**Project Design Document (PDD)**

**SAR ADC Simulation, Processing, and Design**

**A picture containing text, monitor, screen

Description automatically generated**

**Author: Devin Santos**

**Date: 4/23/2021**

#### 1.0 Architectural Design 1.0.1 The overarching theme of a SAR ADC is to convert analog data into digital data. To attain a SAR ADC, one needs several sub-blocks. Each block is linked together in such a way that accomplishes the data conversion. In the beginning an input signal, such as a voltage sine wave, is fed into the sample and hold sub-block. On the rising edge of the clock the sub-block captures and outputs the voltage value that is being fed into the input of the sample and hold. The output is fed into a comparator sub-block. The comparator compares the two voltage values and outputs high or low. One value is the sample and hold output, and the other value is the output of the DAC sub-block. The comparator outputs 5V if the sampled signal value is larger than the DAC output value coming from the SAR, and outputs 0V if the sampled signal value is less than the DAC output. The SAR sub-block takes in the output of the comparator to realize which digital value needs to be reached to match the incoming sample. The SAR sub-block consists of D-FlipFlops wired together in a specific way. The digital output of the SAR is fed into the DAC sub-block to get the voltage value to compare with the sample. It takes N clock cycles to get the final digital value that matches the input’s value closely, if not exactly. LTspice was utilized to create a netlist for a 10-bit SAR ADC, using ideal sub-blocks that exist in the LTspice database. Besides the SAR sub-block needing to be designed using ideal flipflops and the DAC sub-block needing a slight equation modification, all other sub-blocks came directly from LTspice. After setting up a specific transient simulation in LTspice, a “.RAW” file is generated and saved. This “.RAW” file is then uploaded into the MATLAB environment, where its recorded data is processed using my MATLAB code. After processing the data in the “.RAW” file, one will be able to visualize the FFT and SNR of the SAR ADC model. Coherent sampling must be used when trying to realize the FFT and SNR of an ADC. This is a method used for getting the best ADC characterization results. It eliminates error sources due the discrete nature of the FFT. A perfectly ideal SAR ADC netlist is used to test the MATLAB code that performs the coherent sampling, FFT, and SNR to ensure the code is working as expected. The comparator is the only sub-block that I was able to design at the transistor level. It turns out that there are many comparator structures that are used for different purposes. When creating a SAR ADC, it is important to evaluate each sub-block carefully, because they all play a crucial role in the overall performance. For this project I had to create the SAR ADC netlist, however it is assumed that the customer will create their own ADC netlist and use the MATLAB code I have generated to test their ADC.

#### 1.0.2

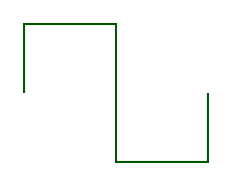
|  |  |  |  |
| --- | --- | --- | --- |
| Customer Need | Functional Element | Physical Element | Related S/W, F/W, HDL |
| Extract Desired Simulation Array | The generated .RAW file | 1D array of recorded data for desired LTspice node | LTspiceMATLAB |
| Extract Coherent Sampled Data | MATLAB code to keep desired samples using coherent sampling | For loops that pick which data samples are valid/desired from the array | MATLAB |
| Examine Frequency Spectrum | Perform FFT on the coherent sampled ADC output signal | Center and normalize to get correct FFT data array | MATLAB |
| Signal-to-Noise Ratio | Separate noise and signal samples from FFT data array | Square all values, sum all values, and take square root to plug into SNR equation | MATLAB |

