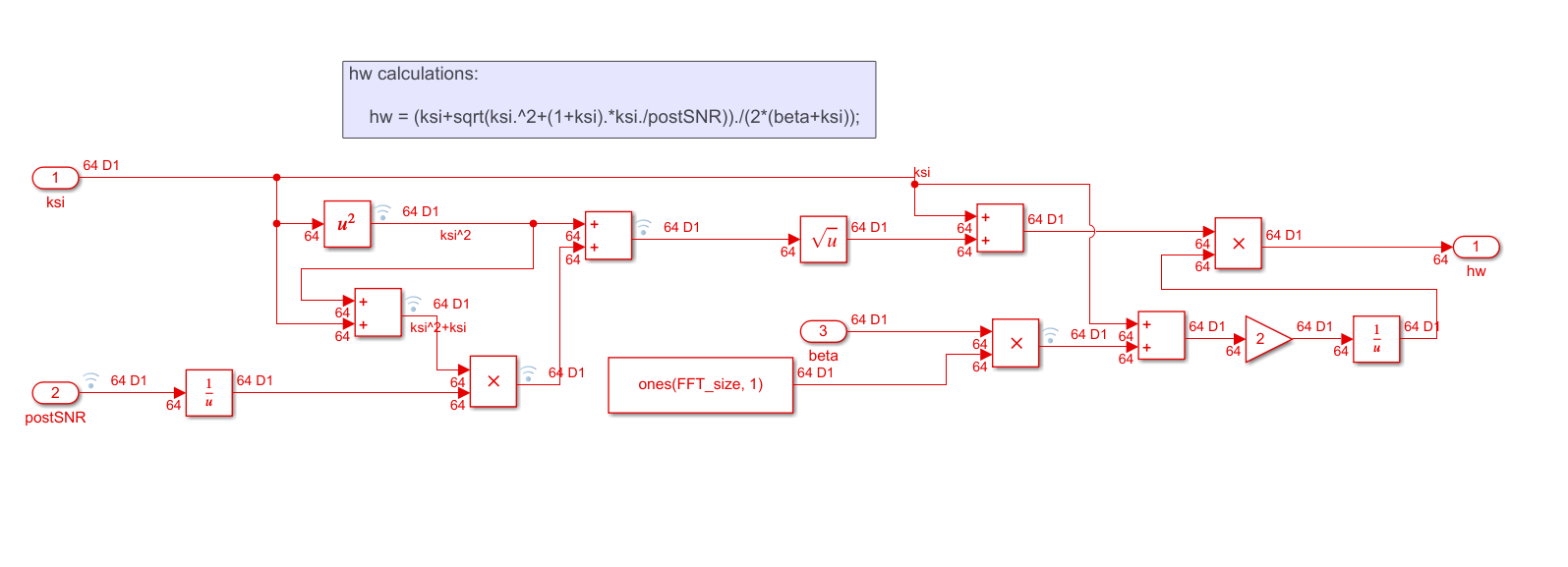
where is ksi, *w* is the posteriori SNR, and is the tradeoff factor.

Note, that to implement this into hardware, a lookup table will need to be made for the expression within the radical in the above equation. The Simulink block diagram representation has two levels, a decision factor determining whether it is the first frame, and the calculation based on this decision. If it is the first frame, then an initial condition of that is pre-calculated is then passed to the subsystem where *hw* is then determined. These two blocks can be visualized below,



**Figure 3: Initial Condition Decision Sub-system**

To re-iterate, the block seen in Figure 3 determines whether this is the first frame using the delay block and an initial condition. If it is the first frame, ksiInit goes through the switch as the output variable to the next sub system seen in Figure 4 below. Otherwise, the calculated value seen in Section 2.2 is passed through as an output with a minimum accepted value of 0.0032. Below is the sub-system where *hw* is calculated,

