SNR IMPROVEMENT REPORT: NON-UNIFORM SAMPLING ADC FILTER

Hao Xiang Yang

University of Toronto

Intelligent Sensory Microsystems Laboratory

Report of the NUS ADC Filter design

Toronto

Summer 2021

**Table of contents**

1. **SNR REPORT -------------------------------------------------------------------------------------- 3**
   1. Definitions---------------------------------------------------------------------------------------- 3
   2. SNR Calculations ------------------------------------------------------------------------------- 3
   3. Examples of SNDR Calculations ------------------------------------------------------------- 4
2. **NUS ADC Filter ------------------------------------------------------------------------------------ 6**
   1. Non-Uniform Sampling ADC Motivation --------------------------------------------------- 6
   2. NUS ADC Filter Motivation ------------------------------------------------------------------ 6
   3. Basic Implementation -------------------------------------------------------------------------- 7
   4. High-Level Synthesis Implementation ------------------------------------------------------- 9
      1. Source Code ------------------------------------------------------------------------- 9
      2. Testbench Files -------------------------------------------------------------------- 10
   5. Block Diagram --------------------------------------------------------------------------------- 11
      1. Basic Nexys Video Board --------------------------------------------------------- 11
      2. NUS ADC Filter Block ------------------------------------------------------------ 11
   6. FPGA Programming -------------------------------------------------------------------------- 12
      1. Writing to Memory ---------------------------------------------------------------------- 13
      2. Testing the Filter ------------------------------------------------------------------------- 14
   7. Difference in Output Due to Decimal Precisions ------------------------------------------ 15

**1. SNR Report**

**1.1 Definitions (from UNP\_A2Dtest)**

The signal-to-noise ratio (SNR) is the ratio of signal power to noise power, expressed in decibels. Noise is qualified as any part of the signal frequency spectrum that is not the main signal frequency. Due to the nature of analog-to-digital converters (ADCs), a finite resolution of N levels of quantization will introduce some quantization noise to the output signal of the ADC since there will be an error difference between the digital output and the analog input.

The signal-to-noise-and-distortion ratio (SINAD or SNDR) is the ratio of the input signal amplitude to the rms sum of all other signal frequencies. The SNDR depends on the input signal’s frequency and amplitude.

**1.2 SNR Calculations (from UNP\_A2Dtest, Jay’s Matlab code)**

Suppose an ADC with N quantization levels. Define where is the least significant bit and is the full-scale voltage (highest normalized voltage). Since the error introduced by quantization is between and (Figure ##), we get:

and .

A picture containing text, antenna

Description automatically generated

Figure ##. Quantization Error for Ideal ADC (UNP\_A2Dtest).

Thus, the SNR of a conventional ADC would be:

where N is the number of levels for quantization.

Similarly, the SNDR can be calculated from the FFT of the input signal:

with a frequency bin m, amplitudes A, a M-point FFT and a frequency bin window of n.

On MATLAB, we can apply SNDR calculations with built-in MATLAB functions. To obtain the spectrum of a signal , we take the Fourier transform of . Then we identify a , which is a set of frequencies containing the most significant frequency of the signal . The rest of the signal would be labeled as . Then we apply the formula above translated in MATLAB (from Jay’s toolbox, Figure ##):



Figure ##. Equation of SNDR in MATLAB.

**1.3 Examples of SNDR Calculations**

For a conventional ADC, the interval between samples is constant, therefore, it adds constant noise to the input signal spectrum. The example below is the spectrum of a sinewave sampled by a conventional ADC. As the presence of higher frequencies on the spectrum suggests, the ADC added some higher frequency noise to the signal (see Figure ##).

Chart

Description automatically generated

Figure ##. Spectrum of Output of Conventional ADC

To improve SNDR by removing the higher frequency noises, we pass the data through a filter (Figure ##). The conventional ADC filter uses a moving average algorithm, that takes the average of a window around a data point as the output.

Chart

Description automatically generated

Figure ##. Spectrum of Output of Conventional ADC after a Filter.

After passing through the conventional ADC filter, the higher amplitude noises were suppressed (see Figure ##) and lowered. By comparison, before the filter, the SNDR of the ADC output was , and after the filter, the SNDR became . A small increase was achieved by the conventional ADC filter.