#### 

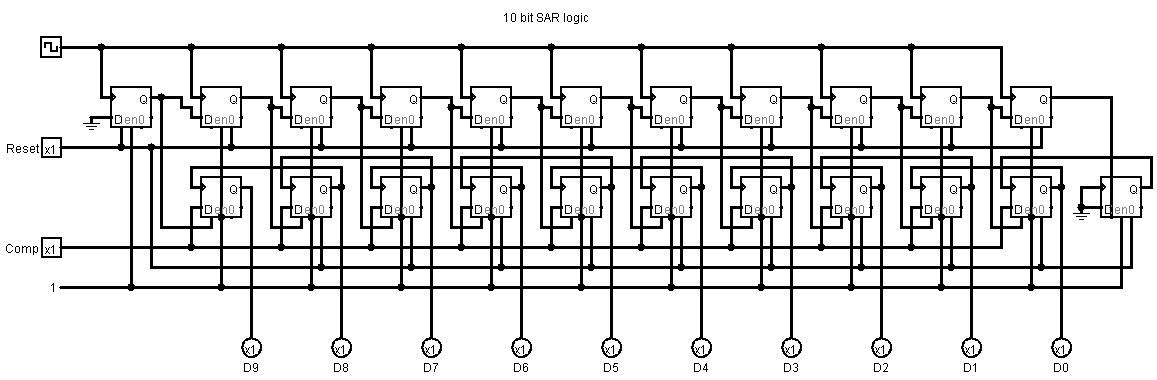
#### Architectural Graphic

#### 

#### 



**Figure 1.) SAR ADC Flow Chart**

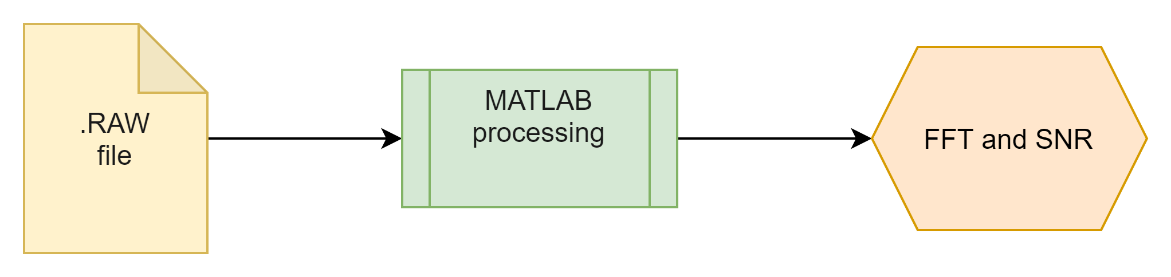


**Figure 2.) SAR Logic sub-block**

Diagram, schematic

Description automatically generated

**Figure 3.) Comparator sub-block**



**Figure 4.) Data Processing Flow Chart**

#### 2.0 Hardware Design 2.0.1 Comparator[4]

**2.0.1.1** The comparator has two inputs. One input comes from the sample and hold circuit and the other input comes from the digital-to-analog converter. The two inputs are both voltage values that can range anywhere from 0V to 5V, respectively. I chose for my SAR ADC to be a single-ended input with a range from 0V to 5V.

**2.0.1.2** The comparator, hence the name, compares two voltage inputs. The more precise the SAR ADC is, the comparator needs to be just as precise. Since my SAR ADC ranges from 0V to 5V and has 10-bits, the comparator needs to be able to determine a difference down to 4.8mV. The comparator also needs to be functional at near 5V and down to 0V. This has proven to be a challenging task to overcome when designing the comparator at the transistor level. The overall operation of the comparator is to determine which input is greater in voltage value and output 5V or 0V, a digital output.

**2.0.1.3** There is only one output from the comparator sub-block. It is ideally a digital value with only two values, either 5V or 0V for logic high and logic low. As shown in the flow chart, the output is fed into multiple D-FlipFlops in the SAR block. The comparator output allows the SAR block to correctly save and output the correct estimated digital value. If the comparator does not output the correct digital value then the final SAR ADC output will be incorrect. Therefore, it is very important to have fully functioning sub-blocks with no errors.

**2.0.2** Sample and Hold[6]

**2.0.2.1** The sample and hold block only has one input. This is the input that is being sampled by the ADC. The input is an analog voltage value that is sampled and held for a specific amount of time. The input is changing constantly due to the analog nature of being continuous. We take samples to convert the analog wave into a finite wave with discrete values.

**2.0.2.2** This circuit has the primary job of outputting the same sampled voltage and remaining unchanged until the next rising edge of the sample and hold clock. On the rising edge of the sample and hold clock, the input is sampled, and the output is held until the next rising edge of the sample and hold clock. It is important for the output to update as quickly as possible, so the overall sampling rate of the SAR ADC is high.