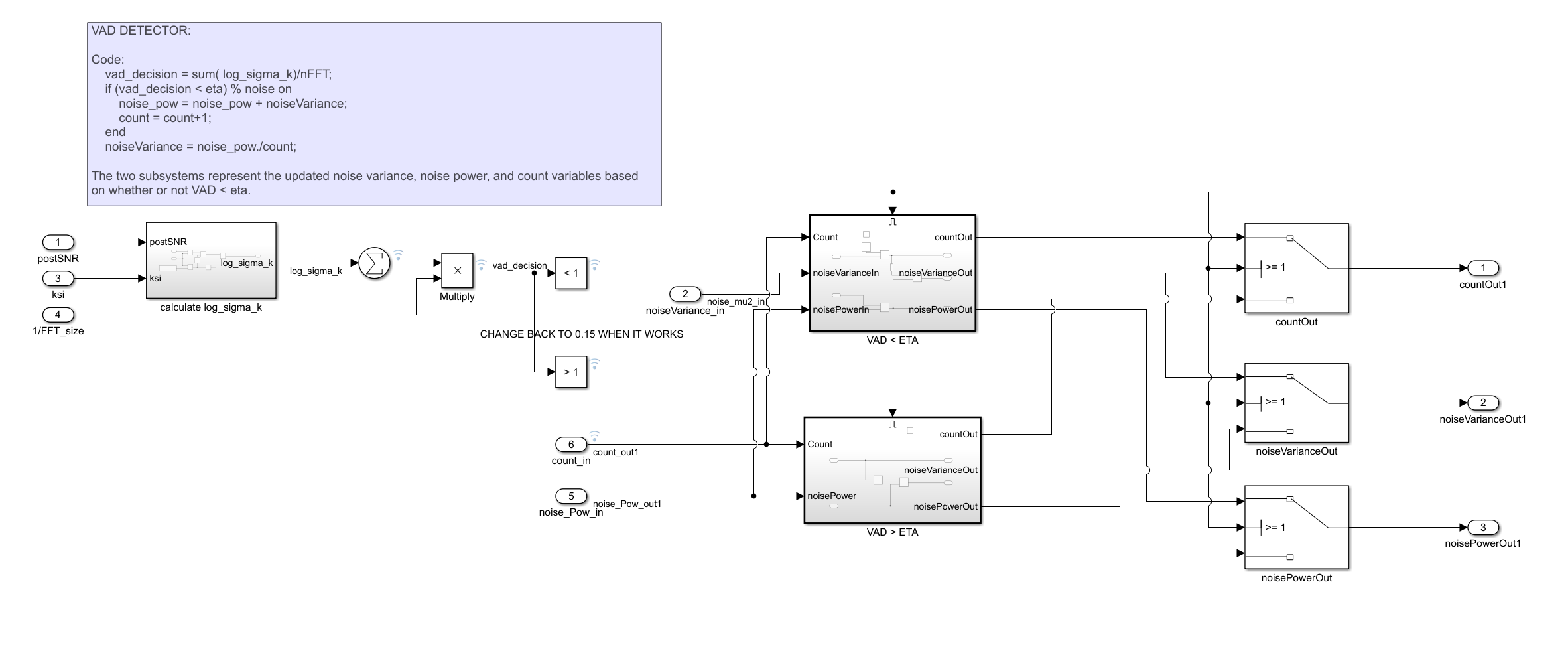


**Figure 4: hw Calculation Sub-system**

As mentioned above, a look-up table will need to be created for the expression in the radical. Otherwise, for a data set of 6 frame inputs, the maximum error for *hw* is approximately 0.563%.

