EEM216A Design of VLSI Circuits and Systems **Fall 2016**
Prof. Dejan Marković Class Project

Due: Thursday, Dec 1 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, 1 week for functional verification, 2 weeks for synthesis and energy/sample optimization). Submit following files by email (ee216a@gmail.com) with "**Project submission: SID #**" in the subject line:

- **NLC.v** Your Behavioral Design
- **NLC.vg** Your Gate-Level Design (Post-Synthesis)
- **Timing-SID.txt** Post-Synthesis timing report (report_timing setuptime)
- **Power-SID.txt** Post-Synthesis power report (report_power)
- **Summary-SID.pdf** 1-page summary report (template provided)

1. Introduction

Assume you have an analog-to-digital converter (ADC). This ADC will act as the front end of your low-power design (every modern design is low-power), and in your design you need the ADC with 15-bit resolution (this number is NEVER arbitrary, and it is often determined by the minimum number needed by the systems folks to achieve a certain performance). The ADC you have has a resolution of 21 bits (these are your "marketing bits"), but you typically lose a few of bits due to nonlinearities and noise performance of the ADC. When you measure the effective resolution of this ADC, you find that its linear range could be as small as 6 to 8 bits (these are your "true bits"). By sweeping the voltage input to the ADC you get the output voltage sweep and realize that the reason for the significant degradation of bit resolution is the implementation-inherent non-linearity of the ADC. Assuming the design of the ADC is finalized and can't be changed, you have to recover bits in the digital domain. You were initially targeting an ENOB (effective number of bits or "true bits") of 14 bits. Instead of redesigning the ADC you can take an easy approach and correct the non-linear mapping such to reach your target ENOB as below.

As an aside, an inherent attribute of designs in the sub-micron regime (e.g. 65nm, 40nm, 22nm, etc.) is increasing nonlinearity. This isn't usually a large problem for digital design given the idiot-proofing built into digital design, but is a big problem for analog designs. In fact, nonlinearity is an increasing problem with decreasing technology node. Often, this nonlinearity is corrected in the digital back-end. This problem of non-linear correction (NLC) is an exceptionally timely and practical problem for all of you to tackle, so here it goes.

2. Correcting for ADC Non-Linearity

Begin by sweeping the input voltage of the ADC, observe the ADC output and graph V_{in} vs ADC_{count}.

- 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 is a lookup table. You feed the ADC_{count} as your memory address and then the corrected value (which was previously stored in memory) will appear at the output. This method requires 2^{21} words and 15 bits for each word. Assuming an area of $0.124um^2$ for an 6T SRAM cell [1], you need $3.9mm^2$ for each ADC channel. For a 16-channel system the required area will be $62.4mm^2$, which is prohibitively expensive. Game over!

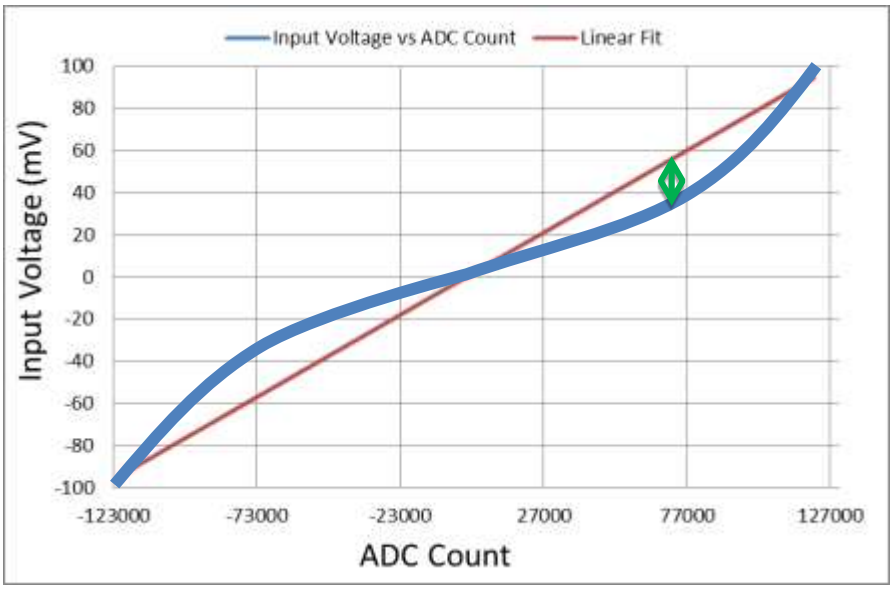


Fig. 1. Inverted Non-linearity Curve.

Another approach is to fit a non-linear function (i.e. polynomial) to the above curve and calculate the V_{in} for each ADC_{count}. For this approach, we will need to save the coefficients only - hence less memory will be required. Sounds promising… This also reduces leakage power (due to lower area), but since we will be doing computations for each ADC sample, the dynamic power will increase. OK, let's examine this tradeoff more closely and look at the polynomial we need to implement:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Looking 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 30th-order polynomial may give a "good enough" fit to achieve the required specification, however raising a 21-bit input into 30th power will be computationally expensive 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 sections and do piecewise polynomial fitting we can reduce the polynomial order (see **polynomial_order_and_coeffs_cent_and_scale.m** Matlab script as a reference). We can then implement a polynomial computation engine using the worst-case polynomial order across all sections and use it to calculate the polynomial fit. Note that in this case each section will have different sets of coefficients 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.

