ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A 16-b 32 MSPS CMOS VOLTAGE OUTPUT DAC in 0.18 um WITH 80+ dB SIMULATED SFDR at 1 MHz OUTPUT FREQUENCY

M.Sc. THESIS

Çağlar ÖZDAĞ

Department of Electronics and Communications Engineering

Electronics Engineering Programme

JANUARY 2015

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A 16-b 32 MSPS CMOS VOLTAGE OUTPUT DAC
in 0.18 um WITH 80+ dB SIMULATED SFDR
at 1 MHz OUTPUT FREQUENCY

M.Sc. THESIS

Çağlar ÖZDAĞ
504111202

Department of Electronics and Communications Engineering

Electronics Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Türker KÜYEL

JANUARY 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

1 MHz ÇIKIŞ FREKANSINDA 80+ dB SFDR BAŞARIMI ELDE EDEN 0.18 um 16-b 32 MSPS CMOS GERİLİM ÇIKIŞLI SAYISAL-ANALOG ÇEVİRİCİ TASARIMI

YÜKSEK LİSANS TEZİ

Çağlar Özdağ
504111202

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Dr. Türker KÜYEL

OCAK 2015

Çağlar Özdağ, a **M.Sc.** student of ITU **Graduate School of Science Engineering and Technology** student ID **504111202**, successfully defended the **thesis** entitled "**A 16-b 32 MSPS CMOS VOLTAGE OUTPUT DAC in 0.18 um WITH 80+ dB SIMULATED SFDR at 1 MHz OUTPUT FREQUENCY**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Türker KÜYEL** İstanbul Technical University	....
Jury Members :	**Prof. Dr. Ece Olcay GÜNEŞ** İstanbul Technical University	....
	Prof. Dr. Günhan DÜNDAR Bogazici University	....

Date of Submission : 15 December 2014
Date of Defense : 6 January 2015

For science,

FOREWORD

I would like to express my deepest gratitude to my thesis advisor and mentor Dr. Türker KÜYEL for being a rich source of knowledge and motivation throughout my graduate eduction. Many thanks to my dear friends and teachers at ITU VLSI Labs for creating an environment where research can flourish. And finally, with my warmest affection, I dedicate this work to my parents Ufuk and Osman for their constant support, love and patience.

January 2015 Çağlar ÖZDAĞ

TABLE OF CONTENTS

Page

FOREWORD ix
TABLE OF CONTENTS xi
ABBREVIATIONS xiii
LIST OF TABLES xv
LIST OF FIGURES xvii
SUMMARY xix
ÖZET xxi
1. INTRODUCTION 1
  1.1 DAC Architectures 1
  1.2 Resistor String Type VODAC Architectures 2
  1.3 Design Summary 9
  1.4 Design Environment 10
  1.5 Thesis Organization 14
2. ERROR MECHANISMS 15
  2.1 Static Error Mechanisms and DC Error Correction 15
  2.2 Dynamic Error Mechanisms 17
    2.2.1 RC model of DACs and method 18
    2.2.2 Code dependent equivalent resistance 20
    2.2.3 Code dependent equivalent capacitance 21
    2.2.4 Code dependent charge injection 21
    2.2.5 Code dependent data feedthrough 22
    2.2.6 Glitches 23
    2.2.7 Interpolator non-linearity 23
    2.2.8 Signal-to-noise ratio (SNR) 23
3. DESIGN DECISIONS AND SCHEMATIC 25
  3.1 Architecture Overview 25
  3.2 String 26
  3.3 Switches and Resistive Calibration 32
  3.4 Interpolating OPAMP, Velocity Saturation and Capacitive Calibration 37
  3.5 Pre-charge Stage and LSB Sensitivity 42
  3.6 Instrumentation Amplifier 45
  3.7 Voltage References 46
  3.8 Routine Simulations 48
    3.8.1 Temperature sweep 48
    3.8.2 Input frequency sweep 49
    3.8.3 Output range sweep 50
    3.8.4 INL/DNL 51
    3.8.5 Mismatch 53
    3.8.6 Schematic corners 54

4. LAYOUT **55**
4.1 Floorplan 55
4.2 String and 1st Switch Bank 55
4.3 2nd Switch Bank 59
4.4 3rd Switch Bank 61
4.5 Interpolating OPAMP 63
4.6 Complete DAC Core Layout 65
4.7 Instrumentation Amplifier Configuration 66
4.8 Bandgap Reference 67
4.9 Decoder 68
4.10 Extracted Results 69
5. CONCLUSION **71**
REFERENCES **73**
APPENDICES **74**
APPENDIX A: Decoder Verilog Code 77
APPENDIX B: Ideal ADC-DAC Testbench VerilogAMS Code 89
APPENDIX C: Calibration Resistor Generating Script 93
CURRICULUM VITAE **97**

ABBREVIATIONS

ADC : Analog-to-Digital Converter
BBM : Break Before Make
Cgd : Gate-Drain Capacitance
Cgdovlp : Gate-Drain Overlap Capacitance
Cgs : Gate-Source Capacitance
Cgsovlp : Gate-Source Overlap Capacitance
CMOS : Complementary Metal Oxide Semiconductor
DAC : Digital-to-Analog Converter
DRC : Design Rule Check
DFT : Discrete Fourier Transform
DNL : Differential Non-linearity
IC : Integrated Circuit
INL : Integral Non-linearity
LSB : Least Significant Bit
LUT : Look-up Table
LVDS : Low-voltage Differential Signaling
LVS : Layout vs. Schematic
MBB : Make Before Break
MCU : Microcontroller
MSB : Most Significant Bit
MSPS : Mega Samples Per Second
OPAMP : Operational Amplifier
ppm : Parts per million
PVT : Process, Voltage, Temperature
RC : Resistor-Capacitor
Ron : On Resistance
SFDR : Spurrious-free Dynamic Range
SNR : Signal-to-noise Ratio
Vgs : Gate-source Voltage
Vbs : Bulk-source Voltage
VODAC : Voltage Output Digital-to-analog Converter

LIST OF TABLES

Page

Table 1.1 : SFDR of basic r-string DAC architectures 8
Table 3.1 : Fully differential vs. single ended string.. 26
Table 3.2 : String resistive compensation 29
Table 3.3 : Switching order and its effect on SFDR 31
Table 3.4 : Effect of compensation resistors on SFDR.. 35
Table 3.5 : DAC reference voltage ranges for constant V_{GS} switching 36
Table 3.6 : Effect of constant V_{GS} switching on SFDR 36
Table 3.7 : Effect of data feedthrough on SFDR.. 36
Table 3.8 : Short channel effects on SFDR.. 40
Table 3.9 : Effect of dummy output stages on SFDR 42
Table 3.10 : Effect of pre-charge on SFDR 45
Table 3.11 : Instrumentation amplifier configuration 46
Table 3.12 : Real references.. 47
Table 3.13 : Schematic corners 54
Table 4.1 : Single ended vs differential output (layout)....................................... 69
Table 4.2 : Layout corners... 69

LIST OF FIGURES

Page

Figure 1.1 : Data converters 1
Figure 1.2 : Current-mode DACs 1
Figure 1.3 : Voltage-mode DACs 2
Figure 1.4 : 10-bit resistor string DAC 3
Figure 1.5 : Interpolating amplifier 4
Figure 1.6 : Interpolating amplifier DAC 4
Figure 1.7 : Dual output resistor string DAC architecture 5
Figure 1.8 : 16-bit low bandwidth DAC 6
Figure 1.9 : Coarse-fine intermeshed 10-bit string 7
Figure 1.10 : 16-bit DAC 8
Figure 1.11 : Fully differential 16-bit DAC 9
Figure 1.12 : SFDR vs. output frequency 10
Figure 1.13 : Simulation environment (schematic editor) 11
Figure 1.14 : Simulation environment (hierarchy editor) 12
Figure 1.15 : Transient output waveform of the layout extracted design 13
Figure 1.16 : DFT of layout extracted design (84 dB SFDR) 13
Figure 2.1 : Transfer function of an ideal 3-bit DAC 15
Figure 2.2 : DC (static) errors of DACs 16
Figure 2.3 : DC calibration setup 16
Figure 2.4 : DC calibration algorithm 16
Figure 2.5 : Ideal DAC model 17
Figure 2.6 : Basic RC model of our DAC 18
Figure 2.7 : Basic RC model of our DAC (with approximate values) 19
Figure 2.8 : Settling characteristic at the input of the interpolating OPAMP 19
Figure 2.9 : Detailed RC model 20
Figure 2.10 : Charge injection 22
Figure 3.1 : Top level block diagram 25
Figure 3.2 : XY decoded cells 26
Figure 3.3 : Input range for low current 16 bit DAC 27
Figure 3.4 : String node equivalent resistance model 28
Figure 3.5 : String equivalent resistance per code 28
Figure 3.6 : String compensation resistors per code 29
Figure 3.7 : a) No special switching, b) Make before break, c) output glitch 30
Figure 3.8 : a) No special switching, b) Break before break, c) output glitch 30
Figure 3.9 : SFDR vs. string resistance (1) 31
Figure 3.10 : SW3 on-resistance variation 32
Figure 3.11 : SW1 and SW2 on-resistance variation 33
Figure 3.12 : Compensation resistors for SW1 and SW2 34
Figure 3.13 : Compensation resistors for string, SW1 and SW2 34
Figure 3.14 : Constant V_{GS} switch cell basic architecture 35

Figure 3.15 : SFDR vs. string resistance (2) 37
Figure 3.16 : Interpolating OPAMP 38
Figure 3.17 : Interpolating OPAMP currents 38
Figure 3.18 : Interpolating OPAMP frequency response 39
Figure 3.19 : Velocity saturation in short channel devices (I_D vs. V_{DS}) 40
Figure 3.20 : Velocity saturation in short channel devices (I_D vs. V_{GS}) 40
Figure 3.21 : Switch bank 3 41
Figure 3.22 : Dummy interpolating OPAMPs and SW3 41
Figure 3.23 : Input to the second switch bank (no pre-charge) 43
Figure 3.24 : Input to the second switch bank (no pre-charge) zoomed in 43
Figure 3.25 : Pre-charge switch placement 44
Figure 3.26 : Input to the second switch bank (with pre-charge) 44
Figure 3.27 : Input to the second switch bank (with pre-charge) zoomed in 45
Figure 3.28 : Intrumentation amplifier configuration 46
Figure 3.29 : Brokaw band-gap reference 47
Figure 3.30 : SFDR vs. temperature 48
Figure 3.31 : SFDR vs. input frequency 49
Figure 3.32 : SFDR vs. output magnitude (nom: 2 V_{pp}) 50
Figure 3.33 : SFDR vs. input common mode (nom: 0.825 V) 50
Figure 3.34 : 16-bit INL 51
Figure 3.35 : 16-bit DNL 52
Figure 3.36 : Element mismatch SFDR histogram 53
Figure 4.1 : Symbolic floorplan 55
Figure 4.2 : Resistor string w/ compensation resistors and SW1 56
Figure 4.3 : Resistor string w/ compensation resistors and SW1 zoom 1 57
Figure 4.4 : Resistor string w/ compensation resistors and SW1 zoom 2 57
Figure 4.5 : Resistor string w/ compensation resistors and SW1 zoom 3 58
Figure 4.6 : SW1 59
Figure 4.7 : SW2 (one cell) 59
Figure 4.8 : SW2 a) whole block b) zoom 1 c) zoom 2 60
Figure 4.9 : SW3 whole block zoomed 61
Figure 4.10 : SW2 and SW3 whole block zoomed 62
Figure 4.11 : Interpolating OPAMP 63
Figure 4.12 : Differential pair, SW3 (middle) and SW2 (left most) 63
Figure 4.13 : Interpolating OPAMPs (rightmost), diff pair, SW3 and SW2 64
Figure 4.14 : Interpolating OPAMPs, diff pair, SW3 and SW2, string and SW1. 65
Figure 4.15 : Instrumentation amplifier configuration DAC core 66
Figure 4.16 : Brokaw bandgap reference 67
Figure 4.17 : Digitally programmable resistor string 67
Figure 4.18 : Decoder 68
Figure 4.19 : Final layout SFDR vs. output frequency 70
Figure 5.1 : SFDR vs. input frequency summary 71

A 16-b 32 MSPS CMOS VOLTAGE OUTPUT DAC IN 0.18 um WITH 80+ dB SIMULATED SFDR AT 1 MHz OUTPUT FREQUENCY

SUMMARY

Interfacing digital domain signals to an analog control or transfer system requires an integrated circuit (IC) element referred to as a digital-to-analog converter (DAC). Achieving high precision and high dynamic linearity at high sampling speeds and high output frequencies is an ever on-going research challenge due to the complexity of interconnected tradeoffs involved in the performance of such architectures. Due to the nature of these performance tradeoffs, certain architectures are used for certain applications which prioritize 6 main parameters: physical size, power consumption, resolution, bandwidth, precision/sensitivity and cost. Most DAC architectures used in all kinds of communications, data acquisition, signal processing, and control systems can be categorized into one of two families (i) Current-mode type architectures that offer high speed at the cost of monotonicity, drift sensitivity and precision settling; (ii) Voltage-mode type architectures that address the precision settling problem but have its shortcomings in speed and resolution. This work addresses the unmet need for a precision settling, high speed and high bitrate DAC architecture by taking the standard resistor-string type buffered voltage output architecture and greatly improving its dynamic linearity for driving time-varying loads at high output frequencies.

Typical operation of a resistor string-type DAC involves selecting nodes on a resistor string with a certain switching architecture dictated by the input decoder, and driving the output load through a voltage buffer. The resistor string sits between two voltage references and divides the full scale input into equal steps. DC performance of such converters is determined by the precision of the voltage references and more importantly the matching of the elements on the resistor string. These so-called static nonlinearities can be digitally calibrated to give 16-bit accuracy at low bandwidth; but error mechanisms that affect the dynamic linearity at high output frequencies remain mostly unsolved.

The most fundamental dynamic performance metric of DACs is the spurious-free dynamic range (SFDR) of the output waveform. SFDR is the ratio of the root-mean-squre signal amplitude to the highest spurious component in the first Nyquist zone and is closely related to total harmonic distortion (THD) and intermodulation distortion (IMD), thus a good measure of dynamic linearity. In this work, most simulation results are presented in reference to the SFDR of the full-scale output waveform. What is considered to be the current state-of-the-art 16-bit voltage output DAC (TI-DAC8580) gives 63 dB SFDR for a 200 kHz, which is the highest frequency listed on specification. The architecture presented in this work surpasses this performance by a great amount, giving a layout extracted 83 dB SFDR for a 1 MHz signal.

Six main dynamic error mechanisms were identified and compensated to achieve this performance. Code-dependent interpolating amplifier input capacitance is compensated by the inclusion of a dummy interpolating amplifier and dummy differential pair switch structure. Code-dependent resistor string equivalent resistance and code dependent V_{gs} and V_{bs} varying switch bank on-resistance are compensated with the inclusion of tap point calibration resistors. Charge injection and related glitches on the output bus are reduced by a unique fully differential resistor string and differential interpolating instrumentation amplifier architecture. Interpolating amplifier output stage nonlinearity is reduced by driving the class AB output stage transistors at their velocity saturation region. LSB sensitivity to floating resistor string loops are reduced by implementing a loop pre-charge stage on the fully differential resistor string. Aside from the development of such novel architectures, other specifics of all stages on schematic, as well as on layout, are optimized to reduce distortion by keeping the output bus settling characteristic fast and code independent.

Simulation environment is chosen to be Spectre+AMS running on Cadence 6.02 evaluating BSIM4 models of the TSMC 018 um process. The unmodified standard architecture which was the starting point of this work has a 60 dB schematic level SFDR for a 1 MHz (f_0), 32 MHz (f_s), 2 V_{pp} output signal. The final design has a 88 dB schematic level SFDR, 83 dB layout level SFDR under the same conditions, tested under process corner, temperature range and supply drift variations.

Tape out is expected to be April 2015.

1 MHz ÇIKIŞ FREKANSINDA 80+ dB SFDR BAŞARIMI ELDE EDEN 0.18um 16-b 32 MSPS CMOS GERİLİM ÇIKIŞLI SAYISAL-ANALOG ÇEVİRİCİ TASARIMI

ÖZET

Sayısal işaretlerin, analog kontrol ve transfer sistemlerine arabağlanması, sayısal-analog-çevirici (DAC) olarak isimlendirilmiş entegre devreler (IC) ile gerçekleştirilir. Bu elektronik mimariler ile, yüksek örnekleme frekanslarında, yüksek hassasiyet ve yüksek dinamik doğrusallık elde edebilmek, tüm ilgili başarım ödünleşimlerinin karmaşık bir şekilde bağlantılı olması nedeniyle, her zaman süregiden bir araştırma alanıdır.

Başarım ödünleşimlerinin doğası gereği, belli DAC mimarileri belli uygulamalar için kullanılır ve bu uygulamalar altı ana parametreyi önceliklendirir: fiziksel boyut, güç tüketimi, çözünürlük, bant genişliği, duyarlık ve maliyet. Her türlü iletişim, veri toplama, işaret işleme ve kontrol sistemlerinde kullanılan DAC mimarileri, iki aile şeklinde sınıflandırılabilir: (i) monotonluk, sürüklenme duyarlığı ve yerleşme duyarlığı pahasına yüksek hız sağlayan akım-mod tip mimariler ve (ii) yüksek hız ve çözünürlük pahasına yerleşme duyarlığı sağlayan gerilim-mod tip mimariler.

