Deliverable 5

© 2019 Tomi Kalejaiye, Vini Tripathii, Lois Lee (ok93@cornell.edu, ut33@cornell.edu, ll556@cornell.edu)

1 Block Diagram

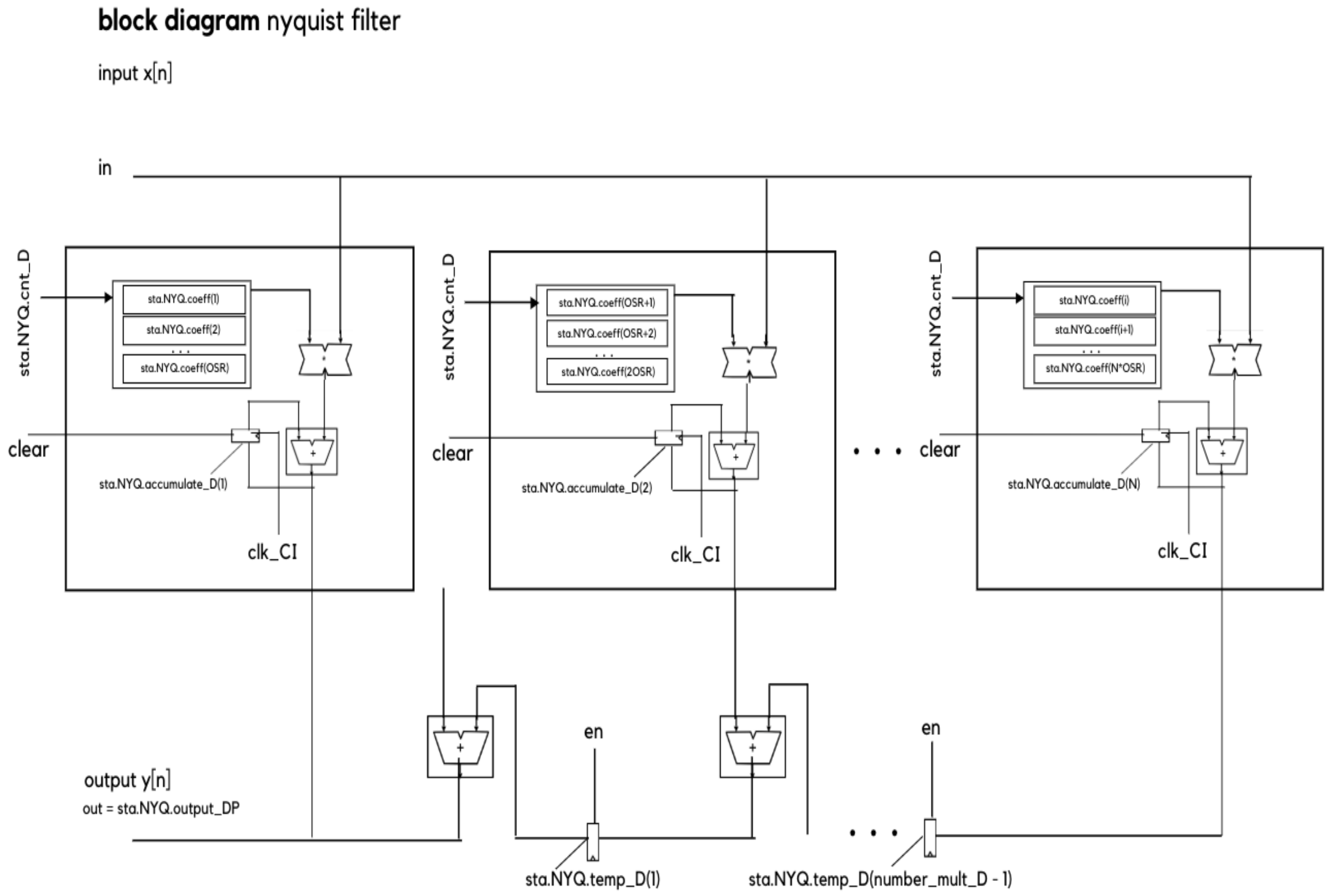


Figure 1: **High Level Block Diagram for Nyquist Filter**
Not shown here is the logic that occurs with the control signals clear and en which work to clear the flip flop in the MAC units and allow passage of data for the output respectively. Also please note that the last MAC block here is shown in more detail than the first two in order to conserve space. The inputs into the multiplier should be the input ″in″ as well as the coefficients from ″sta.NYQ.coeff″ based on the value of counter ″sta.NYQ.cnt_D″