For reasons specified in section 2.2, the conventional ADC filter cannot be used on the output of a non-uniform sampling (NUS) ADC. Therefore, a new filter was designed to handle NUS ADC output data. The example in below is the spectrum of a sinewave sampled by the NUS ADC. Like the conventional ADC, the NUS ADC also introduces noises of higher frequency to the signal (see Figure ##).

Chart, line chart

Description automatically generated

Figure ##. Spectrum of Output of NUS ADC.

The newly designed filter can eliminate higher frequencies and improve the SNDR by quite a bit. The graph does not look the same as the previous graph since, to avoid long computational time, we limited the number of output data of the filter (see Figure ##).

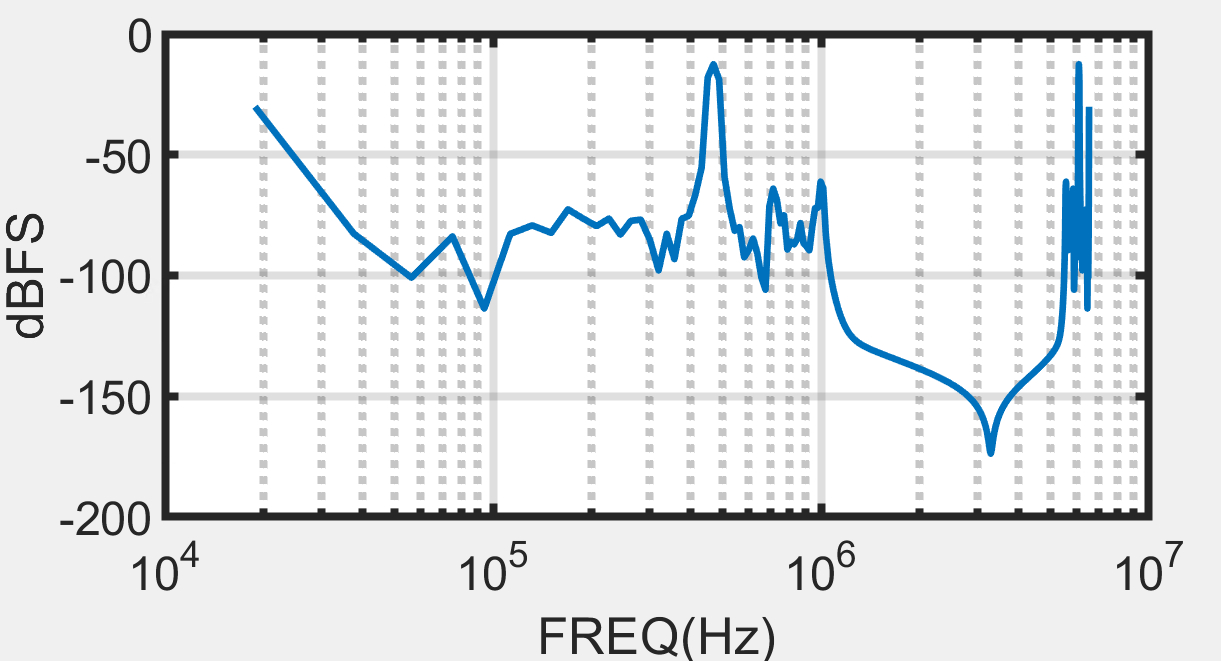


Figure ##. Spectrum of Output of NUS ADC after a Filter.

With the NUS ADC filter, the SNDR of the output data of the ADC improved from to . The motivation and the specifications of this NUS ADC filter are the focus of the next section, section 2.

**2. NUS ADC Filter**

**2.1 Non-Uniform Sampling ADC Motivation**

Traditional analog-to-digital converters (ADCs) sample the input signal uniformly in time. This process creates aliasing, meaning that any frequency components of the input signal that is higher than the Nyquist frequency of the signal would be indistinguishable from its harmonic frequency that is lower than the Nyquist frequency. To solve this problem, an analog anti-aliasing (AA) filter is commonly used to filter out the out-of-band noise and interference. However, modern wireless communication system development requires increased AA filtering agilities over operation bandwidths. In addition, applications, such as biomedical electronics or sensor networks, operate with input signals that are sparse in time. Sampling at constant time would produce useless redundant samples (Wu et al. 2016).