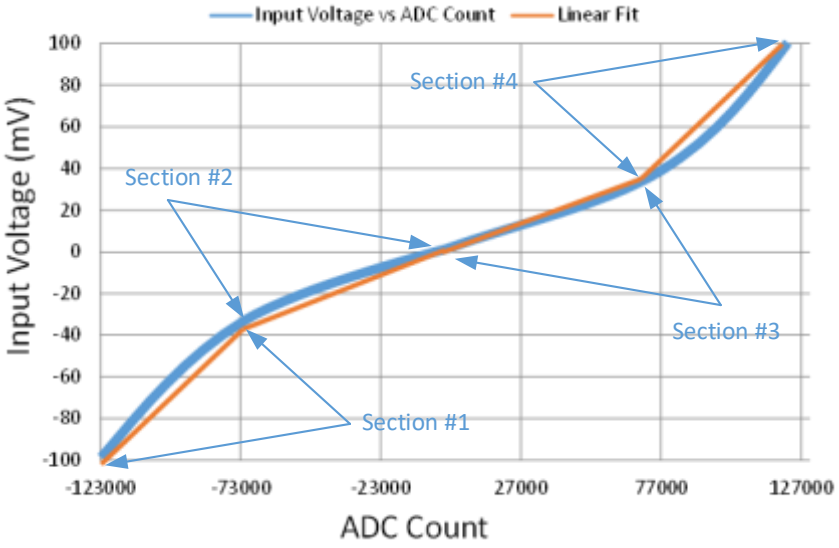


Fig. 2. Piece-wise linear model of the non-linearity curve for polynomial order reduction.

Algorithmic transformation can also ease the computational burden a lot more than hardware implementation, so let's work out algorithm details first. 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 divided our non-linearity curve into 4 sections and our worst-case section order is 5. Figure 3 illustrates the iterative implementation for this architecture.

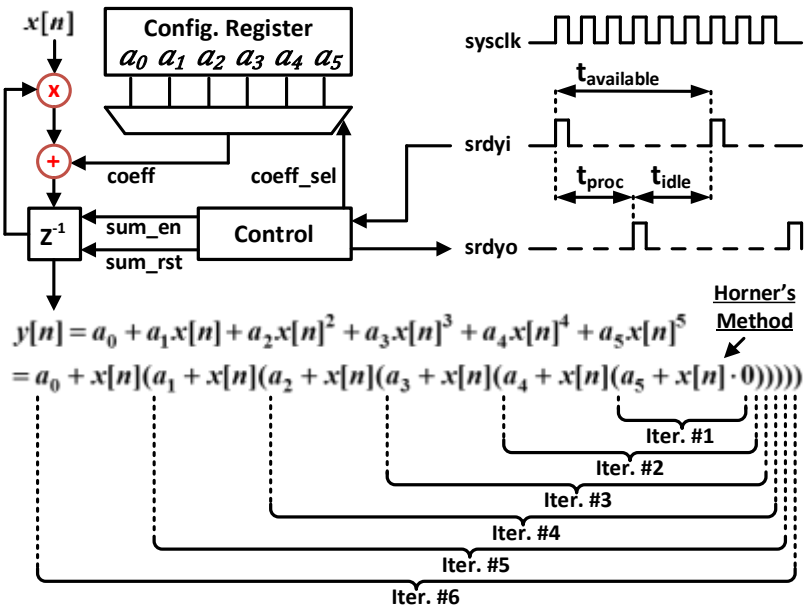


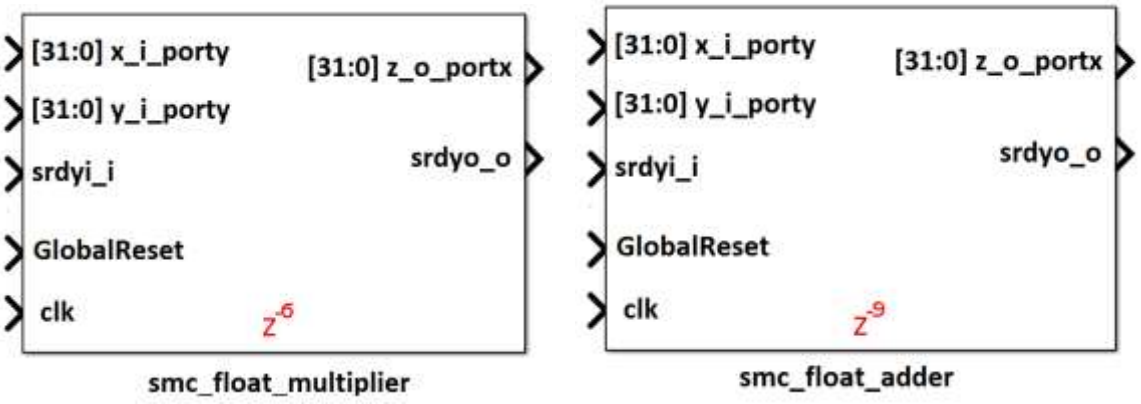
Fig. 3. Iterative implementation of polynomial non-linearity correction for one section.