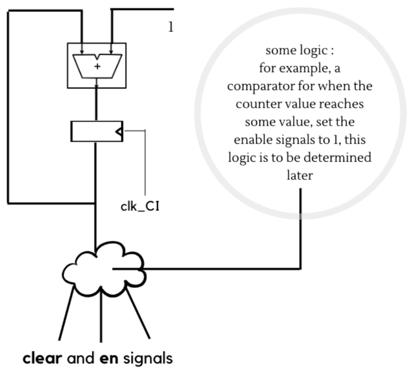

1
+
some logic : for example, a comparator for when the counter value reaches some value, set the enable signals to 1, this logic is to be determined later
clk_CI
clear and en signals

Figure 2: **Counter logic for ”en” and ”clear”**
This figure shows the logic for counter ”sta.NYQ.cnt_D” which increments every clock cycle, and resets automatically once it overflows. After our discussion with Professor Studer, we ascertained that we cannot determine the logic for the ”clear” and ”en” signals at this point in time.

1.1 Block Diagram Elaboration

The sample input here, x[n], is represented here as ”in”. The input parameters include the filter coefficients which are stored in ”sta.NYQ.coeff”. The counter ”sta.NYQ.cnt_D” increases in value every clock cycle and changes the coefficients we multiply the input ”in” by.

The clear and enable signals ”clear” and ”en” are control signals for the flip flops in the MAC unit and between the MAC and adder units. As can be seen in figure 2 below, the value of the counter, ”sta.NYQ.cnt_D” is used to determine the ”clear” and ”en” signals.

2 Functionality of the Nyquist Filter

2.1 Purpose of the Nyquist Filter, Introduction

The goal of our overall project is to create a minimal subtractive digital synthesizer on an ASIC. A digital synthesizer in essence generates digital audio signals which are played as sound. The ASIC has many blocks, and in particular, our group is implementing the Nyquist Filter. A Nyquist filter is a type of low-pass filter that is used to filter out high frequency components of a signal. Low-pass filters are used for subtractive synthesis, to allow certain frequency regions to pass through without being attenuated while significantly subtracting those in others. The Nyquist filter is necessary because although we use a very high sampling frequency on our chip to minimize aliasing and warping, we cannot output a signal at that sampling rate. The bitrate would be too high. We could simply downsample our signal before outputting it, but then aliasing would become a problem again. Therefore, we have to filter out high frequency components first using the Nyquist filter, before outputting at a lower sampling frequency. For our purposes we will be filtering out frequencies above 96kHz. This allows us to minimize aliasing but still provide a good reconstruction of the signal, that has a low enough bitrate to be output accurately by the chip.

For our project, the Nyquist Filter will be receiving input from the Envelope block and its output will go into the I2C block. and functions to remove frequencies above 96kHz from the input signal using filter coefficients.

2.2 Signal and Control Information Generated

The Nyquist Filter generates a new signal to send to the I2S interface. A key purpose of the Nyqyust filter is to decimate (downsample) the input signal – reduce the sampling frequency to a lower value. In order to do this it without suffering from aliasing problems, it is necessary to filter high frequency content – the output signal will have, ideally, no frequencies above 96kHz.

2.3 Inputs and Outputs

The Nyquist Filter block should take in the input from the envelope block, and output a filtered and decimated signal to the I2S interface. Its parameter specifications will include filter coefficients.

2.4 Parameters and States

Parameters	Function
par.GLO.osr	Our block uses the oversampling ratio as a parameter to determine how frequently to output.

States (Write)	Function
sta.NYQ	Our block writes to its own state to update a counter, a buffer of past inputs, and two variables that hold the most recent output and the next output.

We are not reading states from any other blocks.

2.5 Computed Parameters and Additional Details

Filter coefficients are an important aspect to consider. We should also consider the Q value, or resonance.

3 Functionality of Updated Nyquist Filter

3.1 Implementation of Nyquist Filter

The ratio between the internal sampling rate and the output sampling rate (the oversampling rate) and the number of filter coefficients together determine the number of multipliers we need.

$$N = L/OSR$$

where OSR is the oversampling ratio, L is the length of the filter, and N is the number of MAC units.