**2.0.2.3** The output of the sample and hold is also an analog voltage value that gets fed into one of the comparator inputs. Since the input range is limited from 0V to 5V for my specific SAR ADC, the output of the sample and hold block will also be limited from 0V to 5V. Of course, designs differ and this is not always the case.

**2.0.3** SAR logic block[5]

**2.0.3.1** The SAR logic block has three inputs, the SAR clock, reset, and comparator output. The SAR clock has the duty of saving and outputting new values from the SAR logic block. As I mentioned previously, I chose a 10-bit SAR ADC because it can measure down to 4.8mV and have a sample rate near 1,000,000 samples per second. There is a direct trade-off with precision and sampling speed. As precision increases, speed decreases, and as speed increases, precision decreases. This is because the more bits you have the more clock cycles you must wait to get correct output. The less bits you have, the less clock cycles you must wait to get correct output. 10-bits seemed to be a good middle point between speed and precision that can be used in multiple applications. The reset input has the job of resetting the SAR logic block. The SAR logic block consists of many D-FlipFlops that must be reset due to the structure of the design. The reset goes high one SAR clock cycle after the 10 SAR clock cycles. This is to ensure the correct output is captured and recorded. The comparator output feeds into specific flipflops to control what digital values are outputted. All inputs to the SAR logic block are digital values, ranging from 0V to 5V for logic low and logic high.

**2.0.3.2** The SAR logic block works by starting at an initial digital value estimate, which in my case is 2.5V. This is because my max voltage is 5V and the minimum is 0V. This is done by outputting a 1, logic high, at the most significant bit position of the digital output. The digital output is fed into the DAC and the DAC outputs 2.5V to the comparator input. Once the comparator outputs a 0 or 1, the SAR logic clock rises and saves the MSB and outputs a new digital estimate value. This is repeated 10 times until the digital output value is as close as it can be to the sampled input voltage.

**2.0.3.3** There are 10 individual outputs from the SAR logic block that are grouped together to realize a binary number. That binary number is then used to represent the sampled input voltage value. With 10-bits we have integer values from 0 to 1023. Since I chose an input range from 0V to 5V, I have a precision of 5/1023, which is roughly 4.8mV. To realize what the analog input voltage was, we multiply the decimal integer value that is being represented in binary by the precision. For example, if the digital output is 1023, we multiply that number by 4.8mV, which gives roughly 5V. I say roughly because 4.8mV is a rounded number. This is the big picture idea on how we relate the digital output value to the analog input voltage value.

**2.0.4** DAC[7]

**2.0.4.1** Since I am using 10-bits I need a 10-bit DAC. Therefore, there are 10 inputs to this DAC. The inputs must be in the correct order from MSB to LSB. The inputs are also digital values, meaning 0V for logic low and 5V for logic high. These inputs are coming from the SAR logic block output. The DAC also has a reference voltage, so it knows what the minimum and maximum values are.

**2.0.4.2** The reason the DAC is needed is to compare the SAR logic blocks digital estimate with the actual sampled input voltage. Since I used an ideal DAC in the netlist it uses an equation to take in the inputs and output the correlating voltage value.

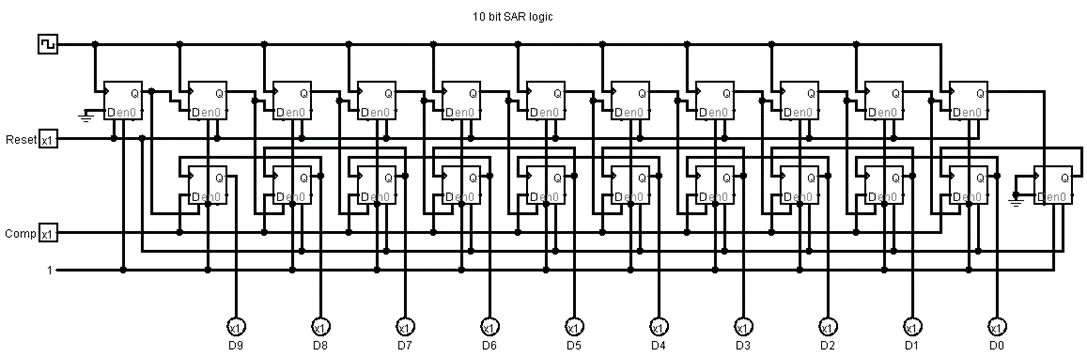
**2.0.4.3** The DAC has only one output, which is the voltage value being fed to the comparator. It determines the voltage value from the set of inputs. The output can range from 0V to 5V, with roughly a 4.8mV step size between output values.