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 data flow, the engine supports forward flow control using srdyi (input enable) and srdyo (output valid) signals.

3. 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 3). These blocks are your "designware." You can treat these as "black boxes" and instantiate them in your top module.

The provided floating-point arithmetic blocks do not conform to the IEEE-Standard floating-point description. You should also note that the blocks have synchronous active-high reset (GlobalReset) and the active clock (clk) edge is the positive edge. The red-lettered Z^{-6} and Z^{-9} indicate that the multiplier and the adder have latency of 6 and 9 clock cycles, respectively (they are 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. This is extra work, but much less than having to design a floating-point unit, and you learn some cool practical tricks.

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 (Synphony 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"** helps to achieve this conversion. The coefficients don't need to be converted, they can be stored in the configuration memory in SMC single-precision (32-bit) floating-point format. The result 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 active-high synchronous reset (GlobalReset) and are triggered by the positive clock (clk) edge. These modules also are pipelined and have latencies of 3 and 2, respectively.

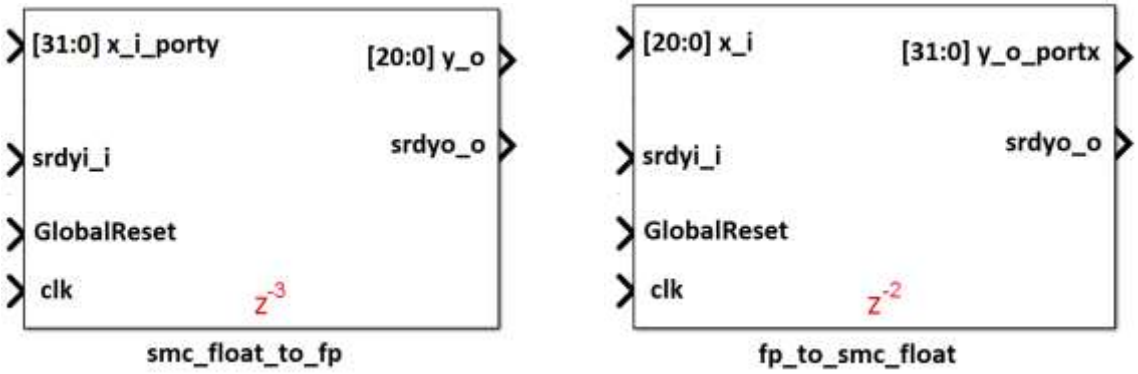


Table I. Provided Verilog Files.

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.

4. Testbench

You will be provided with a testbench to test your single-channel NLC engine. Note that you are responsible for designing a 16-channel NLC system. For the first two parts of the project (functional 4-section 8th-order design, channel interleaving for 8 channels), you can make 16 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.

Also you will be supplied the coefficients for 8th-order polynomial (Xoriginal), the (-mean) and (1/standard deviation(STD)) since we are using the centered and scaled curve, your new X is going to be different. Xnew=Xoriginal−Mean(STD). So you need to do this operation on the coefficients before using them. The new operation will be {Xnew=(Xoriginal+(−Mean))*1/STD.}

5. Design Specifications

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

Table II. System Design Specifications.

Design Parameter	Value
Number of ADC Channels	16
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

6. 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.

7. Suggested Timeline

The project will span six weeks. You will need roughly 3 weeks to develop RTL, one week for functional verification, and two weeks for synthesis and optimizations.

8. Project Submission

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

- **NLC.v** Your Behavioral Design
- **NLC.vg** Your Gate-Level Design
- **Timing-SID.txt** Post-Synthesis timing report (report_timing setuptime)
- **Power-SID.txt** Post-Synthesis power report (report_power)
- **Summary-SID.pdf** 1-page summary report (PPT template provided)

Be sure to include your SID as part of file name

9. Grading

Your project will be graded based on following criteria:

Groups of 2 or more:

Functional 4 section 8th order design:		60%
Functional 4 section 8th order design 8 interleaved channels:		20%
Efficiency metric:	20%	(automated grading)

Single-person projects:

Functional 4 section 8th order design:		60%
Functional 4 section 8th order design 8 interleaved channels:		30%
Efficiency metric:	10%	(automated grading)

The efficiency points will be added only if you have complete functionality of the 4-section 8th-order design. You can also apply the following algorithmic, architectural transformation and circuit optimization techniques for extra credit.

Fixed-point implementation, wordlength optimization:	10%
Try a different algorithm for non-linearity correction:	20%
Log-domain implementation, wordlength optimization	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.