**2.2 NUS ADC Filter Motivation**

A simple way to reconstruct a low duty cycle signals without aliasing is a non-uniform sample (NUS) ADC. A level-crossing quantization NUS ADC would be able to adapt its average sampling rate to the Nyquist rate to produce an error-free signal since there is no voltage quantization error at the level-crossing point. To filter this kind of signal, we could use a conventional AA filter by imposing strict limits of the type of signals received by the NUS ADC. Because the conventional AA filter uses fixed tap delays, it would introduce unwanted aliases, which defies the purpose of a NUS ADC. The introduced filter algorithm would proceed asynchronously to resample the filtered output at a constant rate, and it filters the input signal. The output signal would be able to interact with a synchronous digital signal processing block afterwards. In this section, we will be exploring a simple implementation of the Digital AA filter.

**2.3 Basic Implementation (Refer to Matlab Code)**

The filter implementation follows the equation developed by Wu (Wu et al. 2016):

where is the sampled quantized voltage value of the input signal, is the quantized time valued of , and is the windowing function at frequency .

Although the input of the algorithm is non-uniform, it will evaluate the output at uniform time samples to be processed by a synchronous digital signal processor later. The function is the low-pass impulse response function that cuts-off higher frequencies.

In MATLAB, the integer big N was chosen to be 1000 since experimental testing found out that increasing the number of N would drastically increase the computation time but would not add a significant improvement to the result accuracies. Also, the experimental filter will only output from to reduce computation time. These numbers can be changed to match user requirements. The MATLAB code below (Figure ##) is the above formula written in code.

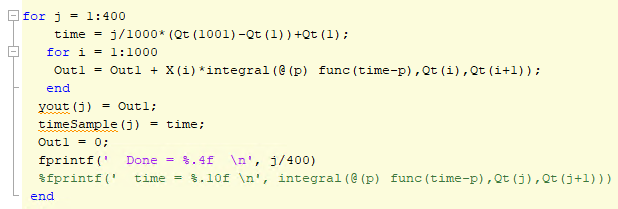


Figure ##. MATLAB Implementation of the Digital AA Filter.

Since the time interval between output data values is constant, we can calculate it beforehand using:

.

To evaluate the effectiveness of the filter, a sin wave with frequency 4684.6 Hz is produced and the filter’s cut-off-frequency is set to 10000 Hz. In Figure ##, the blue line-dotted curve represents the data coming out of the NUS ADC, plotted the voltage quantization with the time quantization. The orange full curve represents the data outputted by the digital AA filter. As the appearance of the curve approaches a sin wave after being processed by a filter, we can conclude that the filter worked as intended to reconstruct the initial signal.