#### 2.1 Theory of operation The SAR ADC sub-blocks have been explained in detail; however, the customer is going to create their own ADC with their own reasoning. The overall theory of operation will be focused on what I am providing to the customer. The customer will have access to my MATLAB code that has been created for the purpose of computing the FFT and SNR from the ADC output wave. I had desires of adding additional code to the script to compute more ADC specifications, but I was unable to manage my time well enough to complete this. The customer will have ideally completed their netlist or schematic in LTspice, just as I have done for this project. They will also have to create their own timing parameters, so the ADC is clocked properly. This means the customer will have control of how many bits their ADC is, the input range, and the sampling speed. I have created the MATLAB code to be as widely accepted as possible, meaning it does not have to be specifically like my SAR ADC to do the computing process. Once the customer has completed their ADC behavioral model in LTspice, they are ready to simulate the ADC. They will need to know their sampling rate, number of samples, and number of cycles to be used for coherent sampling. The correct input frequency will be needed based on their ADC sampling rate. The user is free to choose an odd integer value of sample windows that is greater than one. If their input frequency rate is 1Hz then the minimum amount of time to simulate would be three seconds, for three complete cycles of the input wave. My specific SAR ADC sampling rate is 909090.9091Hz, and I chose a window size of 3 with 256 data points. The amount of data points must be a power of 2 (e.g., 64, 128, 256, 512, etc.). The input frequency equals number of windows divided by number of samples and then that multiplied by ADC sampling frequency. After the customer has found the required input frequency needed, they should add that to the simulation parameters in LTspice and run the simulation. After the simulation has been ran a “LTspice\_netlist\_name.RAW” file will be generated by LTspice and saved in the same folder as the netlist file. The customer will then copy and paste this .RAW file in same folder they have the MATLAB .m file. The user will then open MATLAB and run the code to get the FFT and SNR of their ADC. Based on these outputs the user can determine if their ADC is functioning as desired. The user is then free to make modifications to their behavioral model sub-blocks and repeat the process to see if the ADC is getting better or worse.

* + 1. To extract the simulation variables from the LTspice generated .RAW file, an already existing MATLAB .m file was used. This file is called “LTspice2Matlab.m” and can be found on Peter Feichtinger’s GitHub page. This .m file takes in the .RAW file and creates a structure with all the data. I had to create additional MATLAB code to instruct the user to enter their desired variable name to be extracted from the data file. The variable name is the same as when you add a trace after running the simulation in LTspice. The user selects the correct variable name from a provided list of all variables included in the simulation. This is done in the MATLAB command window.
    2. The user will see variable names at the beginning of the MATLAB code I have provided, whose values need to be updated with the correct coherent sampling values. The user will enter their number of sample windows, ADC sampling frequency, and number of data points. The code will then know the input sampling frequency and step size needed for the correct data extraction to follow coherent sampling. The code uses the time array to get which time values the correct data is. The code then uses a correct time value array to extract the correct data values.
    3. MATLAB’s FFT function is used to go from time domain to frequency domain. Additional lines of code are needed to then center and normalize the FFT array to get useful data. A figure is displayed showing the user the FFT plot where they should see their signal with the correct amplitude and frequency value. All other data points in the FFT array are noise data points.
    4. The SNR value is found by using the FFT array. The signal and noise data elements are separated to calculate their values. As mentioned previously, the signal and noise data points are squared, summed, and square rooted. These values are then plugged into a logarithmic equation to find the SNR in dB. This SNR value should be close to 6.02\*N + 1.76, where N is number of ADC bits.