Zamanla değişen yükleri doğrusal bir yerleşme karakteristiği ile sürebilmek için DAC'ların çıkışlarında bir tampon katı olmalıdır. Akım-mod tip mimarilerde bu tampon katı, akımdan gerilime dönüştürücü olarak işlev görür. Dönüşümü yapmak için kullanılan geri-besleme direncinin gerilim ve sıcaklık ile sürüklenmesi, bu tip mimarilerin hassas yerleşme niteliğini kısıtlar. Akım-mod DAC çıkış katı direncinin silikon özerinde imal edildiği ve bu sorunun kısmi olarak giderildiği tasarımlar mevcut olsa da, bu mimarilerin de gliç (zamanlama hatalası) problemleri vardır. Bu nedenle yerleşme karakteristiğinin doğrusal olması gerektiği hassas dalga üretimi uygulamalarında, tercih edilmezler.

Bu çalışma, daha karşılanmamış bir gereksinim olan, hassas yerleşme, yüksek hız ve yüksek çözünürlük sağlayan bir DAC mimarisi önerisidir. Tasarım, standart direnç-dizesi tip tamponlu gerilim çıkışlı DAC mimarisini baz almakta ve zamanla değişen yükleri yüksek çıkış frekansları için sürebilmek adına dinamik doğrusallık başarımını geniş ölçüde geliştirmektedir.

Direnç-dizesi tip DAC mimarilerinin çalışma prensibi, bir direnç dizesi üzerindeki düğümlerin, giriş kod çözücüsü tarafından kontrol edilen bir anahtarlama şeması ile seçilmesi, ve bu düğüm üzerindeki gerilimin çıkış tamponu tarafından sürülmesi üzerine kurulmuştur. Bu direnç-dizesi, anahtarlar ve çıkış tamponunun çalışma aralığı tarafından belirlenen iki gerilim referansı arasında, tam ölçek çıkış aralığını eşit parçalara böler. Bu tür veri dönüştürücülerinin DC başarımını, gerilim referanslarının hassasiyeti, ve daha önemlisi, direnç-dizesi elemanlarının uyuşması

belirler. Statik doğrusalsızlıklar olarak adlandırılan bu sorunlar, düşük bant genişliklerinde tipik olarak 10 bitlik, yüksek maliyetli proseslerde lazer kırpma gibi özel teknikler kullanılarsa 12 bitlik doğruluk verebilecek derecede sayısal olarak kalibre edilebilir. Bu tekniklerden, off-chip taramalı-tablo (look-up-table) DC kalibrasyonu olarak adlandırılan yöntem, düşük maliyetli DC kalibrasyonlar arasında standart uygulama haline gelmiştir ve bu tasarım için tape-out sonrası kullanılacağı varsayılmıştır. Bu yöntemle alınabilecek çözünürlük sınırlı olduğundan, 16 bit seviyesinde doğruluk alabilmek için, ikinci bir DAC katı olarak ara-değer-bulan OPAMP (interpolating OPAMP) gibi mimariler kullanılabilir.

Devreyi hızlandırmak adına, çıkış katı zaman sabitini asgariye indirmek için düşük eşdeğer dirençli direnç-dizesi mimarilerinin de kullanımıyla, piyasadaki mevcut en iyi performans veren DAC tasarımlarına yakın benzetim sonuçları alınabilmektedir. Fakat, yüksek bant genişliklerinde dinamik doğrusallığı etkileyen hata mekanizmaları çoğunlukla çözülememiş durumdadır. Bu eksiklik, gerilim çıkışlı DAC mimarilerinin yüksek hızlı hassas dalga üretimi uygulamalarında kullanılmasını kısıtlamaktadır.

DAC mimarilerinin en temel dinamik başarım ölçüsü, çıkış dalgasının spüriyözsüz dinamik aralığıdır (SFDR). SFDR, işaret genliğinin kare ortalamalarının kökünün (rms), ilk Nyquist bölgesindeki en yüksek spüriyöz bileşenine oranıdır, ve toplam harmonik bozulma (THD) ve intermodulasyon distorsiyon (IMD) ile olan yakın ilişkisinden dolayı iyi bir dinamik doğrusallık göstergesidir. Bu çalışmada sunulan benzetim sonuçlarının çoğu, tam ölçek çıkış dalgasının SFDR'ı üzerinden incelenmiştir.

Şu an piyasada state-of-the-art kabul edilen 16-bitlik gerilim çıkışlı DAC (TI-DAC8580), spesifikasyonunda listelenen en yüksek çıkış frekansında (200 kHz) 63 dB SFDR vermektedir. Bu çalışmada önerilen DAC mimarisi bu çıtayı büyük bir fark ile aşarak, 1 MHz'lik bir işaret için serim-sonrası (post-layout) 83 dB SFDR vermektedir. Bu başarımı elde etmek için, altı ana dinamik hata mekanizması belirlenmiş ve kompanse edilmiştir.

Mimarinin çıkış katı, son altı biti (6 LSBs) sayısaldan analoga çevirme işlemini gerçekleştirmektedir. Bu işlem sırasında, mimarinin doğası gereği ara-değer-bulan OPAMP giriş kapasitesi koda-bağımlı olduğundan, çıkış hattı üzerinde görülen zaman sabiti her kod için değişmektedir. Bu idealsizlik, (i) dummy ara-değer-bulan OPAMP giriş katları ve dummy diferansiyel ikilisi anahtarları ile kompanse edilmiştir.

Çıkış hattı üzerinde görülen zaman sabitinin kapasite bileşeni böylece zamandan bağamsız hale getirilmiştir. Giriş işaretinin ilk 10 bitini (10 MSBs) sayısaldan analoga çeviren blok, bir kaç kattan oluşan bir direnç-dizisi ve anahtarlama ağı olarak düşünülebilir. Çıkış hattı üzerinde görülen zaman sabitini her kod için değişik kılan bileşen bu katlar için dirençseldir. Direnç-dizisinin koda-bağımlı eşdeğer direnci ve koda-bağımlı kapalı anahtar direnci her boğum noktası için hesaplanmış ve benzetim ortamında ölçülmüştür. Ortaya çıkan direnç profili kullanılarak, (ii) koda-bağımlı direnç dizisi eşdeğer direnci ve (iii) koda-bağımlı VGS ve VBS ile değişen kapalı anahtar direnci, direnç dizisinin her boğumuna yerleştirilen seri kalibrasyon dirençleri ile kompanse edilmiştir.

Mimarinin çalışma prensibi, her kod için bir kaç kat boyunca belli anahtarların açılması ve kapanması ile istenen direnç-dizisi boğumunun çıkış hattıyla iletime girmesi üzerine kuruludur. Bu esnada, zamanlama idealsizlikleri ve tranzistörlerin tipik davranışları gereği, bir takım doğrusal olmayan yük boşalımları gerçekleşmeltedir. (iv) Çıkış hattı üzerindeki yük enjeksyonu ve diğer ilgili zamanlama hataları, özgün bir diferansiyel direnç-dizesi ve diferansiyel ara-değer-bulan OPAMP mimarisi ile büyük ölçüde azaltılmıştır.

Çıkış katı tamponunun doğrusallığı, DAC'ın SFDR başarımında kilit rol oynayan unsurlardan biridir. Küçük belirgin özellikli (small feature size) proseslerde, kısa-kanal etkisi (short-channel effect) olarak adlandırılan bir MOS transistör özelliği görülmektedir. Bu etki, uzun-kanallı (1um'den fazla) transiztorlerde tam anlamıyla lineer olmayan ID/VDS özeğrisini, kısa-kanallı transistorlerde daha doğrusal kılan bir etkidir. (v) Çıkış tamponu doğrusalsızlığı, kısa-kanal etkilerinden yararlanılarak class-AB çıkış katı transizstörlerinin hız-doygunluk bölgesinde sürülmesiyle büyük ölçüde azaltılmıştır.

Direnç-dizisi mimarisinin eşdeğer direncini düşürmek için, diziyi düşük dirençli döngülerden oluşturmak, 2 kattan oluşan anahtarlama şemasında tipik operasyon sırasında yüzen düğümler oluşturur. (vi) Direnç dizesi yüzen düğümlerinin LSB hassasiyeti oluşturması, döngü ön-yükleme anahtarlarıyla giderilmiştir.

Bu özgün mimari geliştirmelerinin yanında, tüm katların şematik seviyede detayları (anahtar boyutları, anahtarlama mimarisi, referans gerilimleri, kod çözücü şeması, direnç-dizesi döngü uzunluğu, direnç-dizesi akımı, anahtar katı sayısı, vb.), serimden sonra çıkış hattı oturma karakteristiğini hıza ve kod-bağımsızlığa optimize edecek şekilde belirlenmiştir.

Benzetim ortamı Cadence 6.02 üzerinde Spectre+AMS ile TSMC 018 prosesinin BSIM4 modellerini kullanmaktadır. Bu çalışma yalnızca teorik mimarilerin geliştirilmesi üzerine değildir. Tasarım her yönüyle üretime hazır olacak incelikle geliştirilmiştir ve bu amaca yönelik olarak, proses modelleri tarafından sağlanan eleman uyuşmazlığı istatistiki dağılımları, proses varyasyonu istatistiki dağılımları, sıcaklık ve referans gerilimi kaymaları, serim sonrası parasitik direnç ve kapasiteleri eklenmesi gibi testlerle doğrulanmıştır.

Bu çalışmanın başlangıç noktası olarak aldığımız değiştirilmemiş standart mimari, 1 MHz (fo), 32 MHz (fs), 2 Vpp çıkış işareti için 60 dB şematik seviyesi SFDR vermektedir. Bu performans, çıkış hattı zaman sabiti kapasitif bileşeninin kalibre edilmesiyle 9 dB, çıkış hattı zaman sabiti dirençsel bileşeninin kalibre edilmesiyle 4 dB, diferansiyel direnç-dizesi ve diferansiyel ara-değer-bulan OPAMP mimarisinin geliştirilmesiyle 14 dB, tampon çıkış katı transizstörlerinin hız-doygunluk bölgesinde sürülmesiyle 10 dB SFDR artışı ile ortak etkilerden bağamsız olarak iyileştirilmiştir. Mimari önerilerinin bir araya getirilmesiyle, çalışmanın sonunda önerilen tasarım, tipik proses köşesi, sıcaklık ve referans kayması şartları altında 88 dB şematik seviye SFDR, 83 dB layout seviye SFDR vermektedir.

Tape-out Nisan 2015'te beklenmektedir. Tape-out sonrası ölçüme hazırlık olarak piyasadaki en iyi performans veren DAC (TI DAC8580) spesifikasyonuyla tam örtüşecek şekilde ölçülmüştür.

1. INTRODUCTION

Data converters are a family of integrated circuits that interface the two domains of signal transmission and processing: digital and analog. Basic operation of digital-to-analog converters and analog-to-digital converters are demonstrated in Figure 1.1.

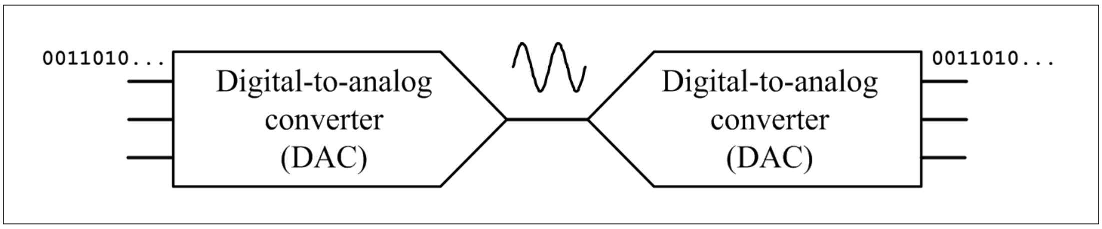

0011010...
Digital-to-analog converter (DAC)
Digital-to-analog converter (DAC)
0011010...

Figure 1.1 : Data converters.

This work is the design of a novel digital-to-analog converter. To give context to the motivation behind this research, basic DAC architectures that led to the design of this work and their shortcomings are presented in the subsections below.

1.1 DAC Architectures

All DAC architectures require an output buffer stage to drive time-varying loads with a linear settling characteristic. In architectures such as current-mode binary-weighted DACs, current-mode R-2R ladder network DACs and other current-steering type DACs, the output buffer acts as a current-to-voltage (I-V) converter as shown in Figure 1.2.

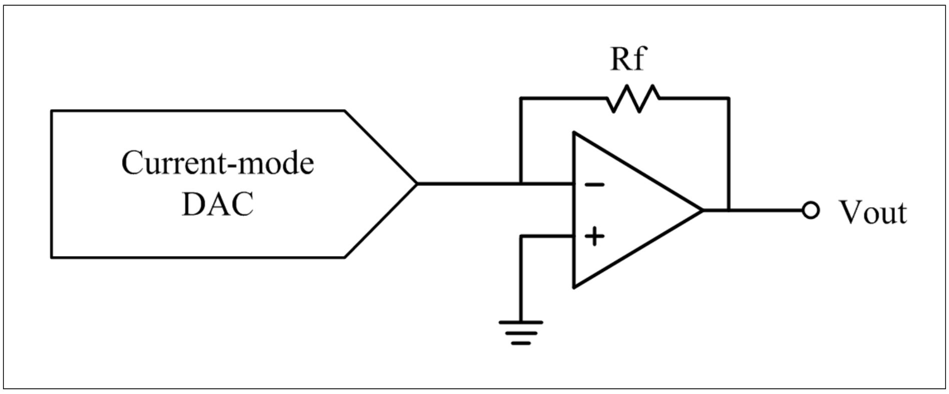


Figure 1.2 : Current-mode DACs.

Settling characteristic of these DACs are determined by the feedback resistors (Rf) sensitivity to temperature and voltage drifts and therefore limits static and dynamic linearity at high output frequencies [1]. Achieving precision settling is also a challenge with current mode DACs due to their inherent non-monotonicity and temperature drift [6]. Voltage-mode DACs shown in Figure 1.3 offer precision settling and monotonicity but have their shortcomings in speed and resolution.

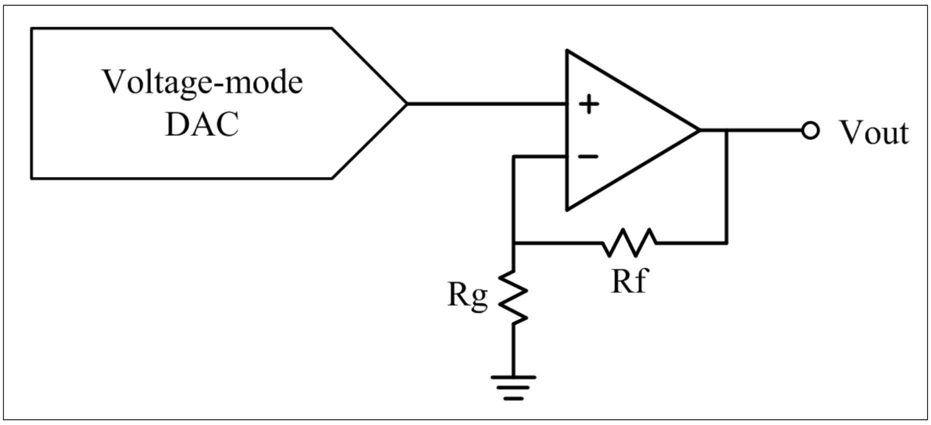


Figure 1.3 : Voltage-mode DACs.

1.2 Resistor String Type VODAC Architectures

Voltage-mode DAC architectures are based on selecting intermediate node voltages within a resistor network that sit between two voltage references. These node voltages are tapped by a decoding/switching architecture and output through a voltage buffer. A basic 10-bit resistor-string type DAC is shown in Figure 1.4.

Speed and dynamic linearity of these basic resistor string architectures are limited by the high and varying equivalent resistance seen for every code, and the resolution is limited by the element matching capabilities of the fabrication processes. Resistor string element matching for 10-bits of resolution is considered to be the limit of current process technologies [2]. With the employment of costly techniques like SiCr resistors, on wafer laser trimming or digital calibration unique per device, these architectures can achieve 12-bits of accuracy at most [3], [4].