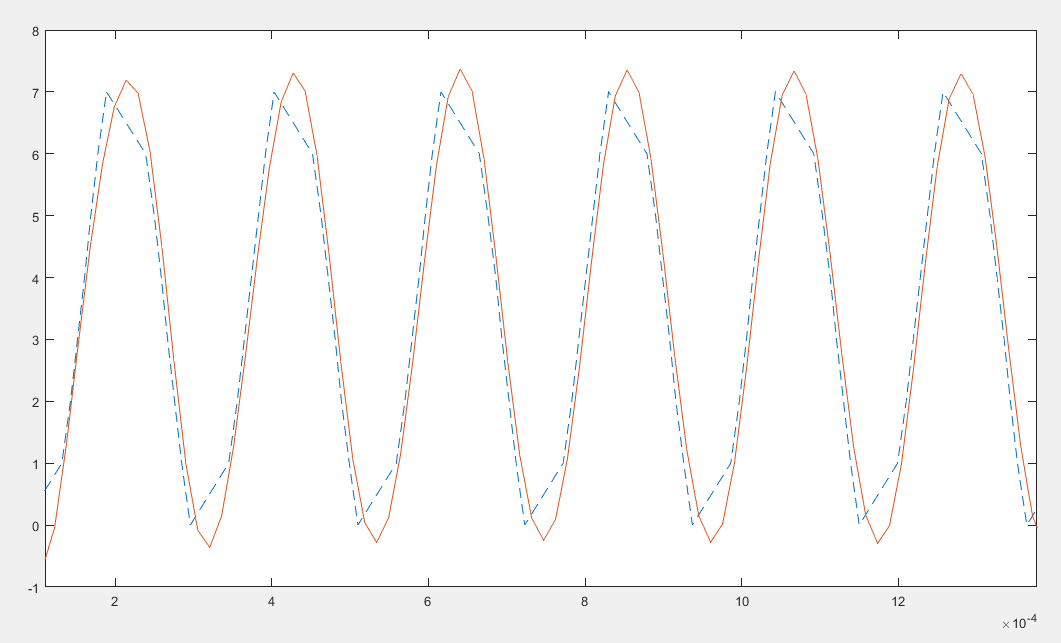


Figure ##. Comparison of Before and After the NUS ADC Filter.

By taking the filter output to the frequency domain (Figure ##), the fundamental frequency of 4684.6 Hz is conserved while the filter acts like a low-pass filter before 10 kHz.

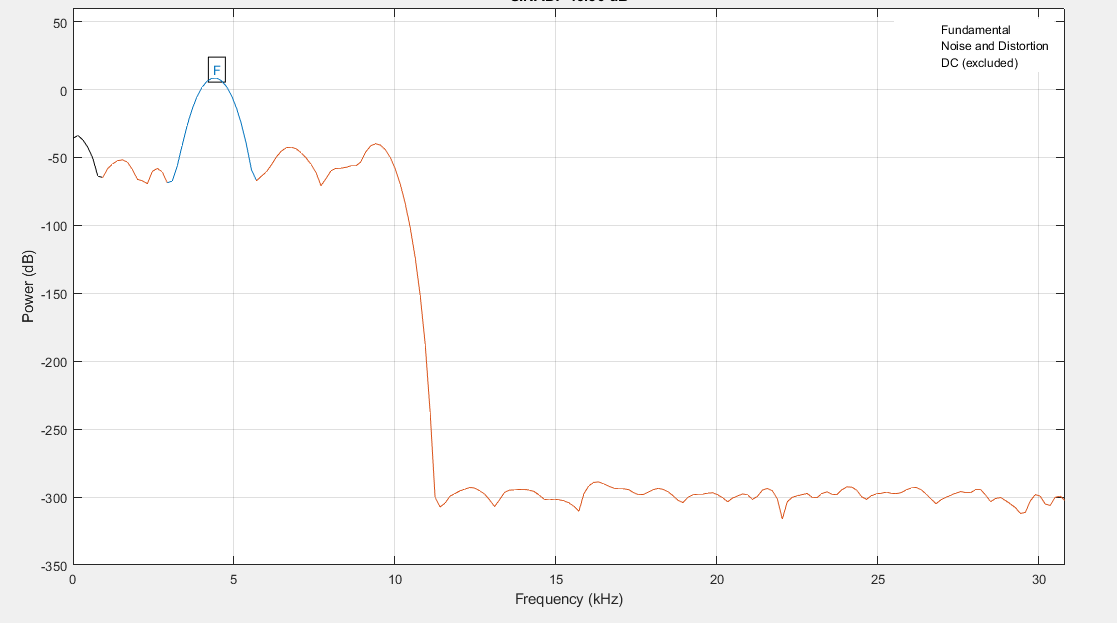


Figure ##. Frequency Domain of the Output of the Filter.

**2.4 High-Level Synthesis Implementation (Refer to Appendix 1)**

**2.4.1 Source Code (nus\_adc\_filter.cpp)**

Since the input data collected would be stored inside memory, the implementation of the NUS ADC Filter in HLS includes a way to access external memory. The top function uses a volatile float pointer to access the DDR memory. Different memory data can be accessed from different memory locations using offset values.

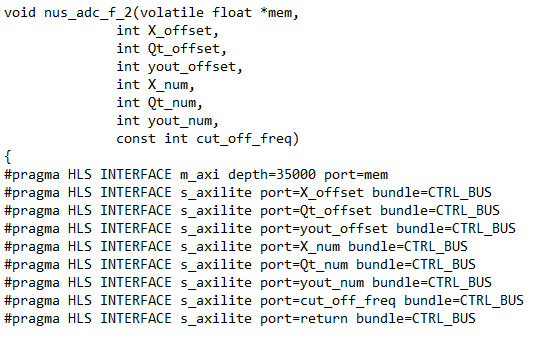


Figure ##. HLS Top Function Parameters.