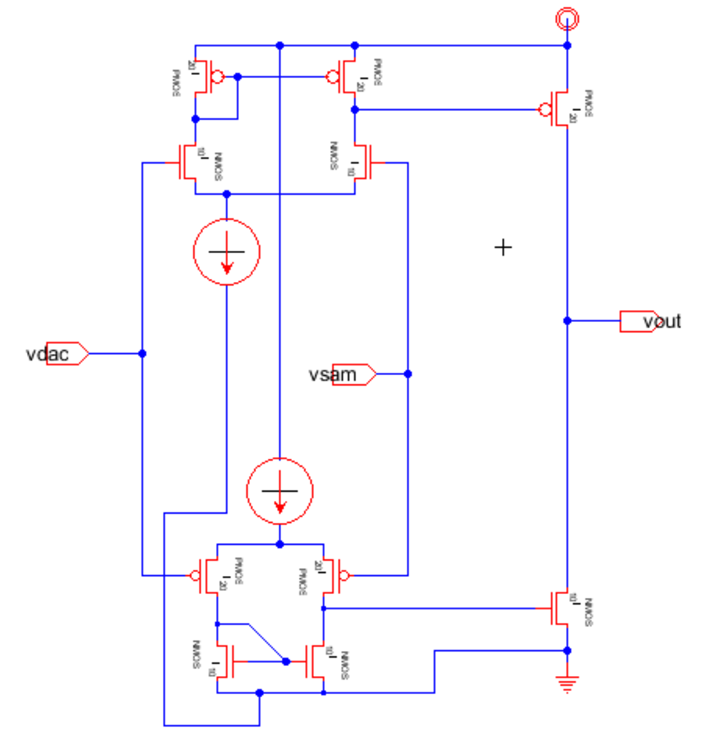
#### 2.2 Schematics

**Figure 5.) Overall Block Diagram**

**2.2.1** SAR Logic

Table

Description automatically generated **2.2.1.1** LTspice Netlist of SAR Logic

**2.2.2** Comparator

Chart, scatter chart

Description automatically generated**2.2.2.1** LTspice Netlist of Comparator

**2.2.3** Sample and Hold LTspice Netlist

Graphical user interface, text, application, email

Description automatically generated**2.2.4** DAC LTspice Netlist

#### Graphical user interface, application Description automatically generated2.3 State Flow Diagram

**2.3.1** The flow of the program starts with the LTspice behavioral model. The behavioral model describes the hardware components and their connections.

**2.3.2** The user will then set appropriate simulation parameters that will control the way the behavioral model operates. This will include timing, such as clock signals, and controlling when to output the digital value. Once these parameters are in place the user will run the simulation for a specific amount of time to capture a specific amount of wave cycles for coherent sampling.

**2.3.3** After the simulation is finished running the user will find a .RAW file in their folder location where the netlist file is saved. They will then copy and paste that .RAW file in the location where they saved the MATLAB .m file.

**2.3.4** Once the user has saved the .RAW file in the same location as the .m file, they are ready to run the MATLAB program.

**2.3.5** After running the program the user will see several graphs and a SNR value. By examining these graphs and the SNR the user will be able to get a rough idea on how their ADC is performing.

**2.3.6** The user can then go on to design the sub-blocks using a program such as Electric VLSI. Once the user is finished with the design, they can translate it into a LTspice netlist. If using Electric VLSI, this can be done by writing spice deck. They will now have an updated behavioral model and can repeat the process to see if the FFT and SNR results improve.

#### A picture containing chart Description automatically generated2.4 Timing Diagram

**2.4.1** At the top of the timing diagram is the valid data clock. This signal clocks a group of D-FlipFlops that hold the final digital value after 10 SAR clock cycles. On the rising edge, 50ns before reset goes high, the output of the ADC is updated.

**2.4.2** The green signal is the SAR logic clock. It has a period of 100ns with 50% duty cycle. A period of 100ns was chosen to ensure the comparator is outputting the correct value before clocking. If the SAR logic block is clocked too fast, we take a risk of losing data.