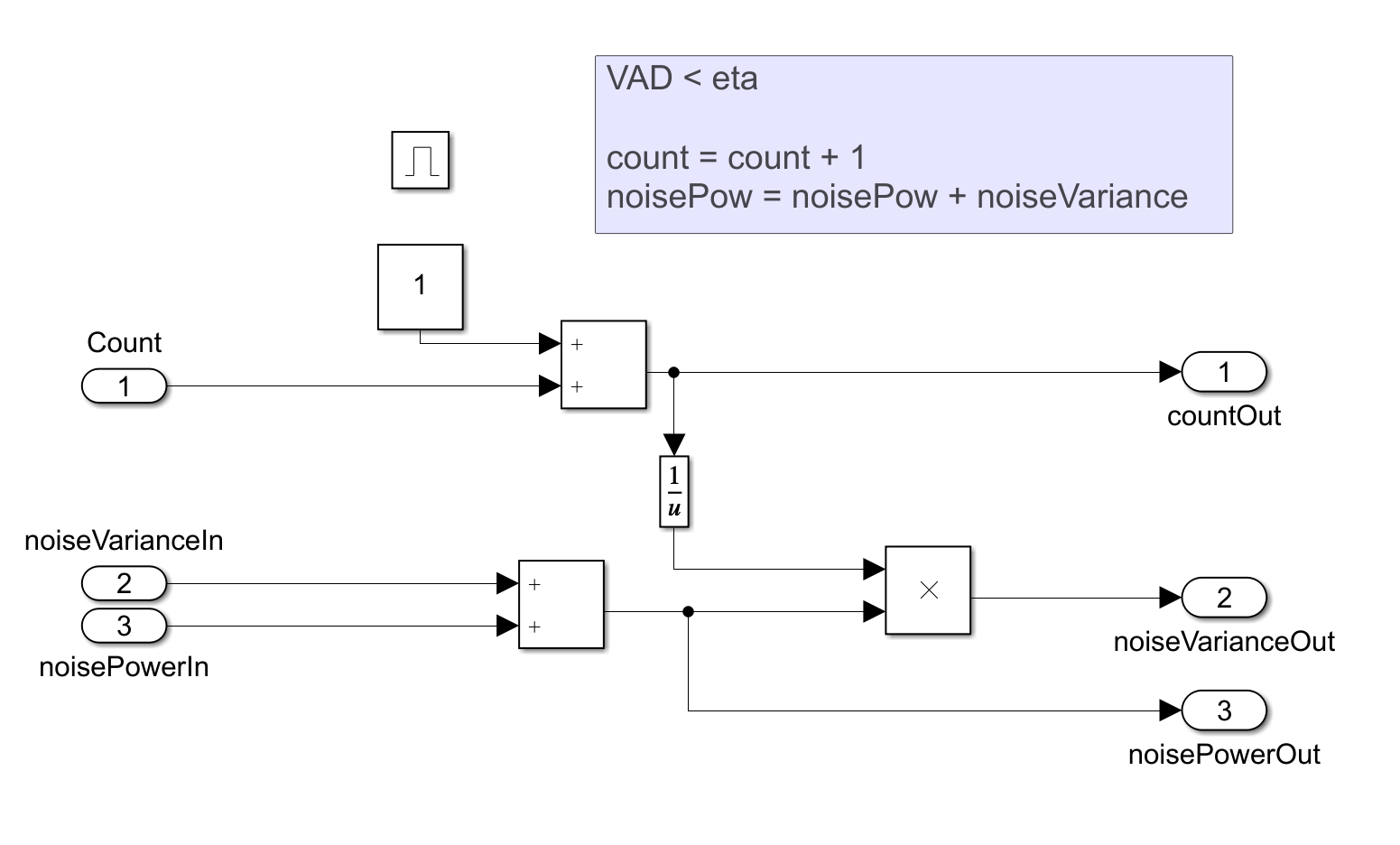
## **2.4 Voice-Activity-Detector Block Verification**

This block is dedicated to determining whether or not there is speech presence. If there is only noise, then the noise power and noise variance are updated. The block representation can be seen below.