In the Figure above, the parameters X\_offset, Qt\_offset, and yout\_offset indicate which addresses from the ip block’s starting base address the first data of each array is located. The parameters X\_num, Qt\_num, and yout\_num change how much of input data the filter block would use, and how much data the filter block would produce (Figure ##). All the parameters can be changed via user inputs while coding the FPGA.

Since accessing the memory takes an excessive amount of clock cycles, the HLS implementation of the NUS ADC filter copies all the voltage quantization and time quantization from the external memory at once, does the filtering process, and then upload the filter output all together onto the memory. The filter itself would follow the MATLAB implementation directly.

The integration in the basic implementation is approximated by a Riemann sum (Figure ##).

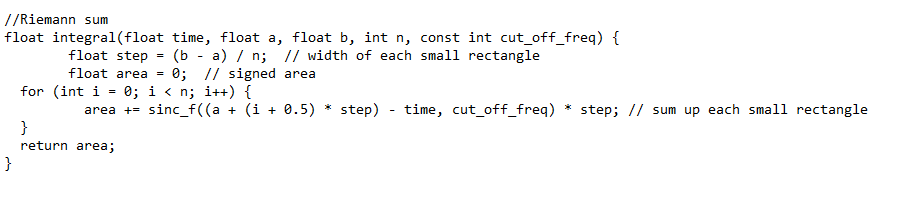


Figure ##. HLS Riemann Sum.

The number of elements in the Riemann Sum is determined to be 100. More elements would lead to a much slower computational speed while not having a significant increase in accuracy, while less element would lose accuracy significantly.

**2.4.2 Testbench Files (nus\_adc\_filter\_test.cpp)**

The testbench uses values produced from the NUS ADC filter implementation in MATLAB. The inputs from MATLAB are fed into the HLS implementation of the filter and the outputs are compared to the “golden” output produced by MATLAB. The testbench creates an artificial memory that would simulate an external DDR memory (Figure ##).

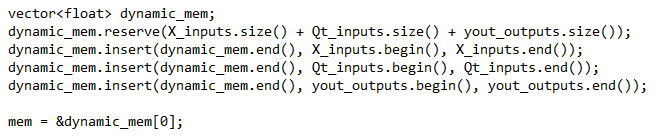


Figure ##. Dynamic Memory With Base Address at &dynamic\_mem[0].

The error between the real output of the HLS code and the “golden” output of MATLAB is a mean square error (Figure ##).

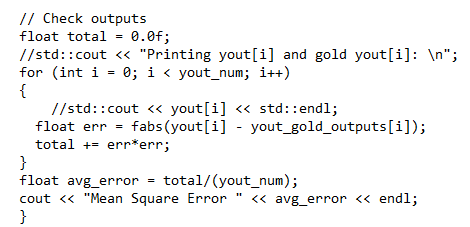


Figure ##. Mean Square Error Calculations.

The mean square error calculation is just the average of the square of the differences between the ideal golden outputs and the actual real outputs of the filter.

With an input frequency of 468 kHz and a cut-off-frequency of 1 MHz, the output of the filter differs with the MATLAB values with about 0.0237657 (Figure ##).

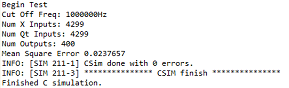


Figure ##. Output Error of the NUS ADC Filter.

**2.5 Block Diagram**

**2.5.1 Basic Nexys Video Board**

The board this project will be implemented on is the Nexys Video board. The MicroBlaze processor would use the system clock of the Nexys Video board. On this diagram, three additional blocks are added:

* The Memory Interface Generator (MIG)
* The AXI Timer
* The AXI Uartlite

The MIG, running at a 200 MHz clock speed, will be connected to an external memory DDR3 and is controlled by the MicroBlaze. The address of the MIG external memory starts at 0x80000000 (see Figure ##).