**2.4.3** The pink signal is the sample and hold clock. It goes high at the same time as the reset signal, every 1.1us. A period of 1.1us was chosen so that we have time to clock the valid data flipflops.

**2.4.4** The signal on the bottom in red is the reset. This is fed into specific flipflops in the SAR logic block to start it over. After the 10th SAR clock cycle, the data needs to be reset to begin the next sample. This is the reason why the SAR logic block uses D-FlipFlops with a reset input.

A picture containing line chart

Description automatically generated**2.5 Circuit Simulations**   
 **Figure 6.) Comparator Output Timing**

**2.5.1** Above with the pink trace and red arrows is a screenshot of how long it takes for the comparator to update its value. It takes approximately 75ns to change its output value. I decided to add an additional 25ns until clocking the SAR block. By examining this circuit simulation, I was able to come up with 100ns period for the SAR logic block.

A picture containing green

Description automatically generated**Figure 7.) Ideal Behavioral Model Sample and Output**

Chart, line chart

Description automatically generated **2.5.2** In figure 7 the sample and hold output is shown in green and the SAR ADC output is shown in pink. These waveforms were attained using an ideal behavioral model. You can see the signals match each other very closely, which is the overall goal.

**Figure 8.) Non-Ideal SAR ADC Output**

**2.5.3** In figure 8 notice the clipping at the peaks. This happening because I replaced the ideal comparator with the comparator shown in 2.2.2. When the input values to the comparator approach the supply voltage, 5V, the comparator starts to stop functioning properly. This hybrid operational amplifier circuit is the best results I could get when building different comparators.

#### 3.0 Software Design

**3.0.1** Extract Desired Simulation Array [8]

**3.0.1.1** The input to this software component is the generated .RAW file. For this section the function script, LTspice2Matlab, is used to gather all the simulation data into a single structure. The contents inside the .RAW file depend purely on the LTspice behavioral model and simulation parameters.

**3.0.1.2** The LTspice2Matlab function takes in the .RAW file and organizes its data contents to be used. It does not add any new data, it only sorts through the .RAW file and gets the variables with their associated data. The most important variables are the time vector and the output wave of the ADC.

**3.0.1.3** The output of the function is a single structure with multiple data contents inside of it. This data structure is very analogous to a structure in C++, where we can access the data inside the structure using the “dot” notation. For example, if the structure is called “raw\_data” we can access the time vector data by this piece of code “time = raw\_data.time\_vector;”. By using the “dot” notation, I can pick which data I want.

**3.0.2** Extract Coherent Sampled Data [1]

**3.0.2.1** In order to extract the desired simulation data or the coherent sampling data, I need to have the coherent sampling time vector and the original LTspice time vector. These two arrays serve as my inputs. The value of each element and the range of the desired time vector depend on the users coherent sampling specifications. By using the coherent sampling values, the code I have written automatically creates the desired time vector for the user, which specifies the step size and total time.

**3.0.2.2** The main operation that is being performed is an iteration of for loops and if conditions. The for loop runs through the time vectors element by element and takes note where their time values match each other. When it finds a match, it records that column number in a new array. Once this process is complete, the new array with have all the correct column numbers for coherent sampling. The reason the column numbers are needed is because we only want the data at specific points in time. So, these specific points in time are related to a specific column number. These column numbers are then used to extract the associated data from the output variable array.

**3.0.2.3** The outputs from this portion of code is the array with all the desired columns and an array with the coherent sampling data variables. At this point, all the correct arrays have been attained and we can check for correctness by plotting these arrays. The user should be able to see that the correct number of cycles and data points have been found.

**3.0.3** Examine Frequency Spectrum[9]

**3.0.3.1** The inputs needed to compute the correct FFT are the coherent sampled data array, the step size of the desired time array, and the end time of time array. These values depend on what the user chooses for their coherent sampling specifications. These values include the number of complete cycles, number of data points, their ADC sampling frequency, and their input frequency being sampled. As long as the user enters the correct information at the beginning of the MATLAB script, the code has all the information it needs.