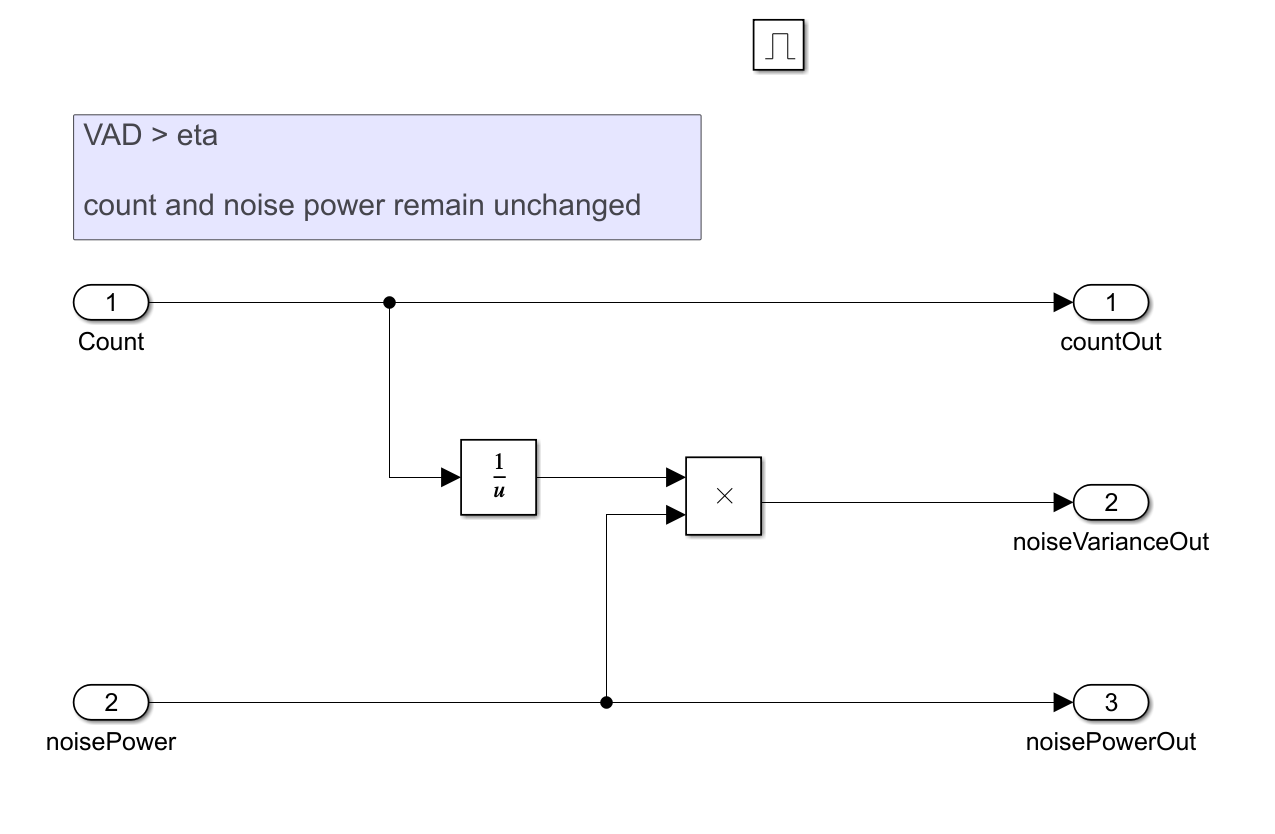


**Figure 5: Top level VAD Detection Block**

The noise variance and power are dependent on the condition of VAD being less than eta. Therefore, if VAD > eta, noise power and count are simply passed through the switch statement. Otherwise, if the VAD < eta, noise power is equal to the sum of noise power and noise variance and count is incremented by one. Some block representations of these can be visualized below.



**Figure 6: VAD < eta – Count & Noise Power Updated**



**Figure 7: VAD > eta – Count & Noise Power Un-changed**

# **10.0 Appendix A: UT Dallas Modified Source Code**

function xfinal = JMAP\_Postfilt\_SE\_OG(x,beta)

%%Implements Proposed JMAP based Speech Enhancement with post

%%processing to reduce musical noise

% Inputs: x - Noisy Speech

% beta - Tradeoff parameter

%

% Outputs: xfinal - Processed Speech Signal

%

% cJMAP Speech Enhancement

% 04/04/17.

