EEM216A Design of VLSI Circuits and Systems **Fall 2015**
Prof. Dejan Marković Class Project

Due: Thursday, Dec 3 at 4pm

ADC Non-linearity Correction Engine

Quick Overview

The project aims to test your understanding of finite-state machine, memory control, and their hardware implementation using chip synthesis. You have 6 weeks (~3 weeks to develop RTL, 2 week for energy/sample optimization, 1 week for physical mapping). Submit following files by email (ee216a@gmail.com) with "**Project submission: SID #**" in the subject line:

- **NLC.v** Your Verilog design
- **NLC.gds** Layout of your design
- **Timing-SID.txt** Post-layout timing report
- **Power-SID.txt** Post-layout power report
- **Summary-SID.pdf** 1-page summary report (template provided)

1. Introduction

Assume you designed an analog-to-digital converter (ADC) with a resolution of 21-bits but due to the implementation-inherent non-linearity you are observing an effective resolution of only 6-8 bits (ENOB). You were initially targeting an ENOB of 15bits. Instead of redesigning the ADC you can take an easy approach and correct the non-linear mapping such that you reach your target ENOB as follows.

- First invert the ADC_{count} vs V_{in} curve such that the ADC_{count} is on the X-axis and V_{in} is on the Y-axis (Figure 1).

- You now observe that you have a function, which can map the non-linear ADC_{count} output to the supplied input voltage. If you design a digital block, which accepts ADC_{count} as input and generates V_{in} at the output you will accomplish your task.

The easiest implementation that can be employed is a lookup table. You feed the ADC_{count} as your memory address and then the corrected value (which was previously stored in the memory) will appear at the output. This method will require 2^{21} memory locations and 15bits for each location to accomplish this. Assuming $0.124um^2$ for 6T SRAM cell area [1], we will need $3.9mm^2$ for each ADC channel. For a 32-channel system the required area will be $124.8mm^2$, which is prohibitively expensive.

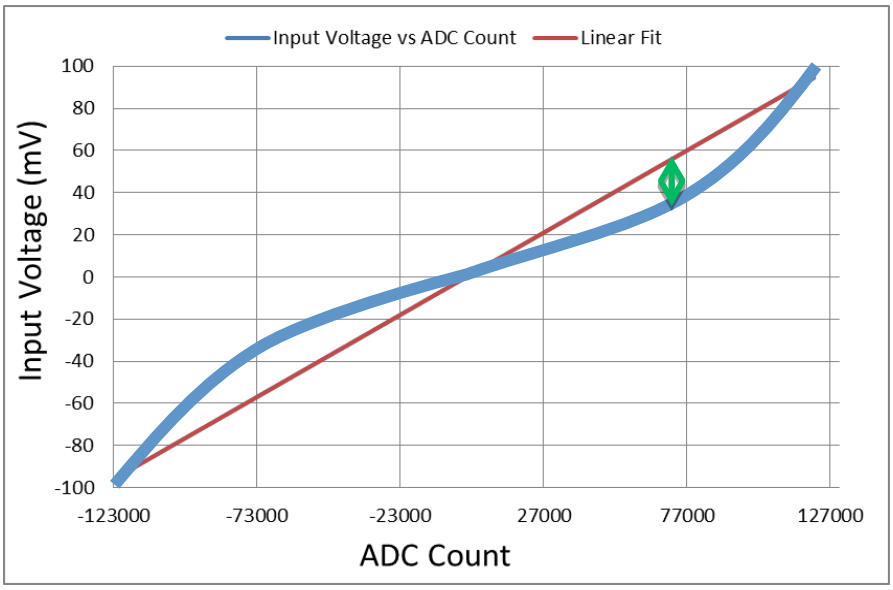


Fig. 1. Inverted Non-linearity Curve.