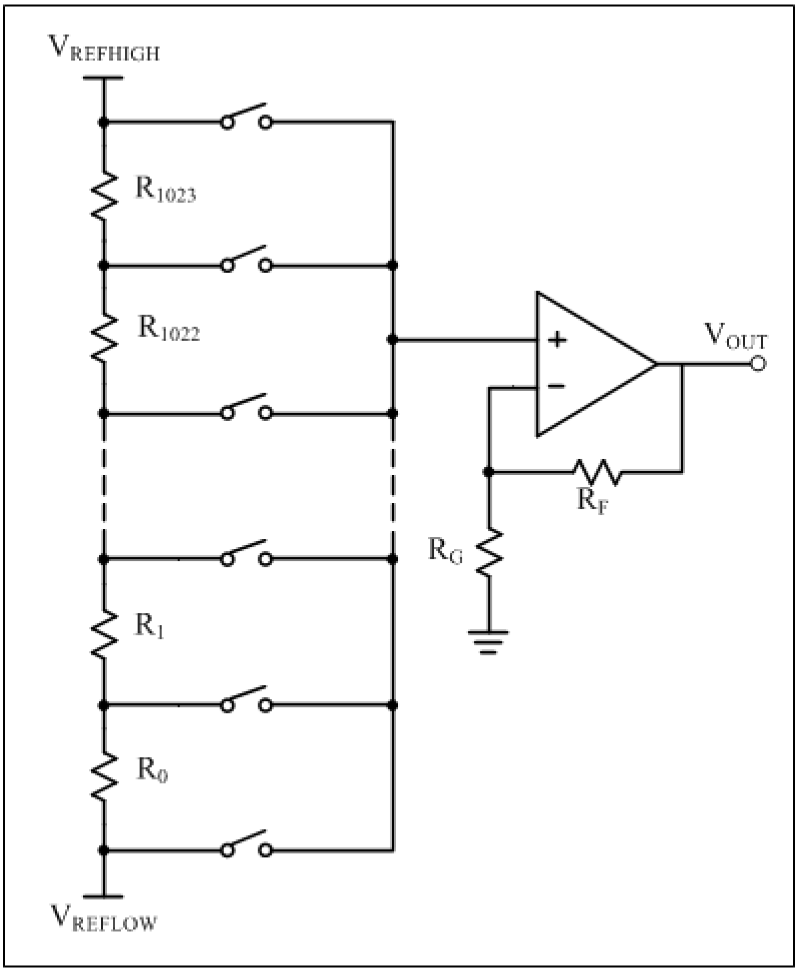


Figure 1.4 : 10-bit resistor string DAC.

The problem of resolution can be addressed with an interpolating OPAMP architecture as the output buffer [5]. This architecture has multiple positive inputs which are averaged and fed back to the negative input, essentially rendering the OPAMP differential pair a voltage interpolator as shown in Figure 1.5.

These multiple positive inputs can be weighted or thermometer coded for binary or unary operation. Figure above demonstrates a 6-bit thermometer coded operation, with 64 same sized positive inputs, and 1 negative input 64 times the size of the multiple positive inputs. A switching structure shown in Figure 1.6 decodes a pair of voltages as the binary input to the interpolator, so that this architecture can serve as a DAC.

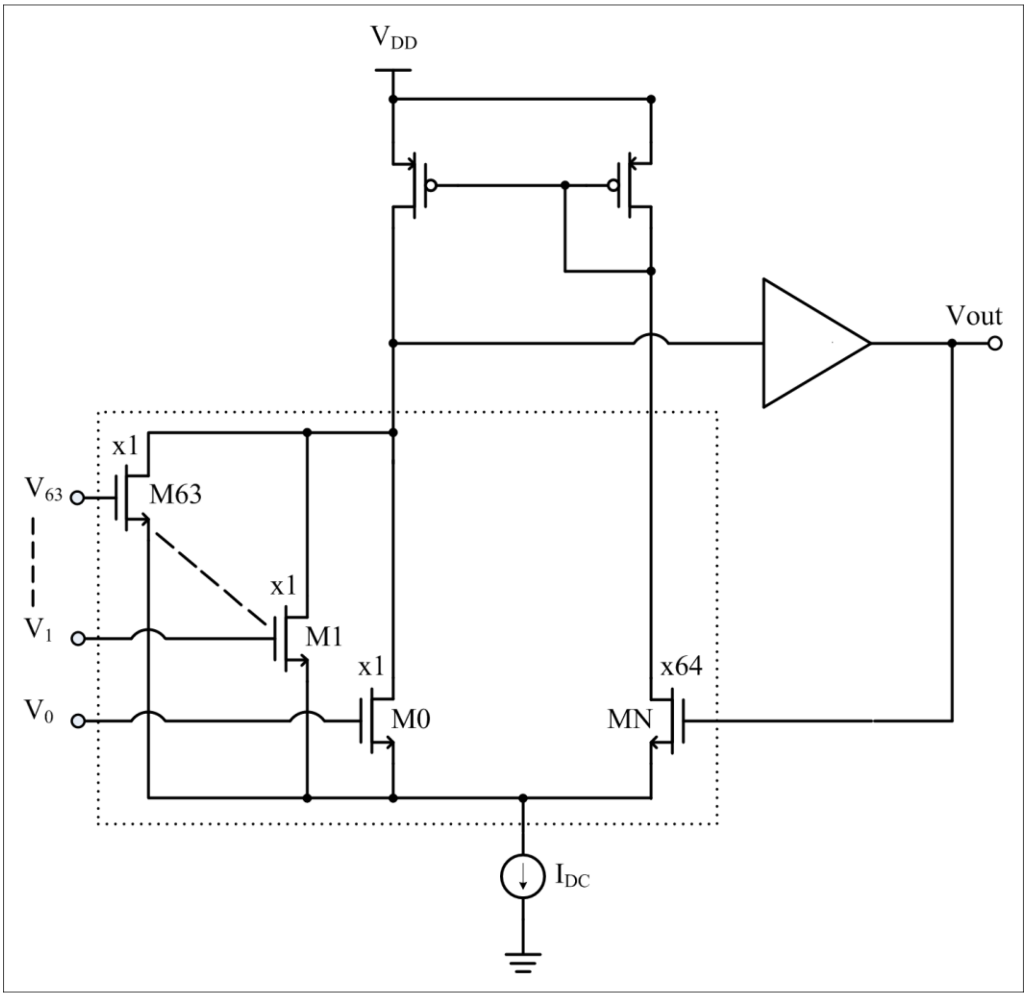


Figure 1.5 : Interpolating amplifier.

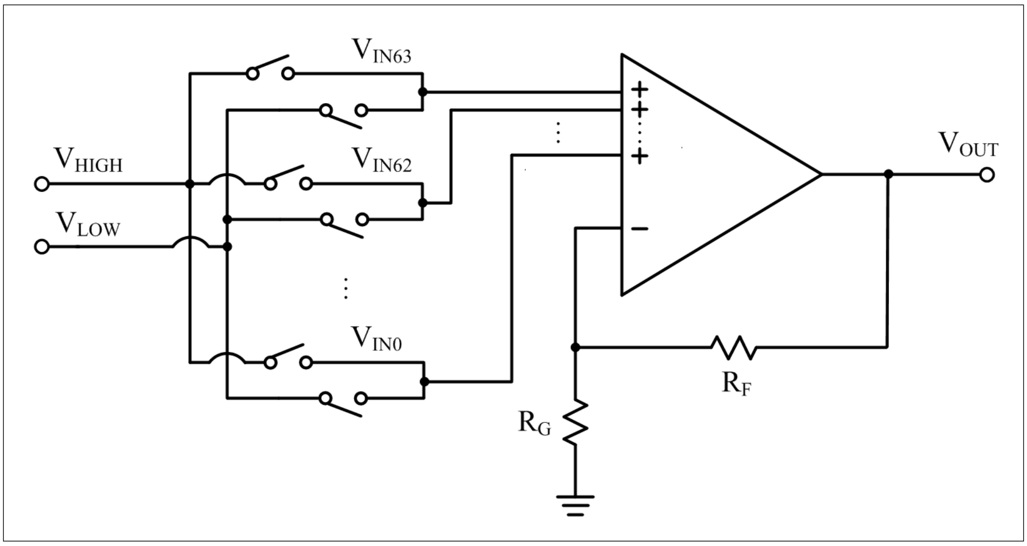


Figure 1.6 : Interpolating amplifier DAC.

Such an architecture can be used as the output stage of a voltage output DAC to increase resolution, if stages that precede the interpolation amplifier outputs a pair of voltages that can serve as V_{HIGH} and $V_{LOW.}$ The basic resistor string architecture given in Figure 1.4 can be modified to give a pair of voltages separated by 1 least significant bit (LSB) as shown below in Figure 1.7.

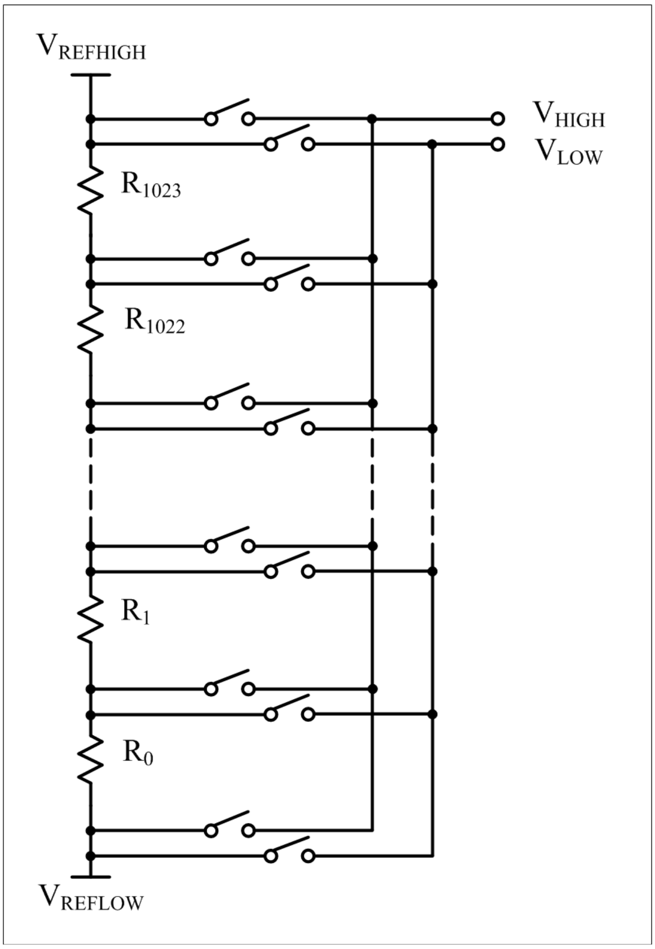


Figure 1.7 : Dual output resistor string DAC architecture.

Cascading the two architectures in stages, the following 16-bit, low bandwidth DAC architecture is realized in Figure 1.8.

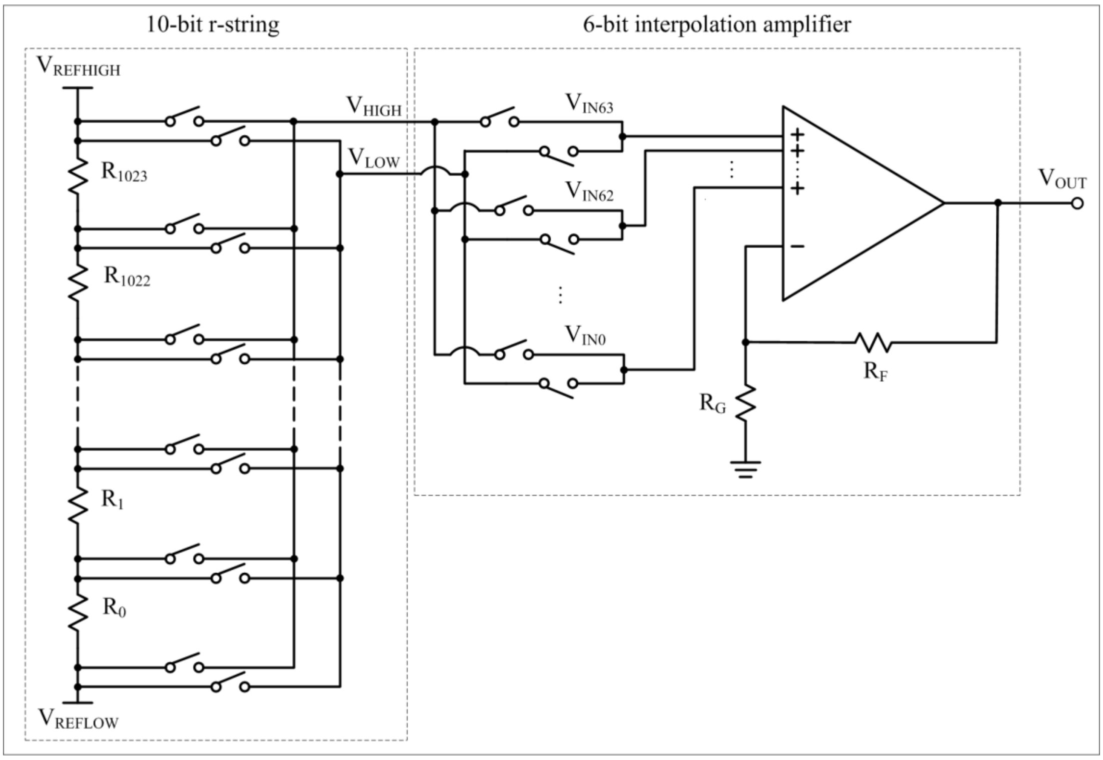


Figure 1.8 : 16-bit low bandwidth DAC.

The problem of speed is addressed by lowering the equivalent string resistance of the first stage. The following architecture in Figure 1.9, called the course-fine intermeshed string, proposed in [2] achieves this by creating low impedance nodes on the 10-bit fine string every 32 resistors by tapping loops of resistors from a coarse string. Tap voltages are selected in two levels of switches; first one to choose the tap voltage, second one to choose the loop the tap voltage is located in. The selected node is output through a voltage buffer as usual. Another improvement this architecture brings is reduction in parasitic capacitance otherwise coming from 1024 switches connected to one output node.

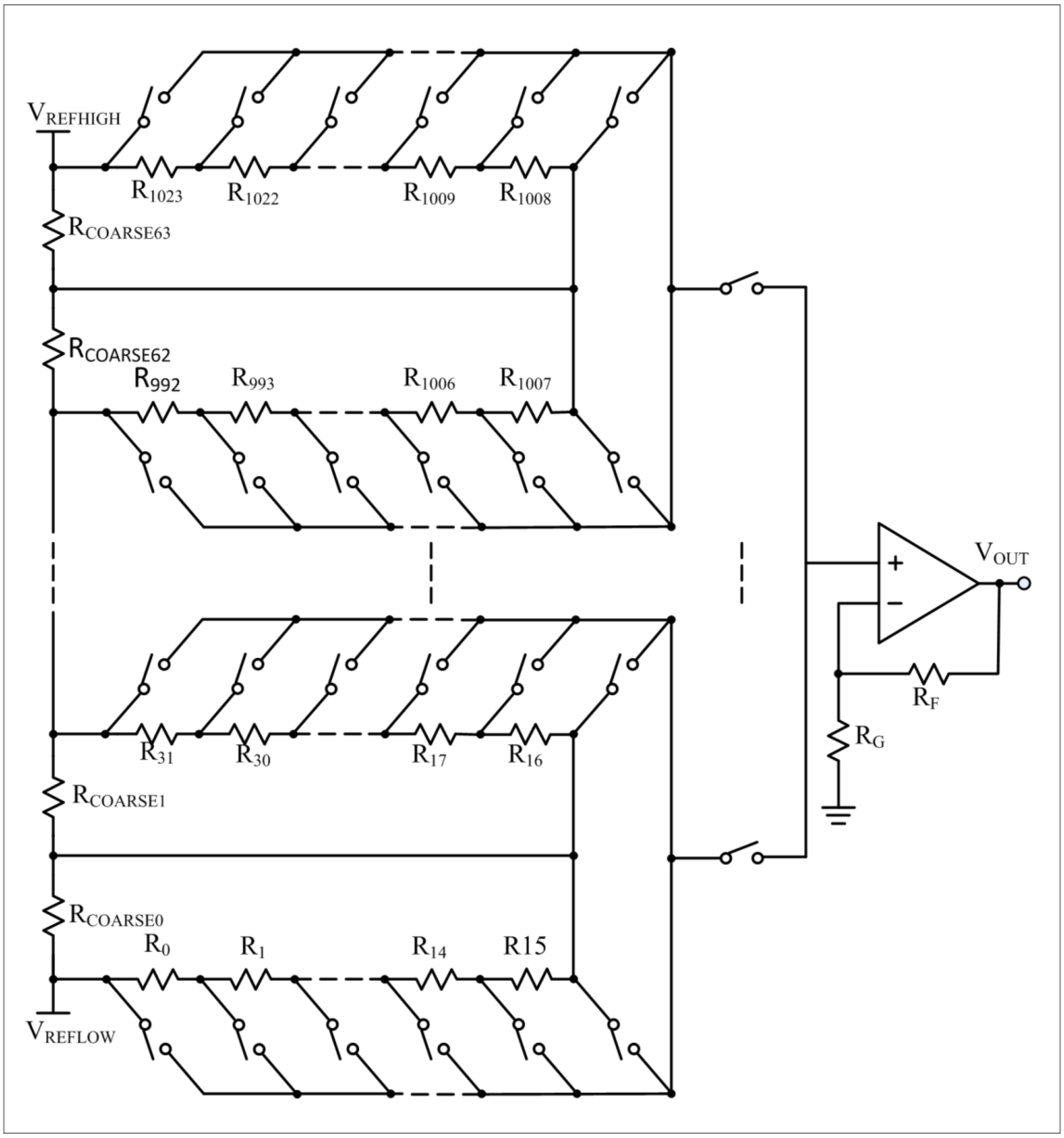


Figure 1.9 : Coarse-fine intermeshed 10-bit r-string.

This architecture can be cascaded with the interpolating amplifier architecture to result in a relatively fast 16-bit DAC design, shown in Figure 1.10 [5], [7].