% Created by

% Chandan K A Reddy

% Nikhil Shankar

% Ram Charan Chandra Shekar

%

% June 2019

% Modified by

% Justin Patrick Williams

% Copyright © 2017

% Statistical Signal Processing Laboratory, University Of Texas @ Dallas.

% All rights reserved.

% -------------------------------------------- %

% INITIALIZE VARIABLES

% -------------------------------------------- %

len = 64; % Frame size in samples

PERC = 75; % window overlap in percent of frame size

len1 = floor(len\*PERC/100);

len2 = len-len1;

win = hanning(len);

win = win\*len2/sum(win); % Normalized Window Input

Nframes = floor((length(x) - len)/len2);

xfinal = zeros(length(x),1);

% Statistical Variables

aa = 0.98;

eta = 0.15;

ksi\_min = 10^(-25/10);

count = 0;

% ---------------------------------------------------------------------- %

% Noise Magnitude Calculations (Assumption: beginning frames are silent)

% ---------------------------------------------------------------------- %

nFFT = len;

j = 1;

noise\_mean = zeros(nFFT,1);

noise\_pow = zeros(nFFT,1);

for k = 1:6

noise\_mean = noise\_mean+abs(fft(win.\*x(j:j+len-1),nFFT));

j = j + len;

end

noise\_mean = noise\_mean/length(k);

noiseVariance = noise\_mean.^2;

% ----------------------------------------------------------------------- %

% Begin Frame-Based Signal Processing

% ----------------------------------------------------------------------- %

k = 1; % Frame Indexing Variable

for n = 1:Nframes

H = zeros(3,1);

frameInput=win.\*x(k:k+len-1);

%-------------------------------------------- %

% FFT Input Frame

%-------------------------------------------- %

frameInputFFT = fft(frameInput,nFFT);

frameMag = abs(frameInputFFT); % compute the magnitude

framePow = frameMag.^2; % Compute power

postSNR = min(framePow./noiseVariance,40);

if n==1 % If first frame, initialize ksi & endSignal

ksi = aa+(1-aa)\*max(postSNR-1,0);

endFrameSig = frameMag;

else

ksi = aa\*Xk\_prev./noiseVariance + (1-aa)\*max(postSNR-1,0);

% decision-direct estimate of a priori SNR

ksi = max(ksi\_min,ksi); % limit ksi to -25 dB

end

log\_sigma\_k = postSNR.\* ksi./ (1+ ksi)- log(1+ ksi);

% ------------------------------------------------------ %

% VAD: Voice Activity Detector

% ------------------------------------------------------ %

vad\_decision = sum( log\_sigma\_k)/nFFT;

if (vad\_decision < eta) % noise on

noise\_pow = noise\_pow + noiseVariance;

count = count+1;

end

noiseVariance = noise\_pow./count;

hw = (ksi+sqrt(ksi.^2+(1+ksi).\*ksi./postSNR))./(2\*(beta+ksi));

% ----------------------------------------------------------- %

% Musical Noise Suppression

% ----------------------------------------------------------- %

PR = sum(abs(endFrameSig.^2))/(sum(abs(frameMag.^2))+eps);

if(PR>=0.4)

PRT = 1;

else

PRT = PR;

end

if(PRT == 1)

N=1;

else

N = 2\*round((1-PRT/0.4))+1;

end

H(1:N) = 1/N; % FIR Filter Coefficients

HPF = conv(H,abs(hw)); % FIR Filter

endFrameSig = frameMag.\*HPF(1:length(frameMag)); % Filtering endSignal

Xk\_prev = endFrameSig.^2; % postSNR estimation reused for next frame

% -------------------------------------------------------- %

% Inverse FFT current frame, take the real portion

% -------------------------------------------------------- %

xi\_w = ifft( endFrameSig .\* exp(i\*angle(frameInputFFT)),nFFT);

xi\_w = real(xi\_w);

xfinal(k:k + len-1) = xfinal(k:k + len-1) + xi\_w; % Overlap and Add

k=k+len2; % Increment k by (1 - PERC) window size

end

end