Another approach is to fit a non-linear function (i.e. polynomial) to the above curve and calculate the V_{in} every time we receive an ADC_{count}. For this approach, we will need to save the coefficients only - hence less memory will be required. This will reduce the leakage power (due to area savings), but since we will be doing computations for each ADC sample, the dynamic power will increase.

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

If we take a look at the general form of a polynomial above we notice that the input (X) has to be raised to the power of n, which is the polynomial order. The order of the polynomial dictates how well the non-linearity curve is matched. For example a 30^{th} order polynomial may give a good enough fit to achieve the required specification, however raising a 21-bit input into 30^{th} power will be computationally expensive to implement in direct form. Since the non-linearity is more pronounced at both extremes of the ADC range, slight reduction in the ADC range (+-80mV instead of +-100mV) can improve the fitting.

If we "chop" the entire non-linearity curve into smaller chunks and try to do piecewise polynomial fitting we can reduce the polynomial order (see the supplied polynomial_order_and_coeffs_cent_and_scale.m matlab script for a sample). We can then implement a polynomial computation engine using the worst case polynomial order among the sections and use it to calculate the polynomial fit for all sections. Note that in this case each section will have different sets of coefficients which will be supplied to the engine based on the input range. If one section's order is less than the polynomial engine order, 0 can be supplied for the higher order coefficients.

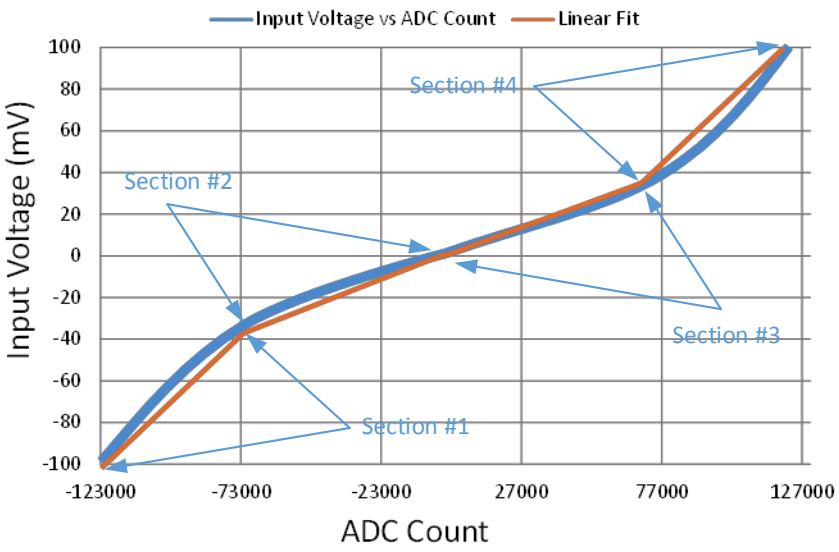


Fig. 1. Sectioning the non-linearity curve for polynomial order reduction.

Algorithmic transformation can also ease the computational burden. If we use Horner's method for iterative polynomial computation we can use one multiply-accumulate unit over many clock cycles to compute the same polynomial. Let's say that we chopped our non-linearity curve into 4 sections and our worst case section order is 5. Figure 2 illustrates the iterative implementation for this architecture.