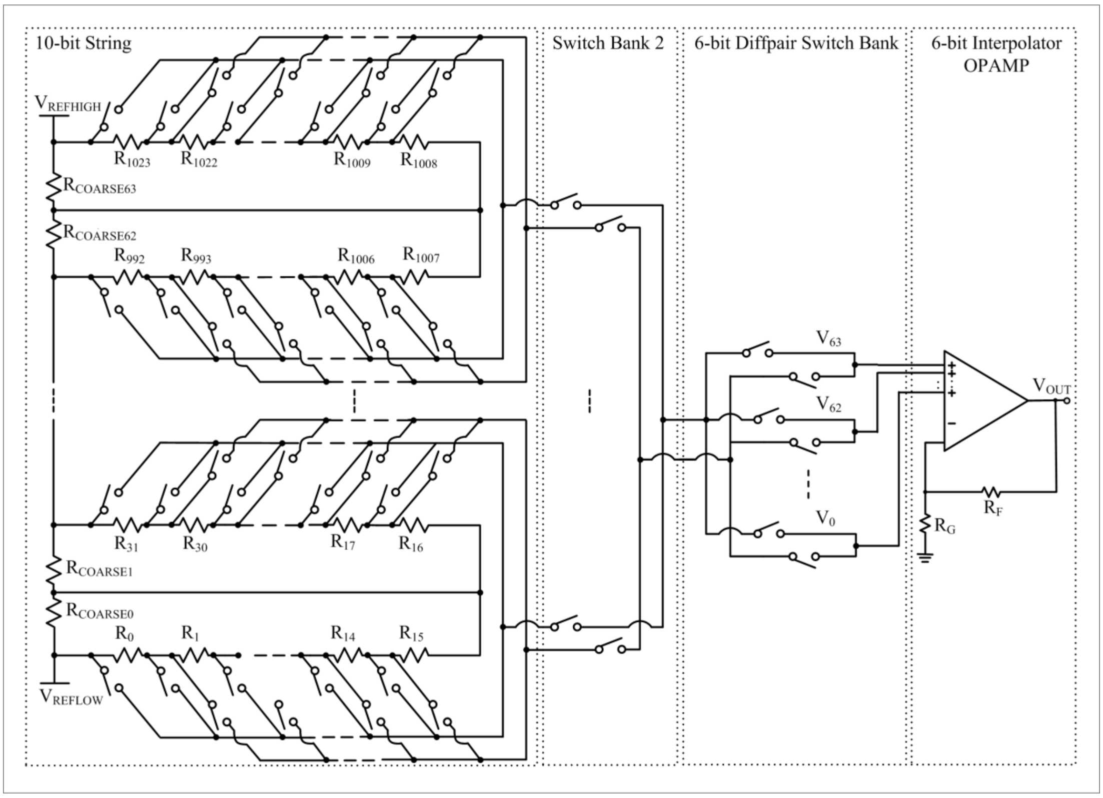


Figure 1.10 : 16-bit DAC.

SFDR of the architectures presented this in this subsection are given below in Table 1.1.

Table 1.1 : SFDR of basic r-string DAC architectures.

Architecture	**SFDR at 200 kHz (dB)**	**SFDR at 1 MHz (dB)**
Figure 1.2.1 (10 bit)	51	37
Figure 1.2.5 (16 bit)	64	48
Figure 1.2.6 (low resistance 10 bit)	68	51
Figure 1.2.7 (low resistance 16 bit)	74	**60**
Current state-of-the-art [8]	62	N/A

It can be seen from the table that the best design available on the market can’t offer much more than 60 dB SFDR at 200 kHz output frequency. Even though speed and resolution problems are partly addressed in the architecture presented in Figure 1.10, dynamic linearity at a MHz order bandwidth is nowhere near what is acceptable for voltage output DACs to be used in precision waveform generation applications.

1.3 Design Summary

Figure 1.11 shows a symbolic block diagram of the final design, the specifics of which will be explained in detail in the following chapters.

This work takes the architecture given in Figure 1.10 as a starting point and eliminates all major dynamic error mechanisms, taking schematic level SFDR at 1MHz to 88 dB and layout extracted SFDR to 83 dB. The final design is a continuation of a work referenced in [7], where several error mechanisms were investigated, and a schematic/ideal-behavioral level SFDR of 72 dB at 1 MHz output frequency were presented.

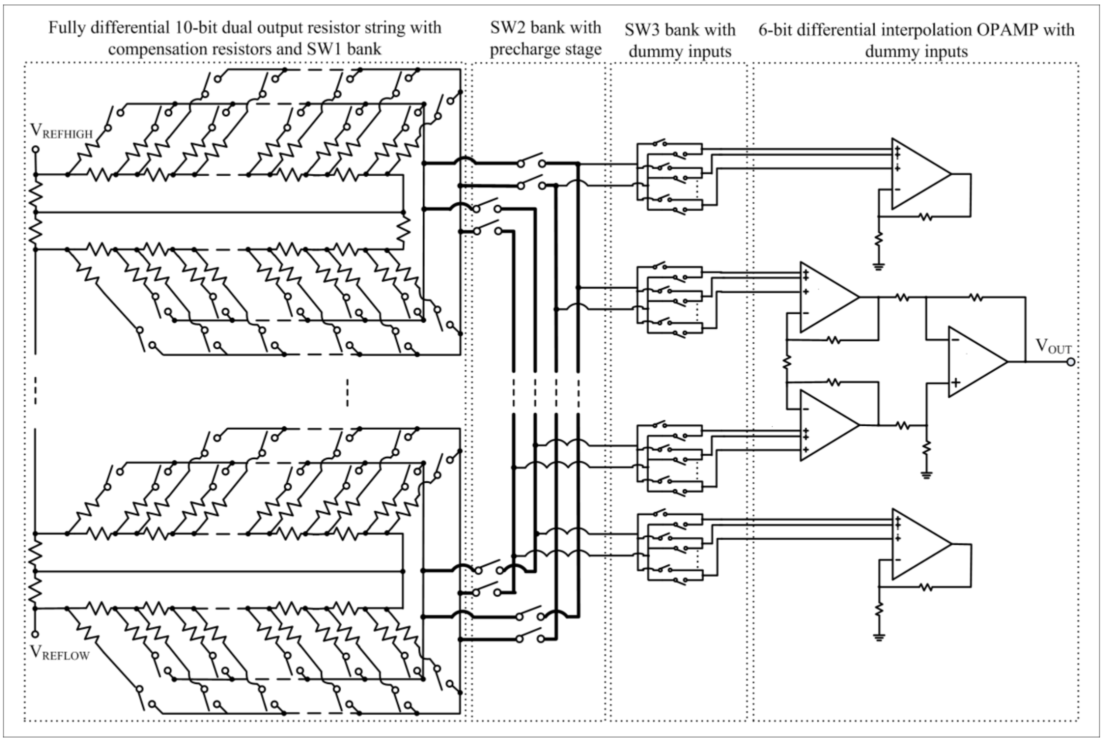

Fully differential 10-bit dual output resistor string with compensation resistors and SW1 bank
SW2 bank with precharge stage
SW3 bank with dummy inputs
6-bit differential interpolation OPAMP with dummy inputs
$V_{REFHIGH}$
V_{REFLOW}
V_{OUT}

Figure 1.11 : Fully differential 16-bit DAC.

The first stage implements a novel fully differential operation on a coarse-fine intermeshed resistor string, which outputs two pairs of differential voltages to the second stage which is a novel fully differential interpolation amplifier. Output bus capacitances are balanced for every code with the implementation of dummy interpolation amplifier inputs. Output bus resistance is calibrated for each code with the implementation of unique compensation resistors that take into account varying switch bank and resistor string equivalent resistances. A precharge stage (not included in the figure) is implemented to hold loop voltages at appropriate levels to reduce glitches on the output bus. A digitally programmable 60ppm bandgap voltage reference (not included in the figure) provides the internal references. A decoder (not included in the figure) controls all switch banks with a break-before-make switching scheme to minimize glitches on the output bus.

SFDR performance of the final design in schematic and layout is given in Figure 1.12 below along with the performance of E. Topcu's design [7] and the basic architecture given in 1.10 under the same conditions.

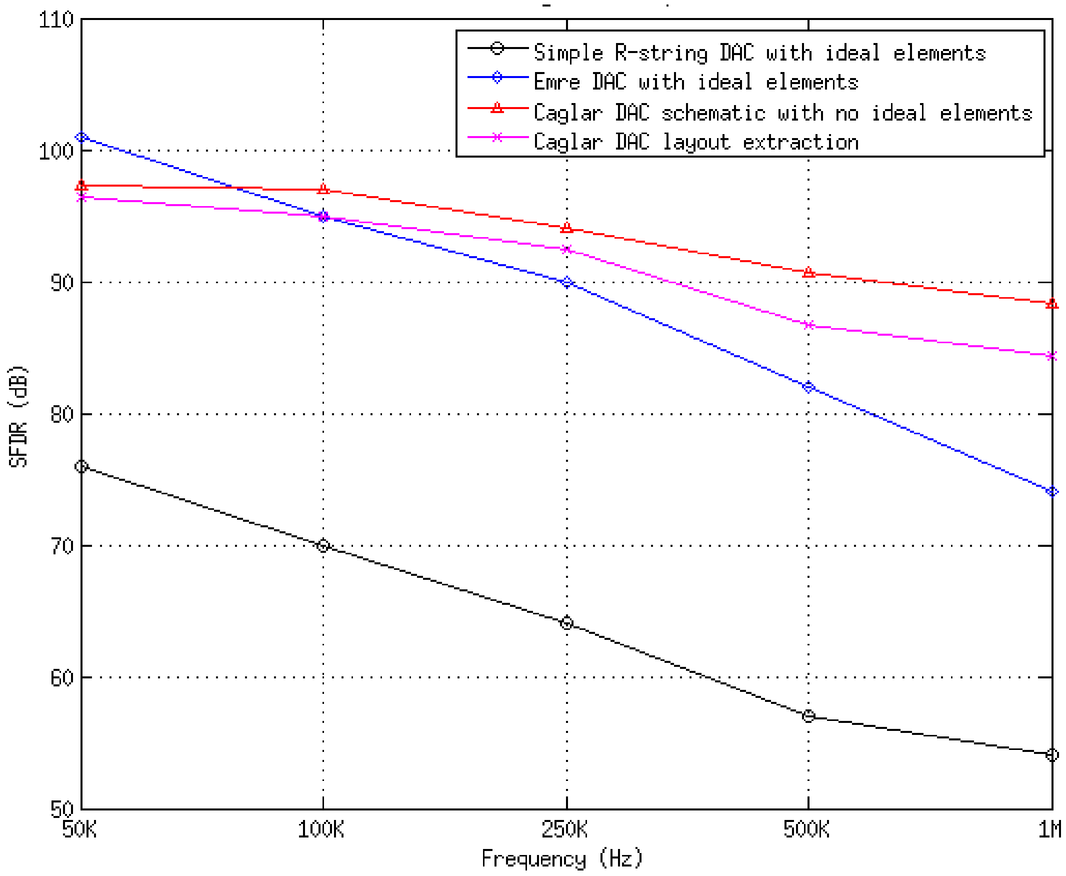
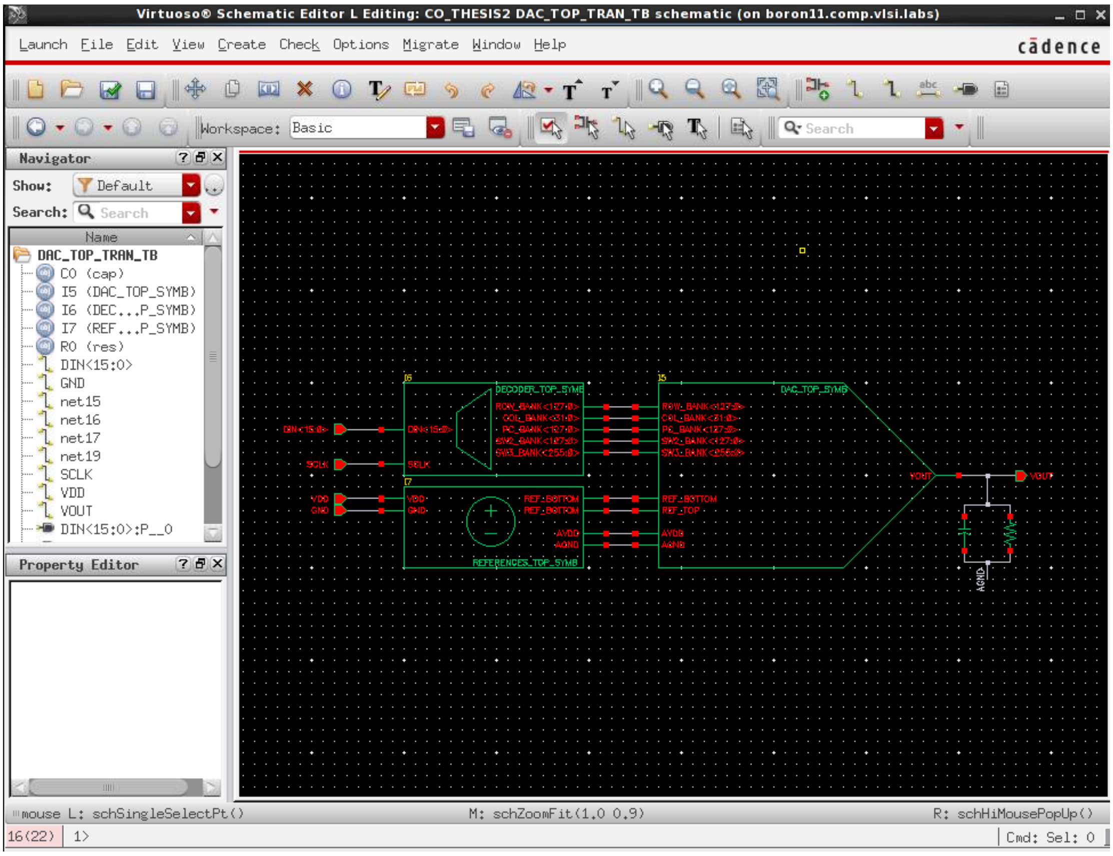


Figure 1.12 : SFDR vs. output frequency.

1.4 Design Environment

Cadence 6.02 Virtuoso Schematic, Virtuoso XL Layout and Spectre+AMS Verification tools referencing BSIM 4 models of the 018um TSMC process were used in the development of the design. A screenshot of one of the testbenches most frequently used in the development of the DAC is given in Figure 1.13. The testbench evaluates DFT for a 1 MHz f_o 32 MHz f_s signal and calculates SFDR, which is used as the primary performance metric in this study.

Voltage references that operate the resistor string, input decoder and the DAC core can be seen in their top cellview. Modifications on blocks lower down the hierarchy can be toggled on and off as dictated by the experiment as shown in Figure 1.14. Input to the design is generated by an ideal 16-bit ADC programmed in VerilogAMS, given in Appendix A. Output of the DAC is loaded with a 10 kΩ resistor and 100 pF capacitor in parallel.

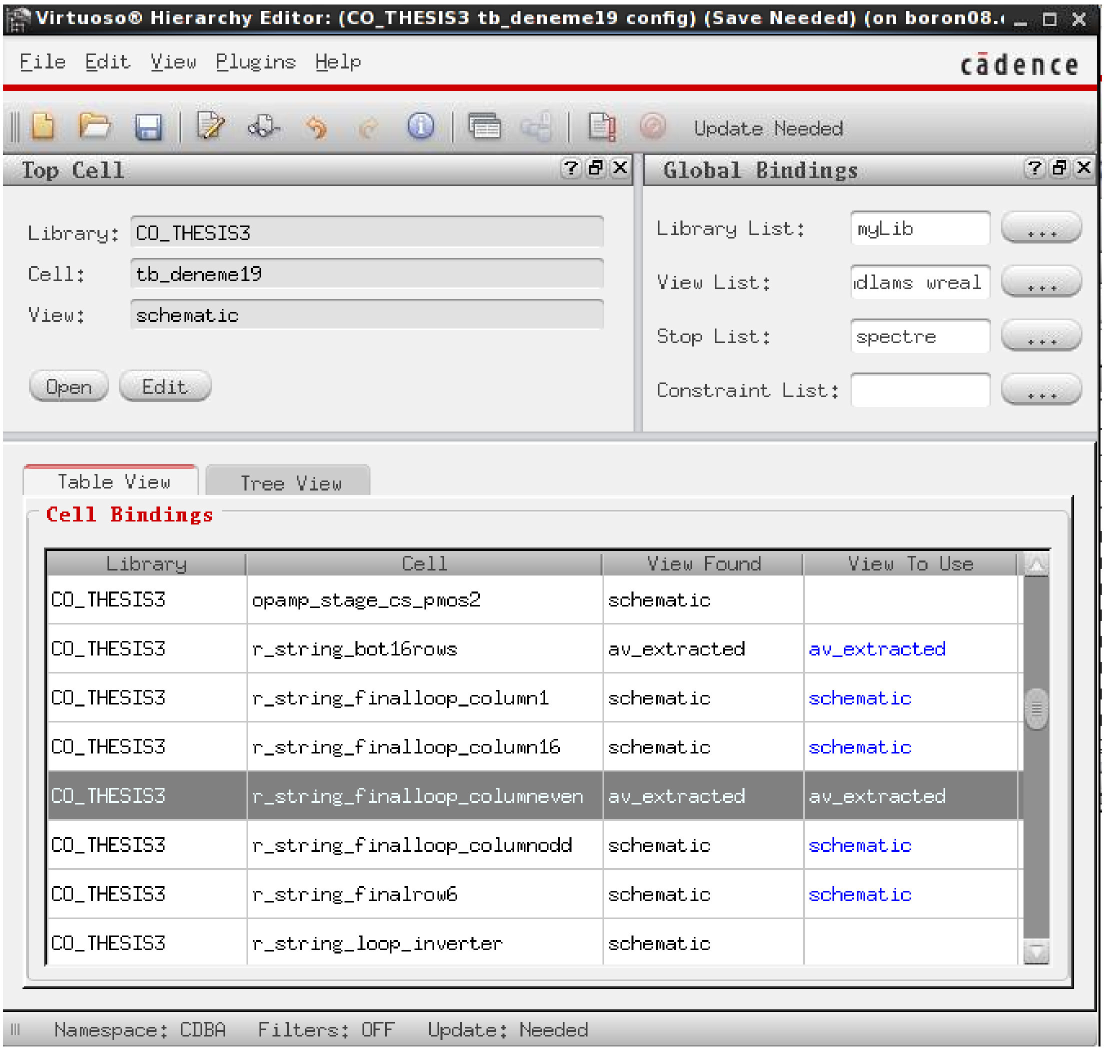


DAC_TOP_TRAN_TB
C0 (cap)
I5 (DAC_TOP_SYMB)
I6 (DEC...P_SYMB)
I7 (REF...P_SYMB)
R0 (res)
DIN<15:0>
GND
net15
net16
net17
net19
SCLK
VDD
VOUT
DIN<15:0>:P__0

mouse L: schSingleSelectPt() M: schZoomFit(1.0 0.9) R: schHiMousePopUp()

16(22) 1> Cmd: Sel: 0

Figure 1.13 : Simulation environment (schematic editor).