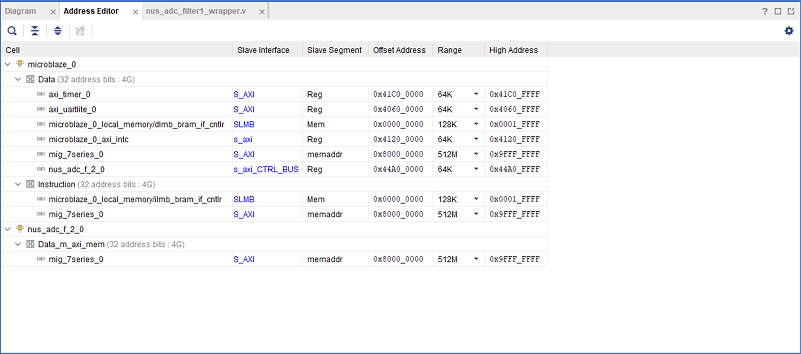


Figure ##. Address Editor of the Block Diagram Components.

**2.5.2 NUS ADC Filter Block**

The NUS ADC filter IP block input port connects to the MicroBlaze AXI Peripheral, which controls all the IP blocks in the block diagram (see Figure ##). The output port connects to the MIG so that the filter IP block can change the values inside the external memory to fit the output of the filter values (see Figure ##).

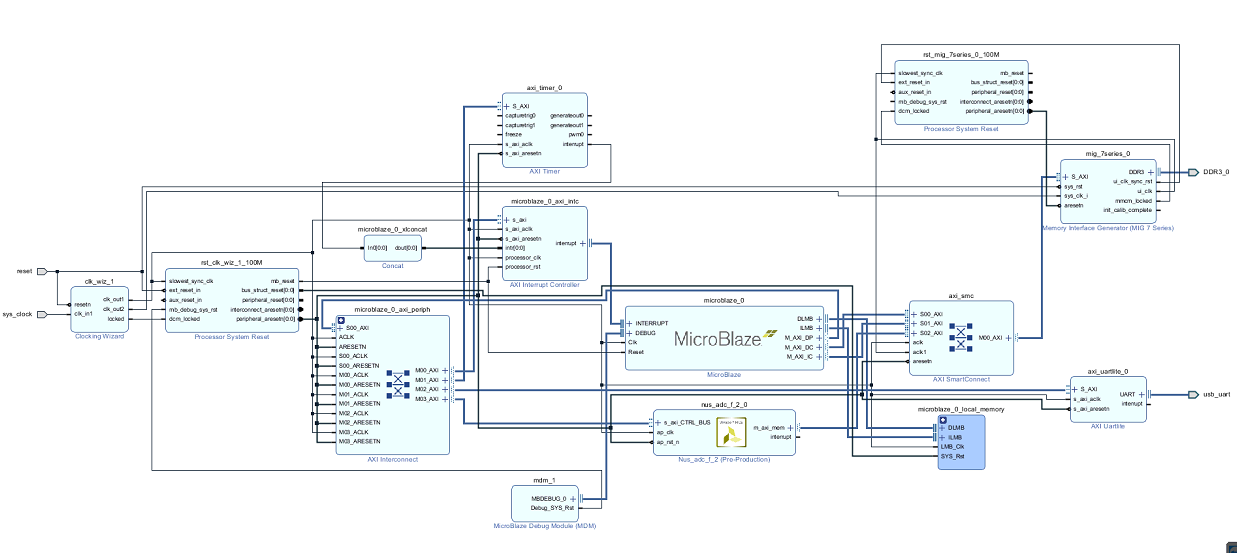


Figure ##. Nexys Video NUS ADC Filter Block Diagram.

The filter IP block has its own memory address starting at 0x44A00000 (see Figure ##). However, every parameter of the IP block has its own address (see Figure ##).

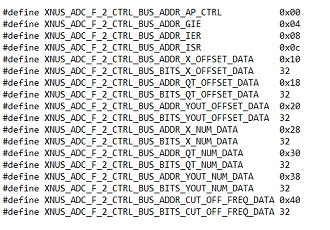


Figure ##. Driver Addresses of the NUS ADC Filter

The first 16-bits are control bits used to start the filtering process and detect when the filter is done with its operations. These bits will be useful for the FPGA Programming in Vitis.