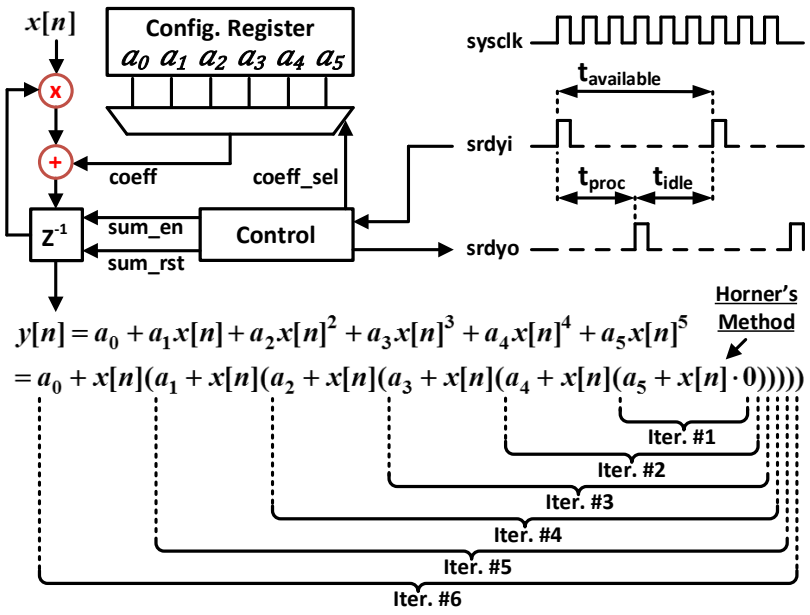


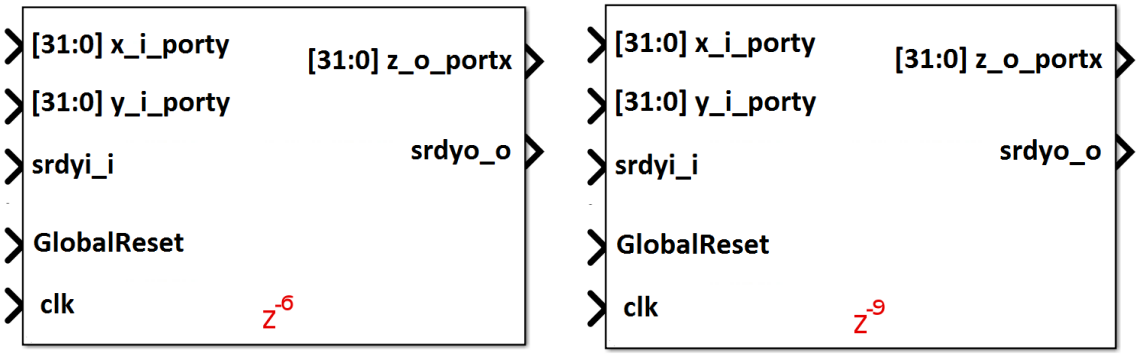
Fig. 2. Iterative implementation of polynomial non-linearity correction for one section.

Note that the system clock frequency is faster than the ADC sampling rate. This is why you will have idle system clock cycles in-between ADC samples, and can utilize them for iterative calculations. . To manage the sample flow, the engine supports forward flow control using srdyi (input enable) and srdyo (output valid) signals.

2. Will any design blocks be provided?

You will be provided with a Verilog implementation of a single precision floating point adder and a multiplier to use in your design (red blocks in figure 2). You can treat these as black boxes and instantiate them in your top module.

Note that these blocks have synchronous active-high resets (GlobalReset) and the active clock edge is the positive edge (clk). The red lettered Z^{-6} and Z^{-9} mean that the multiplier and the adder have latency of 6 and 9 respectively (pipelined).



These floating-point blocks use a floating-point representation that differs from the IEEE floating-point representation. Matlab, on the other hand, uses the IEEE floating-point representation. To feed coefficients found in matlab into your design, or to convert the output of your simulation to matlab readable format, you will need to convert the formats back and forth.

Two floating-point conversion functions are provided:

- syn_ieeefp2smcfp - Converts IEEE format floating-point numbers to SMC floating-point numbers. For example to convert IEEE representation of 0.25 to SMC (symphony model compiler) representation we will use the following command in matlab

```
smcfp = uint32(syn_ieeefp2smcfp(0.25, 8, 23))
```

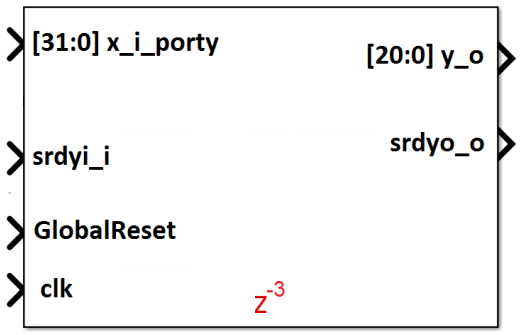
```
smcfp =

        1056964608
```

Here, the 8 and 23 are the exponent and mantissa widths. You should always use these values for the conversions.