Since the internal sampling rate is much higher than the output sampling rate, we don't need to do many multiplications every cycle. In particular, we only need to do N multiplications every cycle, to use every coefficient by the time we have to output a sample.

For example, in the third version of the synth file, we see that par.GLO.fs_o, the output sampling rate is 48000 Hz, and that the high internal sampling rate is 384000 Hz, giving us an oversampling rate, par.GLO.osr, of 8.

We don't determine the oversampling ratio, therefore the number of filter coefficients we decide to use will ultimately determine the number of multipliers we use in the design. Because we will have (number of filter coefficients)/(OSR) multipliers, each of those multipliers will multiply one of the coefficients with one of the inputs at any given time.

y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(end)*x(n-length(b)+1)

To use the Nyquist Filter, one can potentially modify the filter coefficients stored in sta.NYQ.coeff accessible through the initialization file called NYQ_init.m.

The advantage of our block design as seen in our block diagram, is that we do not need a buffer to hold past input values to do these multiplications. We multiply each input by multiple coefficients in each cycle, so we don't need to keep it.

4 Performance Checker

We added some elements to our code to be able to check the performance of our filter using MATLAB's inbuilt filter function as a reference. Our performance checker is split among 3 different files. In the NYQ_init.m file, we first instantiate a few vectors as seen below:

```
%=====================================================================
%================            PERFORMANCE CHECKER         ==================
%=====================================================================

sta.NYQ.last_n = zeros(1, length(sta.NYQ.coeff));
sta.NYQ.S = [];
sta.NYQ.denom = zeros(1, length(sta.NYQ.coeff));
sta.NYQ.performance_checker = [];
for i = 1: length(sta.NYQ.coeff)
    sta.NYQ.denom(i) = 1;
end
sta.NYQ.filt_out = [];
```

Here the sta.NYQ.last_n represents the last n inputs signals, while sta.NYQ.S represents the vector of the sums of the outputs from the MATLAB filter function squared. The sta.NYQ.denom is simply an array of 1s to input into the MATLAB filter function as the denominator values. sta.NYQ.performance_checker is the vector of the sums of the differences between the MATLAB filter function and our filter output squared. sta.NYQ.filt_out is just the array of the MATLAB filter function's output using the same coefficients as our filter and the same inputs as well.

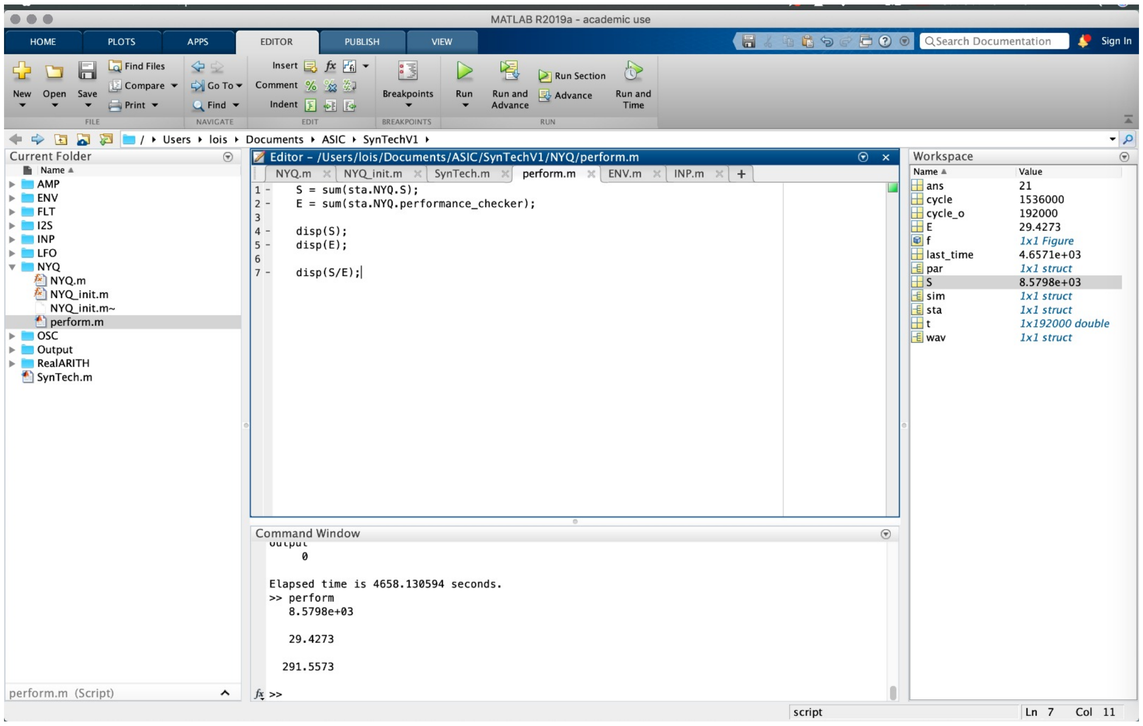
Within the NYQ.m file, we calculate and append to the above arrays. After running the synth.m file, we are left with the sta.NYQ.S and sta.NYQ.performance_checker arrays which are then used in the performance.m file to get the S and E values. S is the sum of all the reference (MATLAB filter) outputs, and E is the squared error (squared difference between our filter and MATLAB filter's outputs) :

```
S = sum(sta.NYQ.S);
E = sum(sta.NYQ.performance_checker);

disp(S);
disp(E);

disp(S/E);
```