Virtuoso® Hierarchy Editor: (CO_THESIS3 tb_deneme19 config) (Save Needed) (on boron08.
File Edit View Plugins Help
cādence
Update Needed
Top Cell
Global Bindings
Open
Edit
Table View
Tree View
Cell Bindings

Library: CO_THESIS3
Cell: tb_deneme19
View: schematic

Library List: myLib
View List: dlams wreal
Stop List: spectre
Constraint List:

Library	Cell	View Found	View To Use
CO_THESIS3	opamp_stage_cs_pmos2	schematic	
CO_THESIS3	r_string_bot16rows	av_extracted	av_extracted
CO_THESIS3	r_string_finalloop_column1	schematic	schematic
CO_THESIS3	r_string_finalloop_column16	schematic	schematic
CO_THESIS3	r_string_finalloop_columneven	av_extracted	av_extracted
CO_THESIS3	r_string_finalloop_columnodd	schematic	schematic
CO_THESIS3	r_string_finalrow6	schematic	schematic
CO_THESIS3	r_string_loop_inverter	schematic	

Namespace: CDBA Filters: OFF Update: Needed

Figure 1.14 : Simulation environment (hierarchy editor).

Unless stated otherwise, DFT is taken for 1 cycle of a 1MHz fundamental signal between 1us and 2us for 65536 samples for all experiments reported in this work. Then, SFDR is calculated as:

$$SFDR(dB) = 20log\left(\frac{V_f}{max(V_{h1},V_{h2},V_{h3},\dots,V_{h10})}\right) \tag{1.1}$$

Figure 1.15 and 1.16 shows the transient and DFT outputs of the layout extracted final design under typical conditions for an arbitrary LSB pattern within 95% of the full scale.

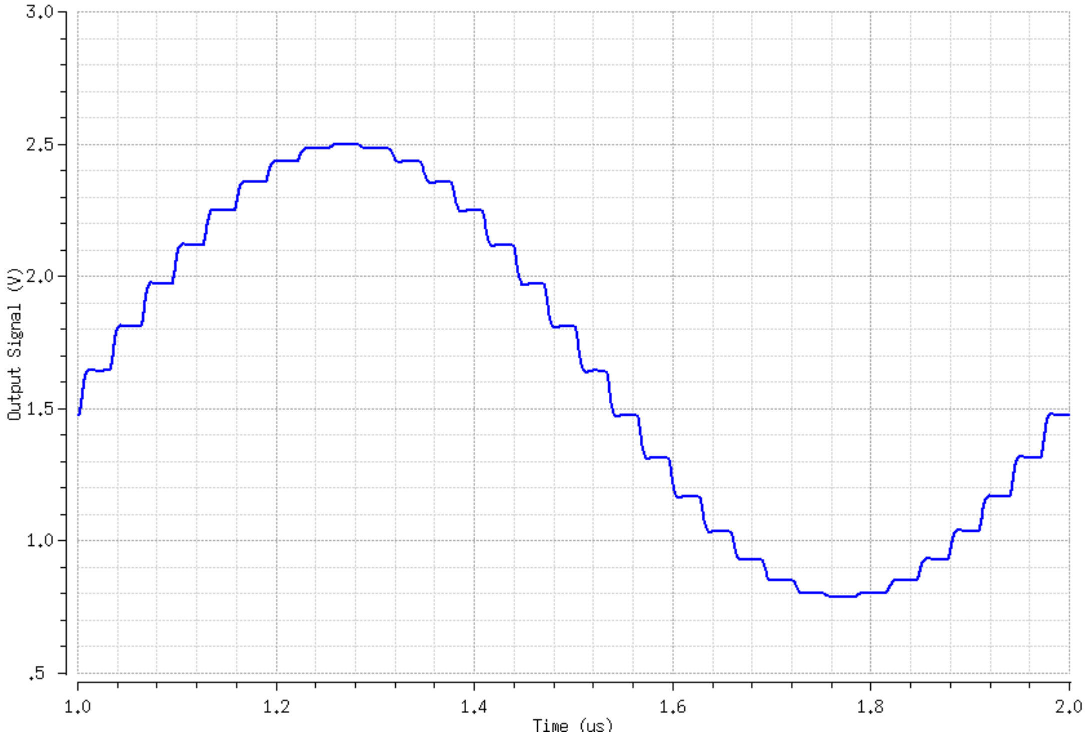


Figure 1.15 : Transient output waveform of the layout extracted design.

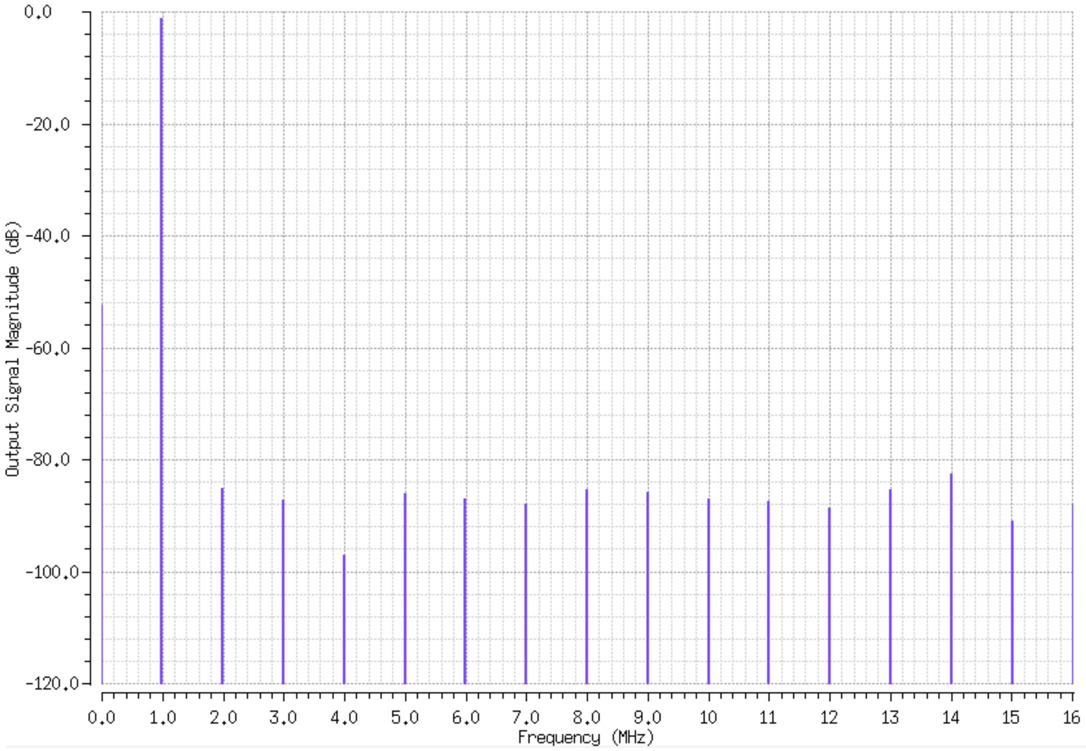


Figure 1.16 : DFT of layout extracted design (84 dB SFDR).

1.5 Thesis Organisation

In chapter 1, basic DAC architectures that led to the development of the final design are introduced, motivations for the research are justified and the final design is briefly overviewed, along with a brief description of the simulation environment in which the architecture was designed, laid out, and verified.

In chapter 2, major error mechanisms that were identified as affecting the dynamic linearity of the architecture are introduced, and their tradeoffs are briefly discussed to give context to the design decisions made.

In chapter 3, design decisions are justified and all stages are described in detail on a schematic level, along with detailed verification of the final design.

In chapter 4, layout of the DAC is presented along with layout extracted results.

2. ERROR MECHANISMS

2.1 Static Error Mechanisms and DC Error Correction

The ideal transfer function of a 3-bit DAC is shown below in Figure 2.1.

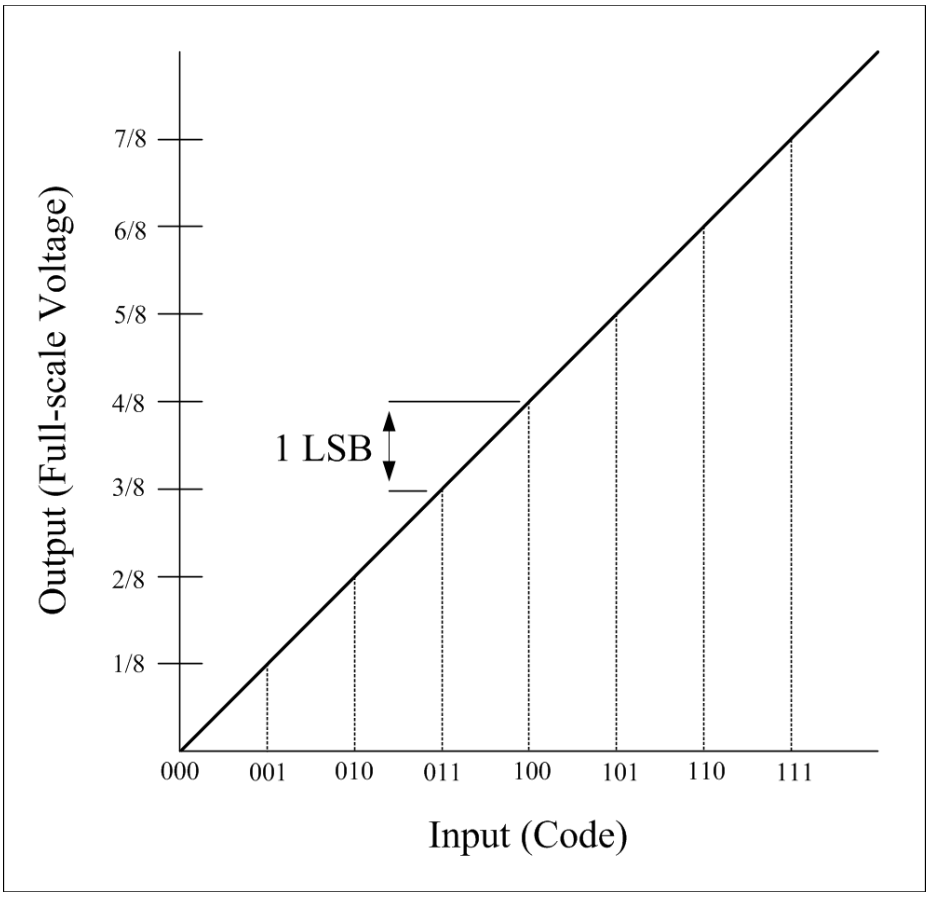


Figure 2.1 : Transfer function of an ideal 3-bit DAC.

Non-idealities such as imperfect voltage references, element mismatch at various stages of the architecture and layout related asymmetries are unavoidable especially with high performance converters. These issues can be local to a section of the DAC, local to periodic sections or can accumulate with a random pattern over all codes and cause the transfer function to be incorrect to various degrees [11]. These matters are often referred to as offset, gain, and INL/DNL errors, or DC (static) errors, and are shown in the representative 3-bit non-ideal transfer functions below in Figure 2.2.

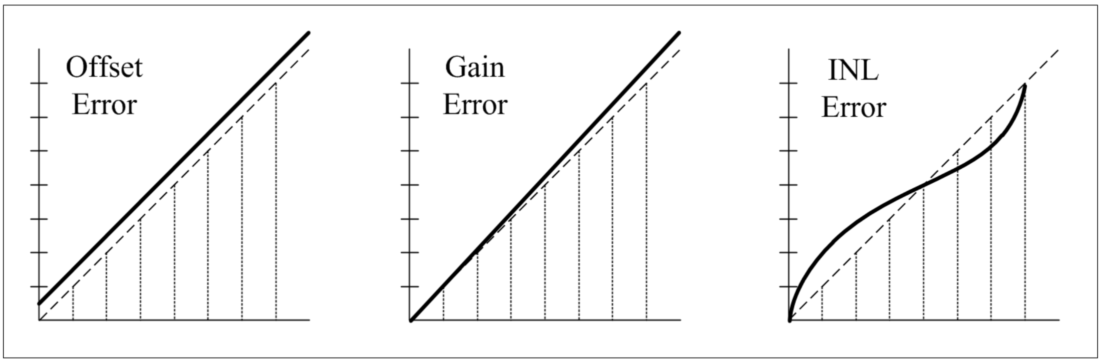


Figure 2.2 : DC (static) errors of DACs.

Developing a novel method for the compensation of such static errors is not the subject of this work, simply due to the fact that current solutions available for a wide range of tradeoffs have already become standard practice [9]. Considering die area, cost and complexity, for this work, we chose to employ a basic, all-digital software calibration shown below in Figure 2.3.

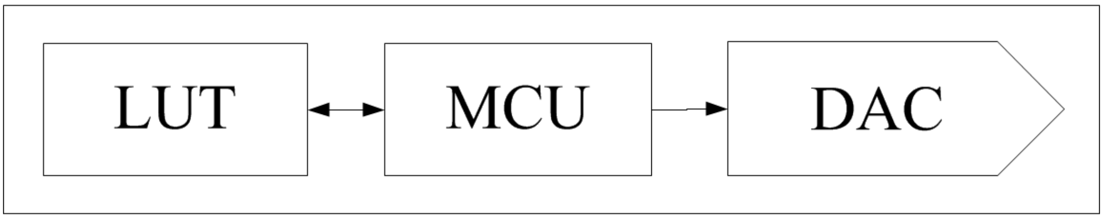
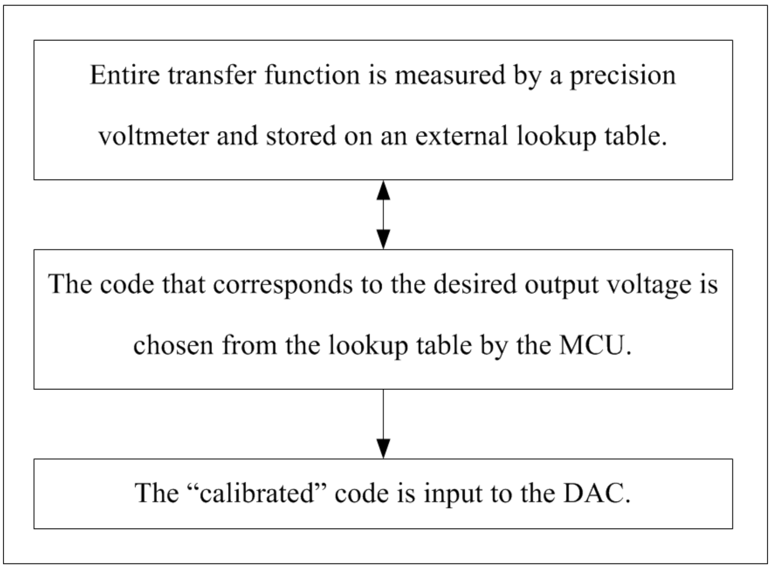


Figure 2.3 : DC calibration setup.

The algorithm carried out by this off-chip setup is very simple and is shown below in Figure 2.4.



Entire transfer function is measured by a precision voltmeter and stored on an external lookup table.

The code that corresponds to the desired output voltage is chosen from the lookup table by the MCU.

The “calibrated” code is input to the DAC.

Figure 2.4 : DC calibration algorithm.

This method is a simplification on a scheme outlined in detail in [9]. As pointed out in the same paper, for a 16-bit DAC, the lookup table requires 65536 words of storage, which is a burden on cost. A suggested solution to cut down on the size of the LUT is to divide the DAC into segments and apply a piece-wise linear algorithm to approximate towards the calibrated code. There are many other on-chip and off-chip solutions available to the DC calibration problems which are out of the scope of this work [9], [10].

2.2 Dynamic Error Mechanisms

Identification and elimination of dynamic error mechanisms are the main focus of this work. Dominant dynamic error mechanisms are briefly explained below, with the help of simple models. An ideal DAC programmed in VerilogAMS (Appendix C) with an ideal RC load can be a simple but accurate representation of a voltage output DAC as shown below in Figure 2.5.