**2.6 FPGA Programming**

To accurately depict the FPGA program on Vitis, two memory addresses must be defined:

* The filter block address at 0x44A00000.
* The DDR external memory address at 0x90000000.

The DDR memory does not start at the very beginning (0x80000000) because we want to make sure the memory used is reserved for filter data only. Since the memory of the MicroBlaze is limited, the instructions and the stack are written to the external memory, which justifies the external memory offset.

The main program will be divided in two parts: writing the initial data onto the external memory and testing the filter block (see Figure ##).

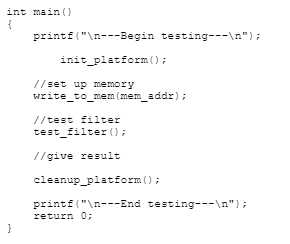


Figure ##. Main Function of processor.c.

**2.6.1 Writing to Memory**

The function write\_to\_mem(mem\_addr) takes the values saved inside two global arrays X\_inputs and Qt\_inputs, and insert their values onto the external memory of the Nexys Video (see Figure ##).

****

Figure ##. Function write\_to\_mem().

By assigning three different pointers to the correct offset addresses of memory where the values of the input data are supposed to be saved, we can ensure that no data is overwritten by mistake. The memory will function like a stack, where the end of X\_inputs values will be the beginning of Qt\_inputs values and so on. To limit the possibilities of mistakes happening, spaces are laid out for yout, the output of the filter, and are initialized to be 0. Those values will be changed later by the NUS ADC filter.

**2.6.2 Testing the Filter**

The function test\_filter() does three operations in order. First, it calculates all the values necessary to initialize the filter as in the nus\_adc\_filter.cpp source code in the high-level synthesis. Then, the function initializes the filter, which starts running and updating the output values inside the external memory. Afterward, the function will calculate a root mean square error using the output values in memory against the golden values from MATLAB. (see Figure ##)



Figure ##. Function test\_filter().

The first operation of the NUS ADC filter is self-explanatory; it is the same logic as in the high-level synthesis test bench files. The only difference is that X\_offset is assigned to be (see Figure ##). Since the address counts in bits and a float takes more than one bit of memory, this step is crucial to avoid accessing the DDR memory beyond the assigned addresses in the block diagram.

The second operation, init\_filter(), starts the filtering according to its parameters. This function is itself divided in three (see Figure ##):

1. Setup.
2. Enable.
3. Poll.

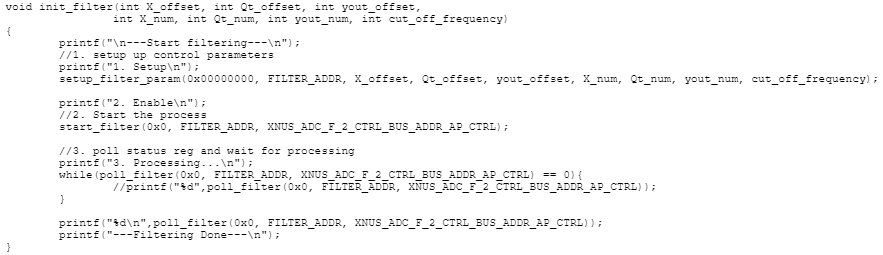


Figure ##. Function init\_filter(parameters).

The function setup\_fitler\_param(parameters) writes the matching parameters one by one to the right place in the NUS ADC filter block by following the address specifications of the driver file produced by the high-level synthesis (see Figure ##). This step ensures that the filter block has all the information it needs to begin the filtering process.

The function start\_filter() enables the filter. To start the filtering process, the function writes 0x1 to the control bit of the filter block, which is the first bit.

For the poll function, init\_filter() will continuously pull the filter control bits (the first 4 bits) for the interrupt signal. This loop allows a waiting time for the filtering process to finish before moving on with the rest of the program. The poll function will look for 0x1 bits in the global interrupt enable register, the IP interrupt enable register, and the IP interrupt status register, which are all supposed to return 0x1 when the NUS ADC filter block finishes running.

The third operation is the error calculation. The error compares values from a golden output stored inside a third global array with the outputs placed in the DDR external memory. The calculations are the same as in the high-level synthesis testbench (see Figure ##).

**2.7 Difference in Output Due to Decimal Precisions**