We got the following output after running on our 512 coefficient array. S/E is the signal to noise ratio, which is fairly high here: 291.55.


MATLAB R2019a - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Editor – /Users/lois/Documents/ASIC/SynTechV1/NYQ/perform.m
NYQ.m NYQ_init.m SynTech.m perform.m ENV.m INP.m

```
S = sum(sta.NYQ.S);
E = sum(sta.NYQ.performance_checker);

disp(S);
disp(E);

disp(S/E);
```

Command Window

```
      0

Elapsed time is 4658.130594 seconds.
>> perform
   8.5798e+03

   29.4273

  291.5573
```

Name	Value
ans	21
cycle	1536000
cycle_o	192000
E	29.4273
f	1x1 Figure
last_time	4.6571e+03
par	1x1 struct
S	8.5798e+03
sim	1x1 struct
sta	1x1 struct
t	1x192000 double
wav	1x1 struct

5 Testing

In order to test the verilog, we first had to create tests for the input and output files. Using the matlab synthesizer as a basis for the value that we wanted, at each call of the NYQ function we appended to an input and output matrix which we then printed.

We used the following lines of code in NYQ.m
sta.NYQ.inputs = [sta.NYQ.inputs,sta.NYQ.Sample_D];
sta.NYQ.outputs = [sta.NYQ.outputs,sta.NYQ.Out_DO];

Then we used a python script to generate an input and output file by inputting both matrix strings from matlab as well as the coefficient buffer from matlab into this new python script.

In [1]:
```
# THIS FUNCTION MAKES THE INPUTS AND OUTPUTS ONE COLUMN TO PUT INTO TXT FILES
def str_without_commas(string):
    c = []
    string = string.replace('[','')
    string = string.replace(']','')
    string = string.replace(',', ' ')
    mat = string.split()
    for i in mat:
#           print i
        c.append(float(i))
    return c


test_bench_params = "[0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45, 0.5, 0.55, 0.6,0.65, 0.7,0.75, 0.8,0.9,
0.9,0.8,0.75,0.7,0.65,0.6,0.55,0.5,0.45,0.4,0.35,0.3,0.25,0.2,0.15,0.1] "
test_bench_coeffs = str_without_commas(test_bench_params)
```

In [2]:
```
print test_bench_coeffs
```

```
[0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.9, 0.9, 0.8,
0.75, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1]
```

```
def col_print_out(lst):
    for i in range(32):
        print 'x'
    for i in lst:
        print i

def col_print_in(lst, test_bench_coeffs):
    # the number of coeffs
    for i in range(len(test_bench_coeffs)):
        # Rst_RB, WrEn_S, Addr_D, PAR_In_D, NYQ_In_DI
        print '1 1 0 ', test_bench_coeffs[i] ,' 0'
    for i in lst:
        print '1 0 0 0 ', i
```

In [5]:
```
col_print_out(out)
```

```
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
x
```

6 Preliminary Synthesis

While our code did compile, because the block was not exactly functional, our Synthesis did not run as smoothly. We ran into a lot of warning, and while we've included the reports from the Preliminary Synthesis, it is not correct.
to discussion 6, we've generated the 5 most critical paths and included the report in the zip file.