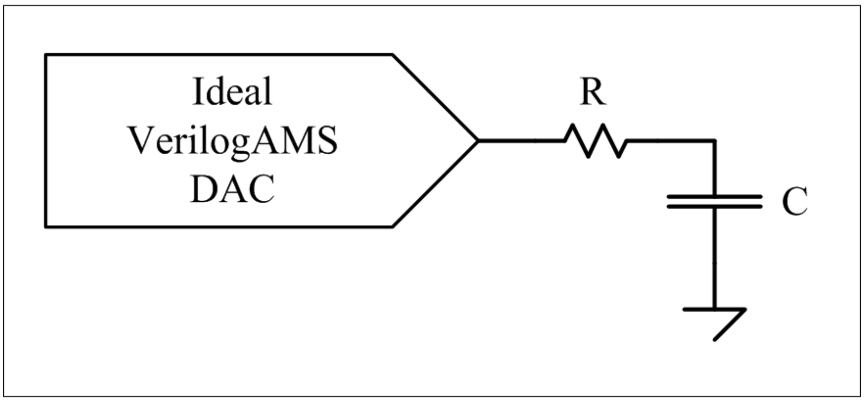


Figure 2.5 : Ideal DAC model.

Simulations show that magnitude of this RC time constant does not affect dynamic linearity (up to some maximum settling time requirement calculated in the next sub-section), giving ~106 dB SFDR for a 1 MHz f_o, 32 MHz f_s, 2 V_{pp} output signal. This premise is the starting point of our research. To ensure code independent time constant equalization, several techniques were used on an architecture level, as well as on layout. Details of such techniques are described in detail in section 3 and section 4. This chapter is only intended to introduce some of the tradeoffs that were faced during the design of our final architecture.

2.2.1 RC model of DACs and method

Most basic RC model of our DAC is given below in Figure 2.6. 10-bit resistor string DAC and its 2 stage switch banks are labeled as ‘STRING’, ‘SW1’ and ‘SW2’. 6-bit thermometer decoded LSB switches and the interpolating OPAMP inputs are labeled as “SW3” and “OPAMP”.

At each stage, code dependent and code independent effects are to be compensated using techniques and architectures to be explained in the following chapters.

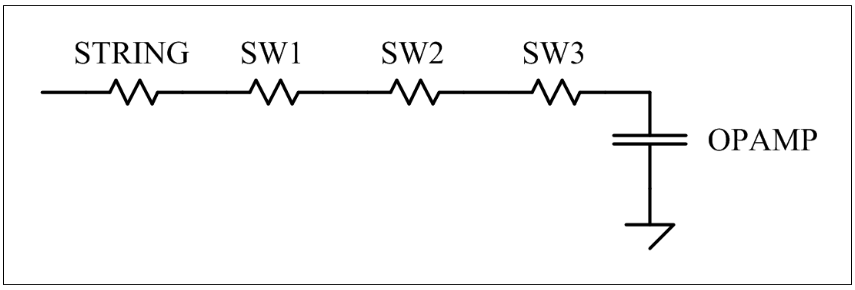


Figure 2.6 : Basic RC model of our DAC.

Balancing of the output bus impedance, and thus the time constant for every code at the input of the interpolating OPAMP comes with a maximum settling time depending on the sampling frequency. The term time constant defines the time it takes for a step response to reach within $1/e \cong 37\%$ of its final value. The RC requirement is 2.85 ns as shown below.

$$y = 1 - e^{-t/\tau} \tag{2.1}$$

For 16 bit settling and 32 MHz sampling frequency,

$$t/\tau \cong 11 \tag{2.2}$$

$$t = \frac{1}{32\times10^6} = 31.2\,ns \tag{2.3}$$

$$\tau = 2.85\,ns \tag{2.4}$$

Various capacitive and resistive compensations to equalize large-signal settling for every code on the output bus cannot exceed this requirement. Approximate equivalent resistances and capacitances of the model shown in Figure 2.6 are shown below in Figure 2.7.

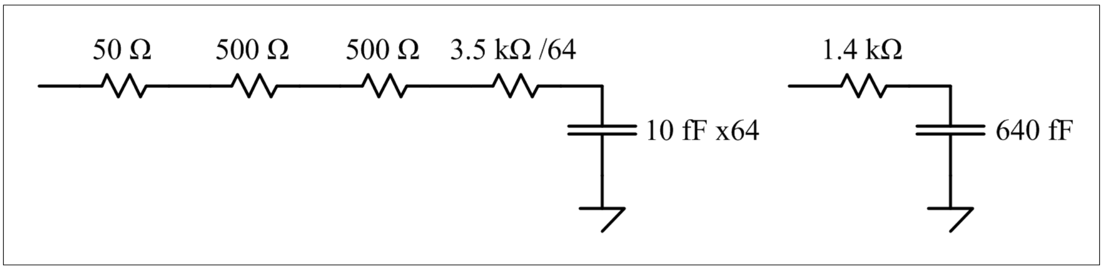


Figure 2.7 : Basic RC model of our DAC (with approximate values).

The specifics of these stages and the values indicated above will be explained in detail in the coming sections. According to this simplified model,

$$RC = 1400 \times 640 \times 10^{-15} \cong 0.8ns \qquad \textbf{(2.5)}$$

This approximation at the input of the interpolating OPAMP easily meets the requirement calculated above. A bit of headroom is a good practice for a starting point in design, since many error mechanisms will be present to interfere with this simplistic theoretical calculation.

Figure 2.8 below shows settling at the input of the OPAMP of the final DAC design. An arbitrary step response reaches 1τ of its final value in approximately 0.9ns, verifying the model approximate calculations.

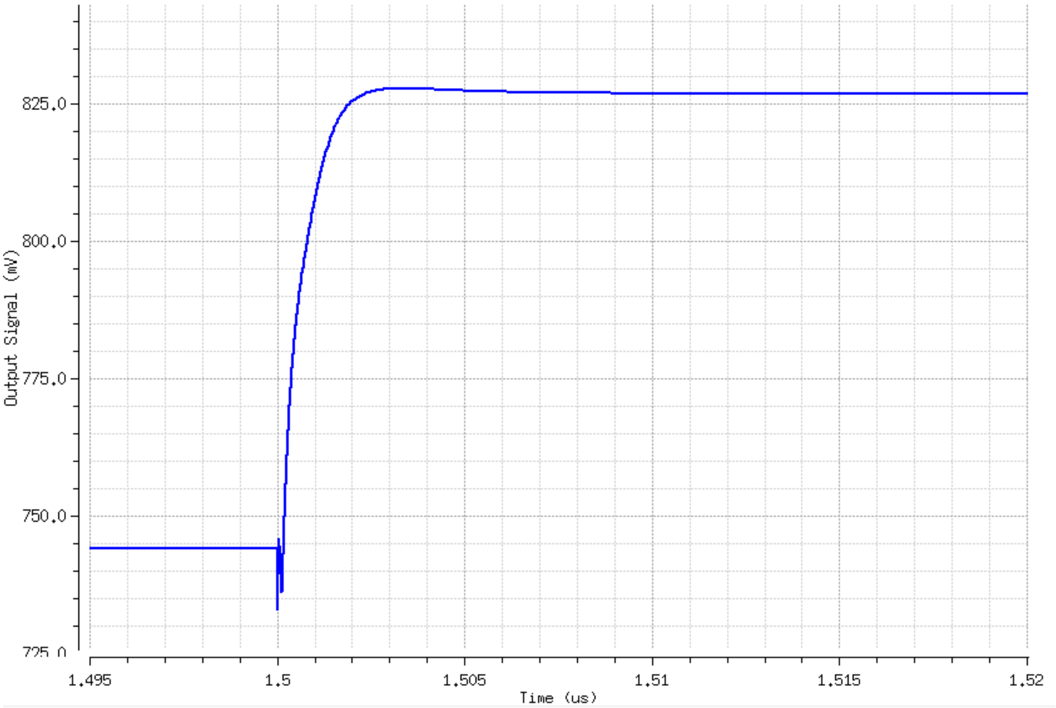


Figure 2.8 : Settling characteristic at the input of the interpolating OPAMP.

More detailed RC models of the simpler model given in Figure 2.6 is given below in Figure 2.9 for representative resistances and capacitances.

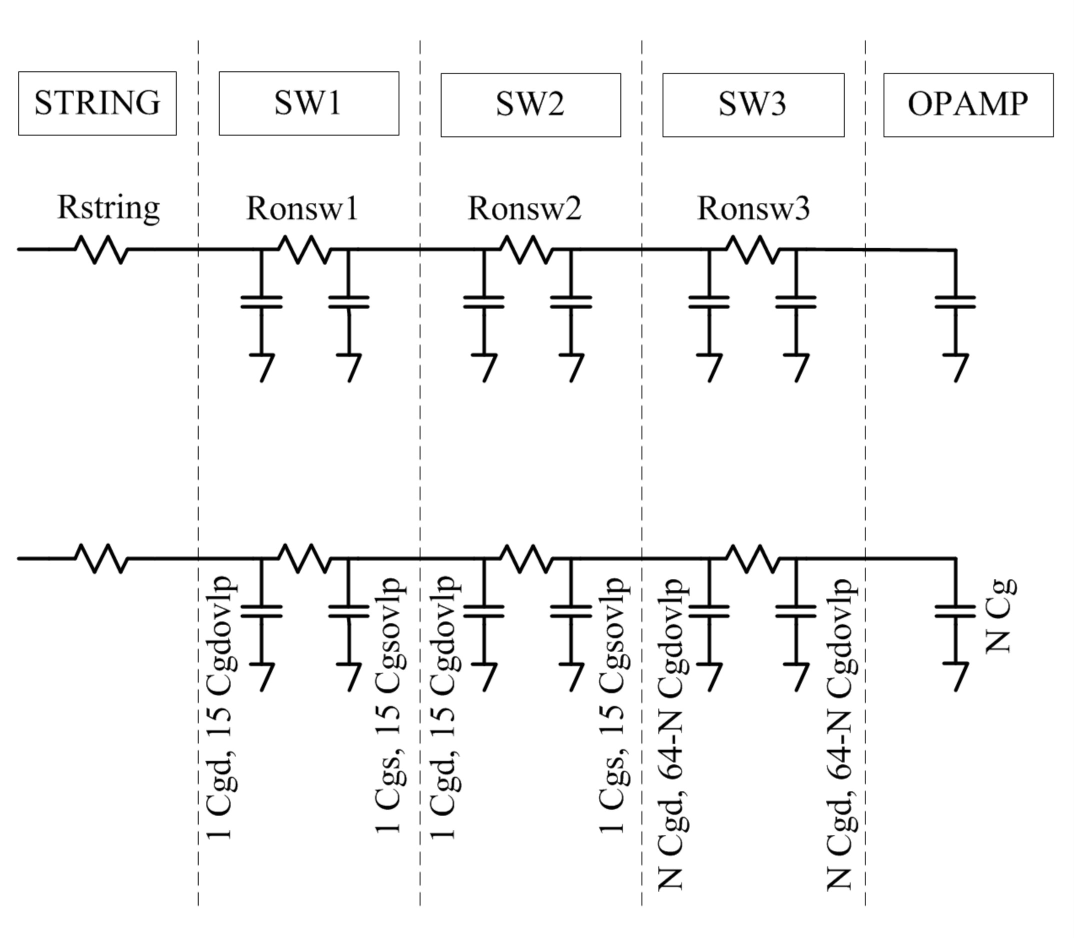


Figure 2.9 : Detailed RC model.

2.2.2 Code dependent equivalent resistance

Code dependent equivalent resistance variation is dominated by the first 3 stages: i) string, ii) switch bank 1, and iii) switch bank 2. For the unmodified architecture, string resistance *Rstring* (for the chosen string current, justified in later sections) varies between 0 and 50 Ω depending on the tap point. On resistances of the switches *Ronsw1* and *Ronsw2* (for the chosen sizes, justified in later sections) vary between 350 Ω and 600 Ω. The code dependent varying V_{gs} and V_{bs} of the NMOS switches cause this effect. Switch 3 bank consists of 64 switches connected in parallel (when dummy switches are in use, justified in later sections) which makes little impact on the added resistance *Ronsw3* along the path since the resistances. For code k, effective output bus resistance is:

$$R_{out}[k] = R_{string}[k] + R_{onsw1}[k] + R_{onsw2}[k] \quad \textbf{(2.6)}$$

Two different methods were explored in depth to counter this effect, constant V_{gs} switches and external resistive compensation. External resistive compensation seems to offer the best tradeoff in performance by simply equalizing the effective output bus resistance as:

$$R_{out}[k] = R_{string}[k] + R_{onsw1}[k] + R_{onsw2}[k] + R_{compensation}[k] \quad \textbf{(2.7)}$$

Details of these analyses for each of the stages will be given in section 3.

2.2.3 Code dependent equivalent capacitance

Experiments of manually editing out MOS terminal capacitances from the netlists show that, code dependent equivalent capacitance variation is dominated by the last 2 stages: i) switch bank 3, ii) interpolating OPAMPs. For the unmodified architecture, output bus capacitance varies between 0 and 640 fF depending on how many switches on the output bus are connected to the OPAMP differential pairs. For code k, effective output bus capacitance is:

$$C_{out}[k] = N \times C_{gdsw3}[k] + N \times C_g[k] \quad \textbf{(2.8)}$$

A dummy switch 3 bank and interpolating OPAMP connected to the same output bus with an inverted switching pattern equalizes this capacitance to a code-independent 640 fF.

$$C_{out}[k] = 64 \times C_{gdsw3}[k] + 64 \times C_g[k] \quad \textbf{(2.9)}$$

Switch banks 1 and 2 always connect 1 on switch and 31 off switches to the output bus for every code, and the small variation in gate and overlap capacitances that come with code dependent V_{gs} and V_{bs} of the NMOS makes insignificant difference in performance. Details of these analyses for each of the stages will be given in section 3.

2.2.4 Code dependent charge injection

Charge injection is the distribution of charge towards the source and drain terminals of a MOS switch when a channel is created or cut off as shown in Figure 2.10

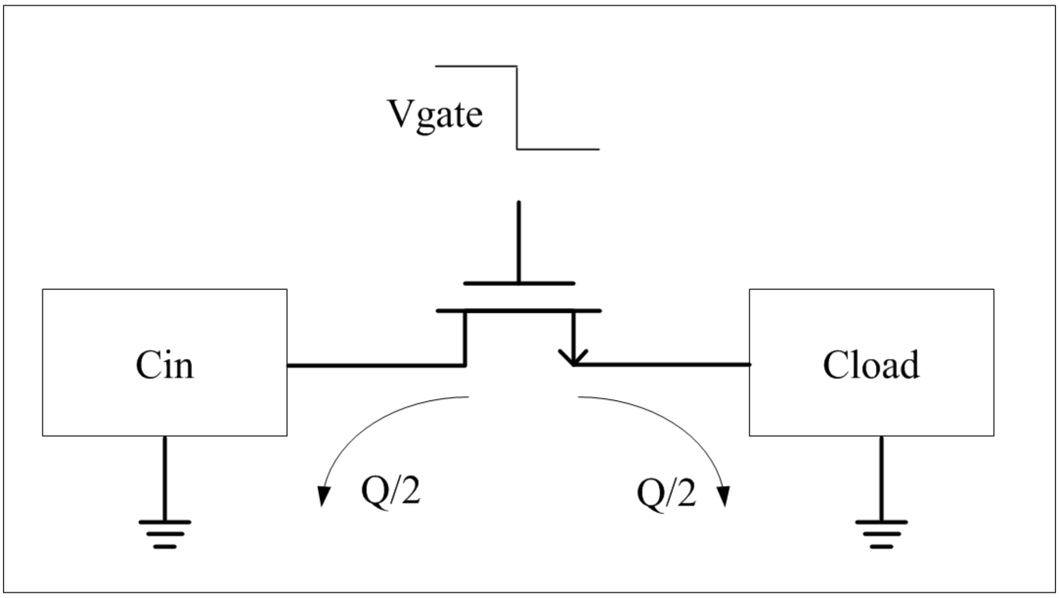


Figure 2.10 : Charge injection.

It is nonlinear due to a combination of effects related to the varying source terminal voltage along the string as the charge is collected and canceled out at varying times at different points of the string. Minimizing switch sizes reduces this effect, but as with all analog design decisions, there is a tradeoff – increased on resistance of the switch, which in turn increases the variance of the RC time constant per code. A constant V_{gs} switch architecture improves the performance when everything else in the design is taken to be ideal – but such an increase in complexity of all the switch cells and the consequent addition of auxiliary strings to operate these switches cause major layout-related problems and in the end does more harm than good. Main string references also has to be cut to almost half the range to ensure constant V_{gs} for both on and off states. With experimentation it is seen that, after layout, not using a constant V_{gs} architecture in all three switch banks results in the best performance. Charge injection is a common mode disturbance, and so the implementation of the fully differential architecture removes 2nd order distortions related to it. Details of these analyses will be given in section 3 and 4.

2.2.5 Code dependent data feedthrough

Data feedthrough is a similar phenomenon as depicted in Figure 2.2.4.1 but instead it is related to the source and drain overlap capacitances and is an effect seen when a switch is turned on or off as the output follows through the switches with varying overlap capacitances due to varying tap voltages along the string. Like code dependent charge injection, this effect can be reduced by minimizing switch sizes, but it comes with the same tradeoff. When overlap capacitances are edited out manually out of the netlist, a 2 dB improvement in performance is seen, when everything else in the design is taken to be ideal. A constant V_{gs} switch architecture significantly increases this effect with an additional increase in complexity due to additional switches and the addition of auxiliary strings which require separate reference voltages. Details of these analyses will be given in section 3 and 4.