**3.0.3.2** MATLAB’s FFT command is used and performed on the output data variable. Prior to performing the FFT, the desired data array is subtracted by its mean to remove DC offset. Then the frequency range and step size are computed using the time array’s step size and the time required to complete the user specified number of cycles. The length of the frequency array is computed to normalize the array after performing FFT. FFTSHIFT is another MATLAB command that is used, along with the length of the array to center and normalize the FFT array.

**3.0.3.3** The final output of this portion of code yields the correct frequency spectrum array of the user’s ADC output signal. This array is then plotted and visualized for the user to examine. The user should be able to see their signal’s frequency and amplitude showing on the graph as a single spike. There should not be any other visible spikes on the graph unless you zoom in to examine the noise. The spike for the desired signal should be at the input frequency being sampled and the amplitude should also match the input signal.

**3.0.4** Signal-to-Noise Ratio[10]

**3.0.4.1** To find what column the signal data is in for the FFT array, the sampled input frequency and frequency axis for the FFT are used. From the input frequency value and the frequency axis array, the code finds which column number the frequencies match. The input frequency will already be known from the coherent sampling specifications and the frequency axis array has already been computed during the FFT process.

**3.0.4.2** After the column of the signal has been found, its amplitude data is extracted from the FFT array. This value is then recognized as the signal. Then all other data elements in the FFT array are assumed to be noise, besides the DC offset column. These noise columns are then extracted into their own array. The noise array is then squared, summed, and square rooted. This gives us the worst-case scenario noise array since we did not take the mean. After this process we are left with a signal value and a noise value, which are plugged into the SNR in dB equation ( SNRdB = 20log(signal/noise) ). It is desired to have a SNR on par with 6.02\*N + 1.76, since that value is the best SNR a specific ADC can have.

**3.0.4.3** The final output result from this section of code is the ADC’s SNR. By examining what this number is the user should have a basic idea of how their ADC is performing. If they have designed a good ADC then their SNR should be close to 6.02\*N + 1.76 and if they designed a poor ADC like myself then their SNR will be several dB lower than that ideal value.

#### Software Inter-Relationships Block Diagram

A picture containing text, electronics

Description automatically generated

#### 3.2 Pseudo code of critical software elements

* + 1. Text

       Description automatically generatedExtract Desired Simulation Array
    2. Text

       Description automatically generatedExtract Coherent Sampled Data

Graphical user interface, text, application

Description automatically generated

Text, letter

Description automatically generated **3.2.3** ExamineFrequency Spectrum

Text, application

Description automatically generated **3.2.4** Signal-to-Noise Ratio

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text

Description automatically generated

#### 4.0 Glossary

|  |  |
| --- | --- |
| **Word or Acronym** | **Meaning in This Context** |
| SAR | Successive Approximation Register |
| ADC | Analog to Digital Converter |
| DAC | Digital to Analog Converter |
| LSB | Least Significant Bit |
| MSB | Most Significant Bit |
| SNR | Signal to Noise ratio |
|  |  |
| FFT | Fast Fourier Transform |
| DC | Constant Voltage |
| MATLAB | Matrix Laboratory program |

#### 5.0 Appendix

|  |  |
| --- | --- |
| **No.** | **URL** |
| 1 | <https://training.ti.com/ti-precision-labs-adcs-coherent-sampling-and-filtering-improve-snr-and-thd> |
| 2 | <https://doc.xdevs.com/docs/_Books/ASIC_Design/cmos%20analog%20circuit%20design%20%28allen%2Cholberg-2002%29.pdf> |
| 3 | <https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1080.html> |
| 4 | <https://ieeexplore-ieee-org.mantis.csuchico.edu/document/8728470> |
|  |  |
| 5 | <https://www.researchgate.net/figure/SAR-Logic-Register_fig11_283721063> |
| 6 | <https://www.electronicshub.org/sample-and-hold-circuit/> |
| 7 | <https://pdfserv.maximintegrated.com/en/an/AN1080.pdf> |
| 8 | <https://github.com/PeterFeicht/ltspice2matlab> |
| 9 | <https://www.mathworks.com/help/matlab/ref/fft.html> |
| 10 | <https://microchipdeveloper.com/adc:adc-snr> |