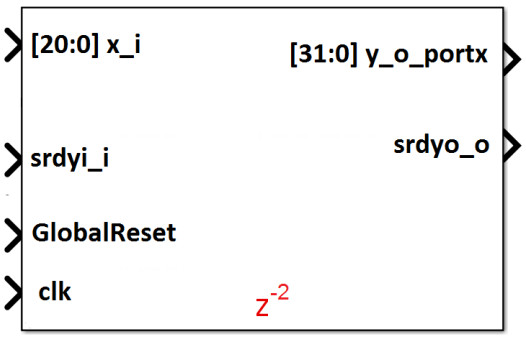
- syn_smcfp2ieeefp - Converts SMC floating-point numbers to IEEE floating-point numbers. For example to convert the SMC representation of 0.25 to IEEE we will use the following command in matlab.

```
ieeefp = syn_smcfp2ieeefp(1056964608, 8, 23)

ieeefp =

        0.2500
```

In addition to the floating point adder and multiplier you will need to convert the ADC output to SMC floating point format so that you can perform addition and multiplication operations. The "fp_to_smc_float" help to achieve this conversion. The coefficients don't need to be converted, they can be stored in the configuration memory in SMC single precision floating point format (32-bit). The result of computation will be in SMC floating point format and will need to be converted to fixed point representation at the output. The "smc_float_to_fp" block will accomplish this. Just like for the adder and multiplier blocks, smc_float_to_fp and fp_to_smc_float modules have synchronous active-high resets (GlobalReset) and the active clock edge is the positive edge (clk). These modules also are pipelined and have latencies of 3 and 2 respectively.



smc_float_to_fp



fp_to_smc_float

Table I. Provided Verilog Files.

File Name	Description
define.h	Contains definitions used by the provided design blocks.
SynLib.v	Contains structures used by the provided design blocks.
smc_float_adder.v	SMC single precision floating point adder
smc_float_multiplier.v	SMC single precision floating point multiplier.
smc_float_to_fp.v	SMC single precision floating point to fixed point converter block.
fp_to_smc_float.v	Fixed point to SMC single precision floating point converter block.

3. Will a testbench be provided?

You will be provided with a testbench to test your single channel NLC engine. Note that you are responsible for designing a 32 channel NLC system. For the first two parts of the project (Functional 4 section 10^{th} order design and Functional 3, 5, or 6 section design) you can make 32 parallel copies of a single engine design and assume that ADC data arrives to NLC block at the same clock cycle for all channels as indicated by the srdyi (input enable) signal. The test bench can be copied to your local folder from the following location.

```
/w/class.1/ee/ee216a/ee216ata/setup/proj_tb_11_11_15.tar.gz
```

Note that the testbench is provided as a tarball archive and can be extracted with the following command once copied locally.

```
tar zxvf proj_tb_11_11_15.tar.gz
```

The testbench contains the following files.

Table II. Files provided with the testbench.

File Name	Description
define.h	Contains definitions used by the provided design blocks.
SynLib.v	Contains structures used by the provided design blocks.
NLC_4sec_10th_order_1ch_v0_Test.v	The Verilog testbench
NLC_4sec_10th_order_1ch_v0.v	Wrapper file for your module. This module is instantiated inside the testbench as UUT. Since by now you have your design which will have different signal names than what is used in this testbench an easy way to "couple" your module to the testbench is to instantiate your design inside this wrapper module and make the

	appropriate port connections. You will see an example on how to do this inside the wrapper.
NLC_ref_output.mat	Contains matlab workspace variables v_uV and max_error. If the testbench fails your module verification you can subtract your NLC output from v_uV and see whether the maximum value element of the resultant vector is < 2*max_error.
*.dat	Test vector data files.