2.2.6 Glitches

When switches are turned off, a part of the charge injection is coupled over the resistor string. A pre-charge stage that holds each of the second switch bank drain terminal voltages at appropriate levels is placed between the middle point of the string loop and second switch bank in order to minimize this effect. Fully differential string architecture is a major improvement to reduce glitches on the output bus. But since the magnitude of a glitch is related to where on the string the switching is taking place, it can never be completely eliminated even with a fully differential structure. Controlling the switches with a make-before-break timing reduces the momentary charge dumps on the string to some degree, but the only solution that offers significant performance improvement is to simply decrease the resistance of the string. The higher the current, the smaller the glitches will be, and thus higher the dynamic linearity. The tradeoff is higher power consumption which is limited to 10 mA over the string as a design specification decision. Details of these analyses will be given in section 3.

2.2.7 Interpolator non-linearity

Interpolator OPAMP stage linearity is a most important design feature of this DAC. Initial architecture comprised of a PMOS folded-cascode, gain-boosted, Monticelli Class-AB output. High dynamic linearity is achieved by driving the output stage transistors at their velocity saturation region. The tradeoff is again high power consumption, which is limited to 15mA at the output stage as a design specification decision. Details of these analyses will be given in section 3.

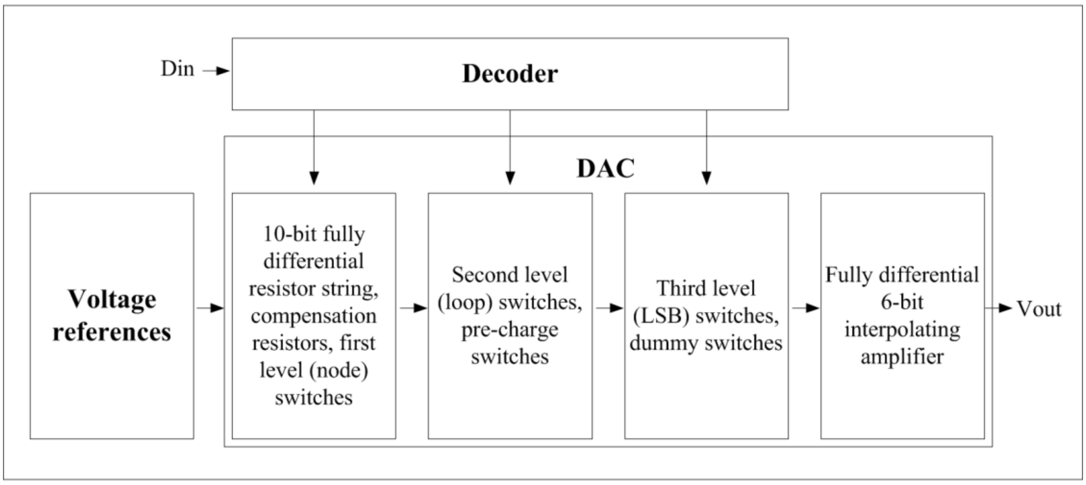
2.2.8 Signal-to-noise ratio (SNR)

Transient noise is mostly related to the voltage range of the architecture. Higher the range, lower the transient noise, so designing the device to have a high range is critical. The linear operating points of the several stages of the architecture determine this range. Another reason why true constant V_{gs} switching architecture is not being used throughout the design is, it is a bottleneck that limits the resistor sring reference range. This is due to the limited nature of proper on and off voltages required to operate NMOS switches with fast and linear characteristic. Details of these analyses will be given in section 3.

3. DESIGN DECISIONS AND SCHEMATIC

3.1 Architecture Overview

The design comprises of 3 main blocks: 1) DAC, 2) Voltage references, and 3) Decoder as shown below in Figure 3.1.



Din
Decoder
DAC
Voltage references
10-bit fully differential resistor string, compensation resistors, first level (node) switches
Second level (loop) switches, pre-charge switches
Third level (LSB) switches, dummy switches
Fully differential 6-bit interpolating amplifier
Vout

Figure 3.1 : Top level block diagram.

Inputs to the top level are 16-bit parallel LVDS DIN and LVDS double data rate (DDR) DCLK clocked at 32MHz. All references come from an internal band-gap reference, powered by 3.3 V VDD. The architecture outputs a single ended LPF pin to a 50-ohm end-terminated transmission line. DAC comprises of a fully differential 10-bit resistor string with first level switches (Stage 1), second level switches (Stage 2), third level switches (Stage 3) and 6-bit interpolating OPAMPs (Stage 4) as shown below in Figure 3.1.2. 10 MSBs are converted to two pairs of differential outputs within 1 of 32 loops in the first stage. Second stage switches select the loop on the resistor string which contain the selected first level switches and connect the tap voltage to the four MSB busses. These two pairs of 10-bit resolved outputs are the inputs to the third switch stage where the remaining thermometer-decoded 6 LSBs are input to 6-bit interpolating OPAMPs in instrumentation amplifier configuration.

3.2 String

First major performance improvement on the standard coarse-fine intermeshed string architecture given in Figure 1.10 is the implementation of a novel fully differential operation as shown in Figure 1.3.1. An SFDR comparison for a 1 MHz (f_0), 32 MHz (f_s) full-scale signal is given in Table 3.1.

Table 3.1 : Fully differential vs. single ended string.

16-bit architecture with	SFDR (dB)
Single ended string	60.12
Fully differential string	**74.43**

Node switches are XY decoded with 16 column, 64 row signals as shown in Figure 3.2 to massively reduce number of wires and therefore, reduce potential timing problems as well as layout related complexity problems. Long and unequal wires can lead to unsynchronized switching and momentary disconnects, which can lead to glitches and momentary floating nodes.

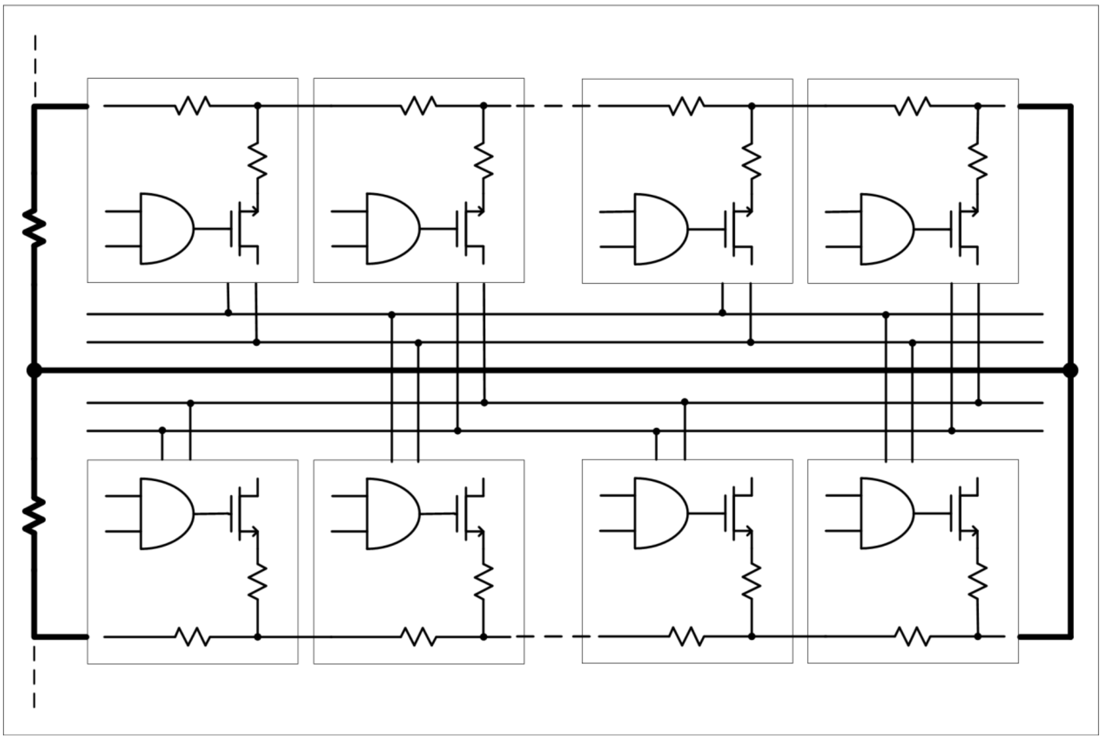


Figure 3.2 : XY decoded cells.

Pairs of 16 series connected resistors forms 32 loops that tap a 64 resistor coarse string to obtain the 10-bit string. Every other coarse resistor is the center tap of the fine string deflection points. All taps on a loop connect to 2 of the 4 primary output busses. There are 32 of these quadruple busses for each loop.

An even/odd node selection is decoded so that either output bus can be output the high voltage which the third switch bank resolves into 6 additional LSBs. This kind of switching architecture reduces the amount of switches in the first stage by half. For every code, the differential code is also selected by a separate switch to allow for fully differential operation.

Not using any constant V_{gs} switching schemes which will be justified in section 3 enables us to have a wide input voltage range which minimizes transient noise. The input range, which is the voltage reference range of the string, is limited by the interpolating OPAMP PMOS differential pair. Figure 3.3 below shows the results of the experiment conducted to optimize the input range of the fully differential architecture.

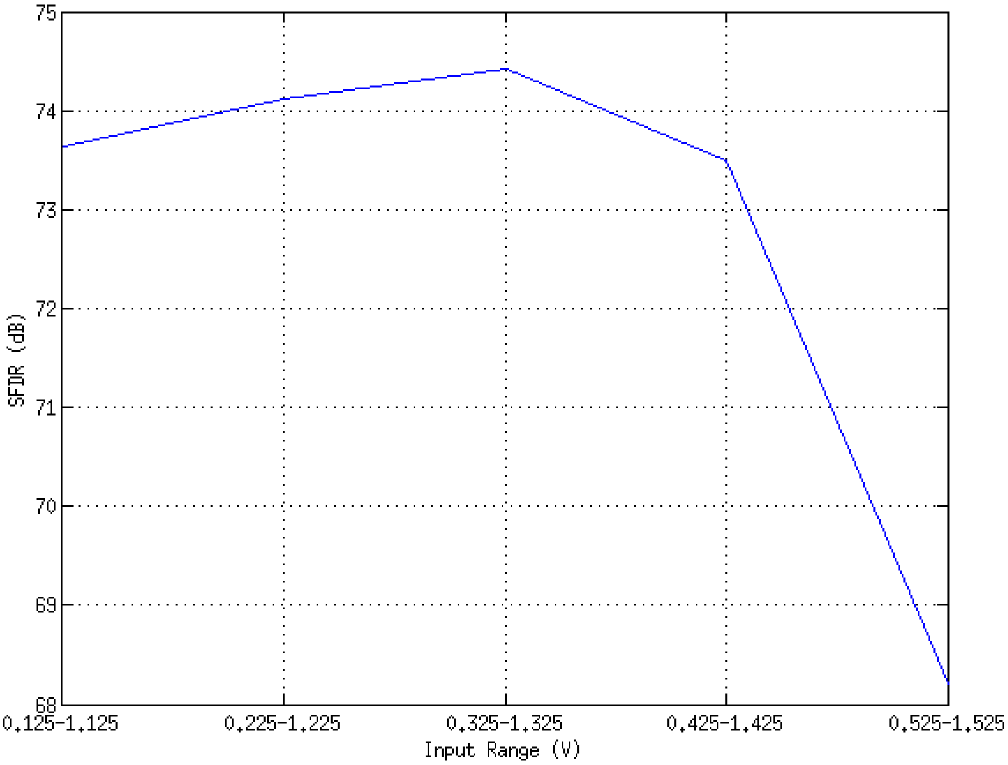


Figure 3.3 : Input range for low current 16 bit DAC.

Input PMOS pair is out of the saturation region for a 0.525 - 1.525 V signal. The optimum 1 V range of the device, and therefore the string voltage references are determined to be 0.325V - 1.325V. String equivalent resistance at each code is calculated using the MATLAB script given in the Appendix B. The script is based on the Delta-Y transformation of the following model of string taps in Figure 3.4.

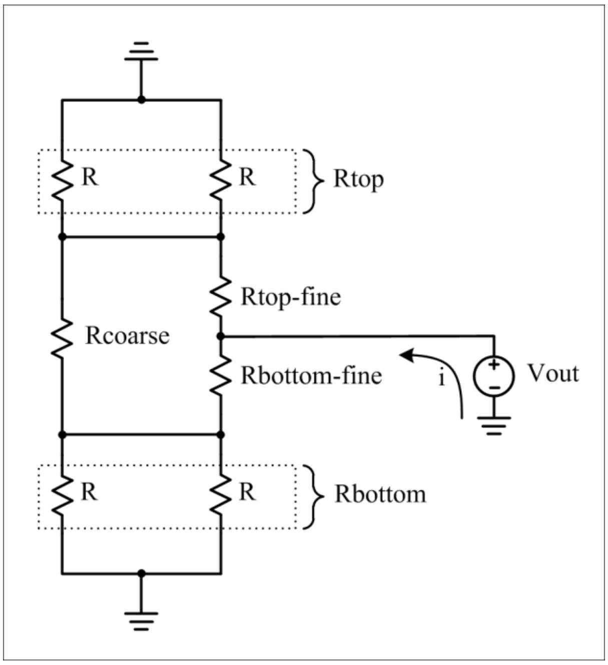


Figure 3.4 : String node equivalent resistance model.

String equivalent resistance calculated for each code is given in the following Figure 3.5.

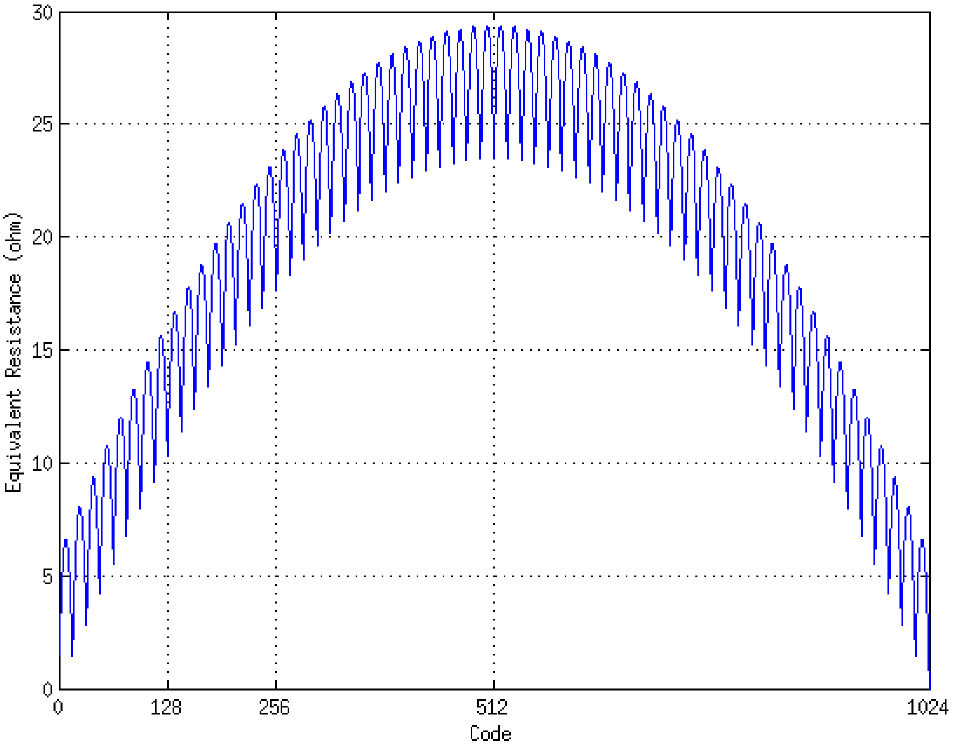


Figure 3.5 : String equivalent resistance per code.

Accuracy of this figure has been cross-checked and confirmed with a DC operating point analysis on the string schematic. Inverse of this equivalent resistance profile is series added before the first switch bank to raise, but balance the resistance seen for every code in the first stage, as shown in the Figure 3.6 below.

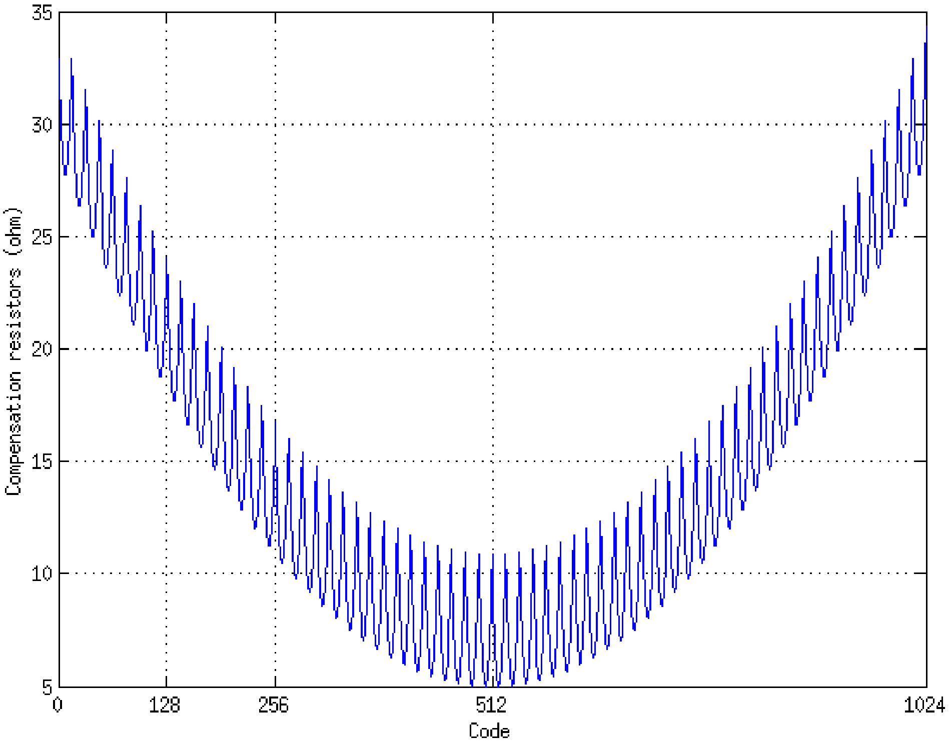


Figure 3.6 : String compensation resistors per code.