After simulation the testbench will generate the following files.

Table III. Files generated by the testbench.

File Name	Description
simlog.txt	Contains test results (PASS, FAIL)
Simout_NLC_4sec_10th_order_1ch_v0_x_lin.dat	Value of the linearized x (x_lin). This can be imported in matlab and compared to v_uV ideal output curve. The max error should be less than 2*max_error (from NLC_ref_output.mat file).
Simout_NLC_4sec_10th_order_1ch_v0_srdyo.dat	Output ready signal (srdyo). This should be imported into matlab along with the linearized x output. It will indicate when the output data is valid.

4. Design Specifications

Table II lists the system design specifications. You will notice that the given non-linearity curve exercises only 17bits of the raw ADC resolution not 21 bits. This is due to the sampling frequency you are given (6kHz), for lower sampling frequencies the ADC raw resolution will be higher and we will be designing the NLC engine assuming 21bit input although your analysis is for ~17bit input. Notice also that even the required ENOB is 14bits, you are asked to design the NLC engine with 21bit output resolution. Besides the previous argument about reduced sampling rates, due to simulation mismatches to actual hardware performance we want to keep a margin so that if the hardware performance is better than the simulation results we can attain higher ENOB than designed for.

Table IV. System Design Specifications.

Design Parameter	Value
Number of ADC Channels	32
ADC Raw Resolution	21 bits
Effective Resolution (ENOB) (after your correction)	14 bits

Output Resolution	21 bits
ADC Sampling Rate	6 kHz
System Clock Frequency	6.144 MHz

5. Design Metrics

The design objective is to minimize the energy per ADC sample (i.e. pJ/sample). Please minimize the energy while maintaining the system throughput. We use the metric of **Efficiency = Chip Power / ADC Sampling Rate** to evaluate the performance. You will need to form a group with another classmate to complete this project. Consultation with others is allowed, but the work has to be distinctly yours.

6. Suggested Timeline

The project will span five weeks. You will need roughly 3 weeks to develop RTL, one week for speed-area optimization, and one week for physical mapping.

7. Project Submission

Submit following files by email (ee216a@gmail.com) with "**Project submission: SID number**" in the subject line:

- **NLC.v** Your Verilog design
- **NLC.gds** Layout of your design
- **Timing-SID.txt** Post-layout timing report
- **Power-SID.txt** Post-layout power report
- **Summary-SID.pdf** 1-page summary report (PPT template provided)
  Be sure to include your SID as part of file name

8. Grading

Your project will be graded based on following criteria (groups of 2 or more):

Functional 4 section 10th order design: 50%

Functional 3, 5, or 6 section design: 20%

Comparing the power for SRAM, Flip-flop, and Register File as storage: 5%

Efficiency metric: 25% (automated grading)

If you don't have a project partner and working on it alone, you will be graded based on the following criteria.

Functional 4 section 10th order design: 60%

Functional 3, 5, or 6 section design: 30%

Efficiency metric: 10% (automated grading)

The efficiency points will be added only if you have complete functionality of the 4 section 10th order design. You can also apply the following algorithmic, architectural transformation and circuit optimization techniques for extra credit.

Channel Interleaving: (levels 2, 4, 8, 16, & 32)	5%
Voltage Scaling:	5%
Power Gating:	5%
Configuration Memory Blocking:	5%
Fixed-point implementation, wordlength optimization:	10%
Try a different algorithm for non-linearity correction:	20%

HAVE FUN!

References

[1] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini, and W. Haensch, "Stable SRAM cell design for the 32 nm node and beyond," in *2005 Symposium on VLSI Technology, 2005. Digest of Technical Papers*, 2005, pp. 128–129.