An SFDR comparison for a 1 MHz (f_0), 32 MHz (f_s) full-scale signal is given in Table 3.2.

Table 3.2 : String resistive compensation.

16-bit fully differential string architecture	**SFDR (dB)**
Without string compensation resistors	74.43
With string compensation resistors	**75.08**

The improvement above is admittedly very insignificant but the error mechanism as a whole is quite dominant. A more meaningful SFDR comparison with and without compensation resistors will be given in the next subsection where we address the compensation of switch resistances as well, which are the main resistance variation sources in the architecture.

Non-linear glitches on the string that cannot be completely eliminated by a fully differential architecture are a major source dynamic linearity errors. When a switch is turned off, a part of the charge injection is dumped over the resistor string. A common solution is to modify the switching orders of the switch banks in order to disconnect the DAC from the output bus until all node voltages settle. Waveforms from 2 examples of switching order experiments are given below in Figure 3.7 (make before break) and 3.8 (break before make) along with the glitch magnitude of the output bus.

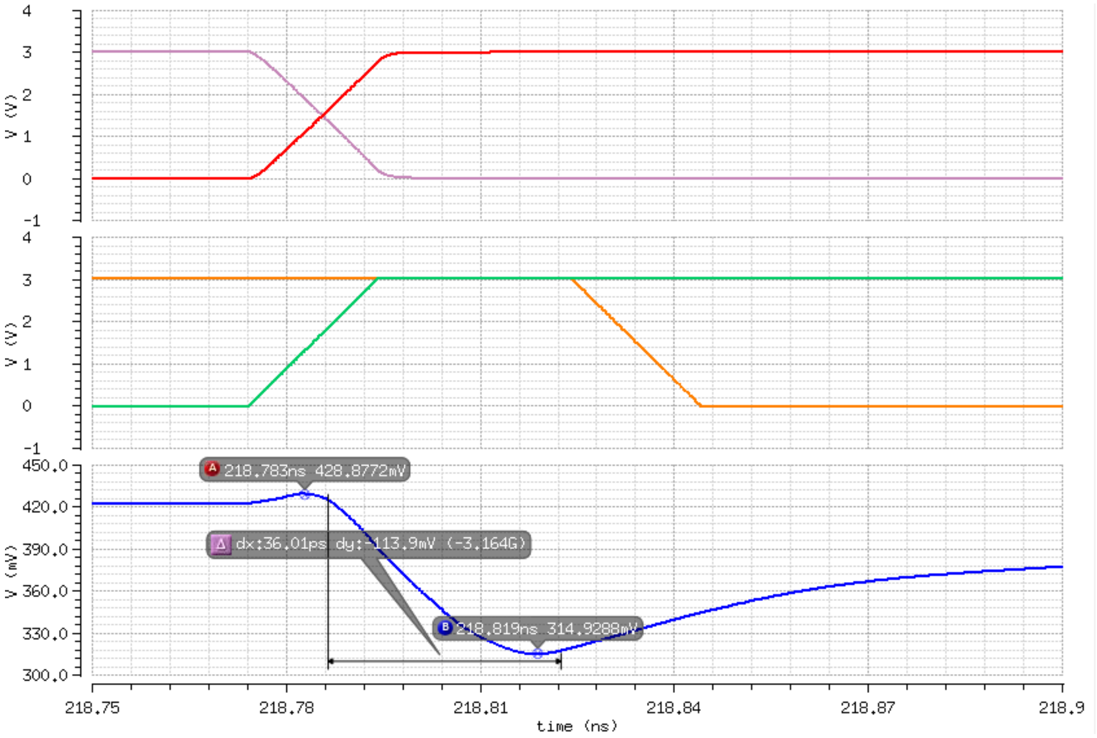


Figure 3.7 : a) no special switching, b) Make Before Break, c) output glitch.

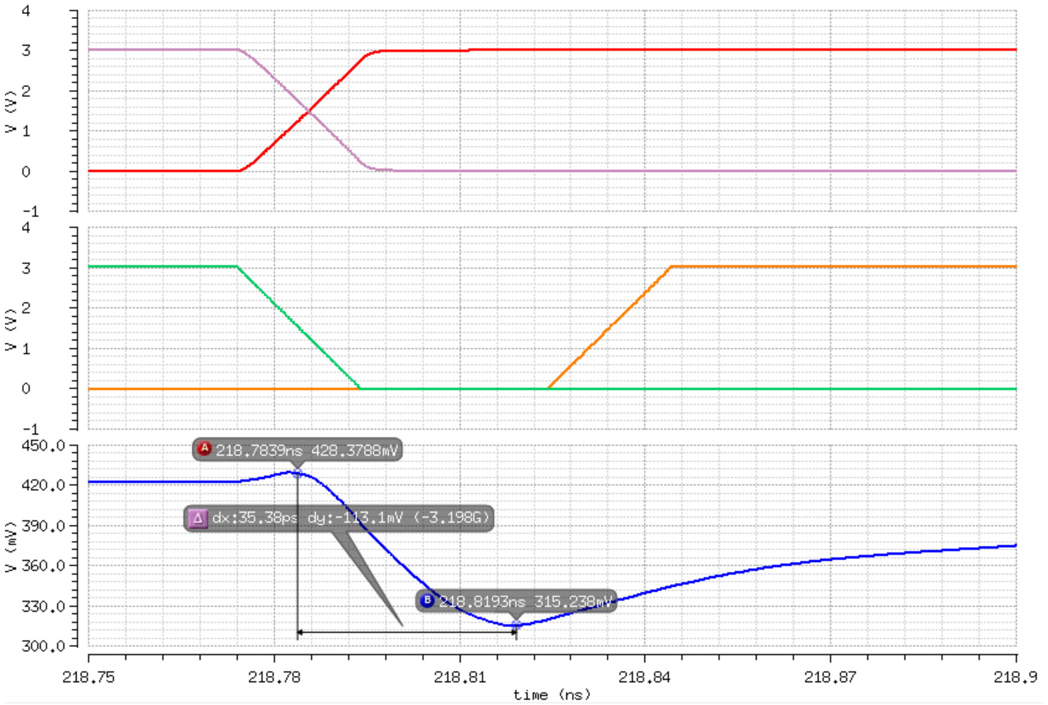


Figure 3.8 : a) no special switching, b) Break Before Make, c) output glitch.

Results of this experiment is given in Table 3.3.

Table 3.3 : Switching order and its effect on SFDR.

Architecture	SFDR (dB)
No special switching architecture	75.08
Make before break	**75.13**
Break before make	75.02

Glitch magnitude is at a level not affected by the switching order. Although simulations showed no significant effect, switching order is kept to be a make-before-break scheme for good practice.

Increasing the current on the string to reduce glitches proved to work very well as shown in Figure 3.9.

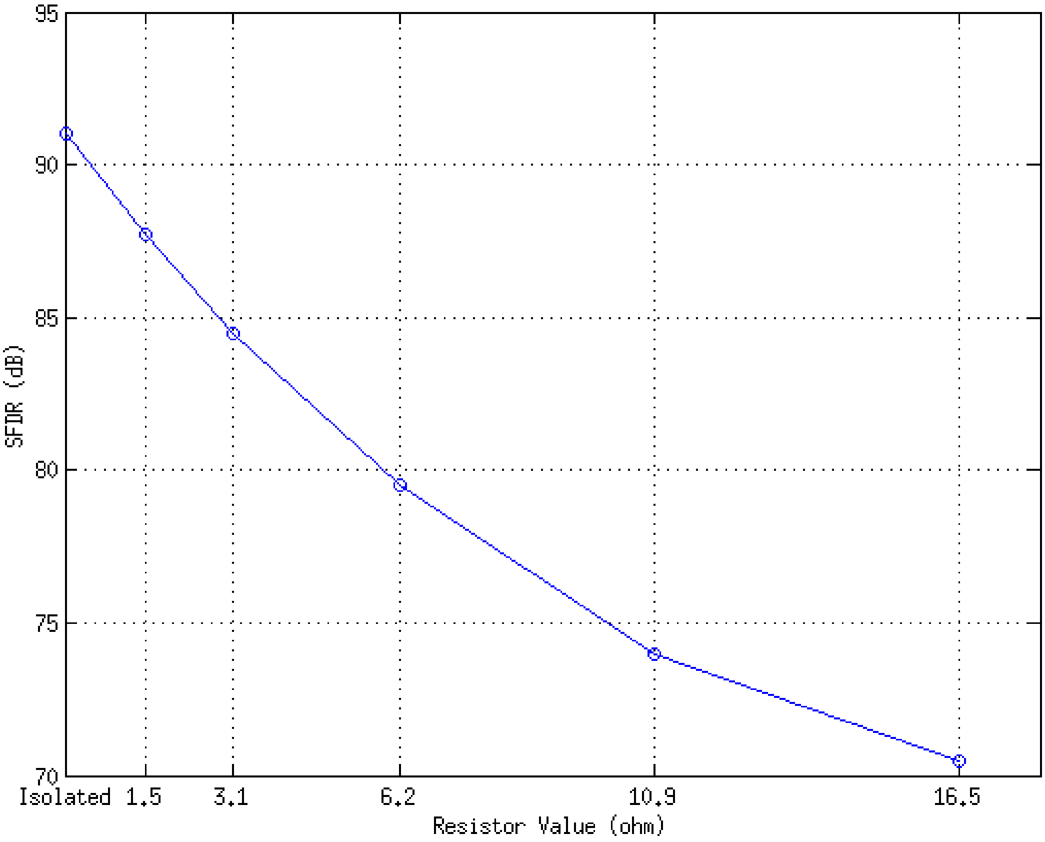


Figure 3.9 : SFDR vs. string resistance (1).

As it can be seen, for the proposed final architecture SFDR is 87.87 dB SFDR when the string is constructed with 1.5 Ω resistors giving a string current of 10 mA. Note that this figure reflects the performance of the final design, many aspects of the design are yet to be described in the following chapters.

3.3 Switches and Resistance Calibration

Switch sizes are primarily chosen to accommodate the simplified DAC model settling time requirement presented section 2.2.1.Third switch bank operates with 64 switches in parallel so it is reasonable to choose these switches to have minimum size (420n/350n) to reduce parasitic and terminal capacitances that will couple to the output bus. For the same reason, the high on resistance per switch will not affect the output bus resistance significantly. The said variation across the input range of the DAC for the minimum sized third switch bank is given below in Figure 3.10.

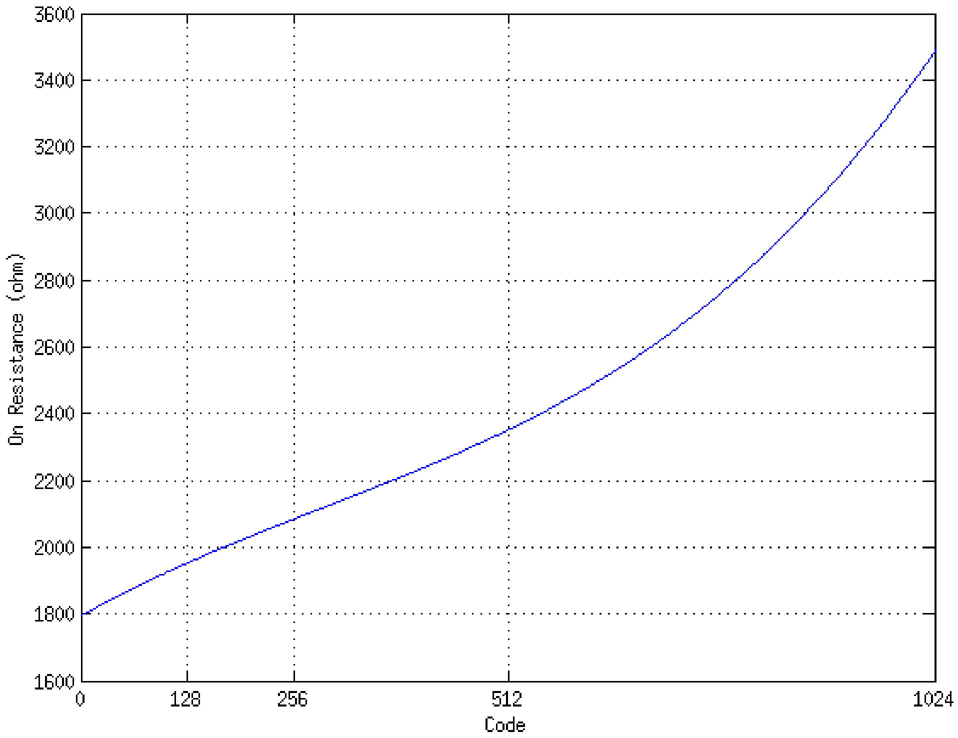


Figure 3.10 : SW3 on-resistance variation.

64 of these switches in parallel result in a 15 ohm variation across the string. Compensation resistors cannot eliminate this effect due to the fact that, for every MSB pattern, the third switch bank will be operating at a different node voltage. Efforts to reduce this effect by using a constant V_{gs} switching architecture results in insignificant performance improvement on a schematic level, with a reduced input range and a major increase in layout complexity. Plus, the OPAMP input capacitance seen by each switch is only 10 fF.

1st and 2nd level switches are connected in series and are therefore chosen to be much bigger (4u/350n) to reduce the baseline on resistance. A 5-bit model of our string-sw1-sw2 block of the DAC was constructed, and the on resistance of the switches ere measured for all codes. This profile was extrapolated to fit a 10-bit string, 1024 unique resistors to compensate for the variable string equivalent resistance was generated, and these resistors were placed as series resistors at each node of the string along with the string compensation resistors given in Figure 3.6. The on resistance profile and the resulting compensation resistors are given in Figure 3.11 and Figure 3.12 respectively.

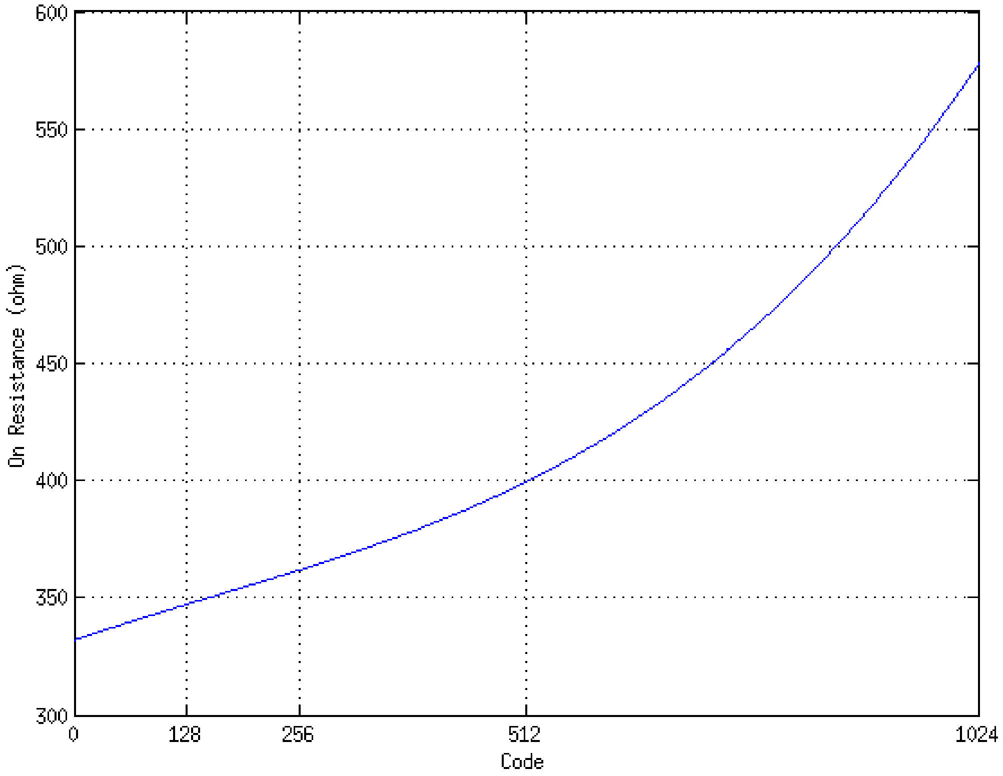


Figure 3.11 : SW1 and SW2 on-resistance variation.

Note that the compensation resistor profile for switch bank 1 and switch bank 2 are the same.

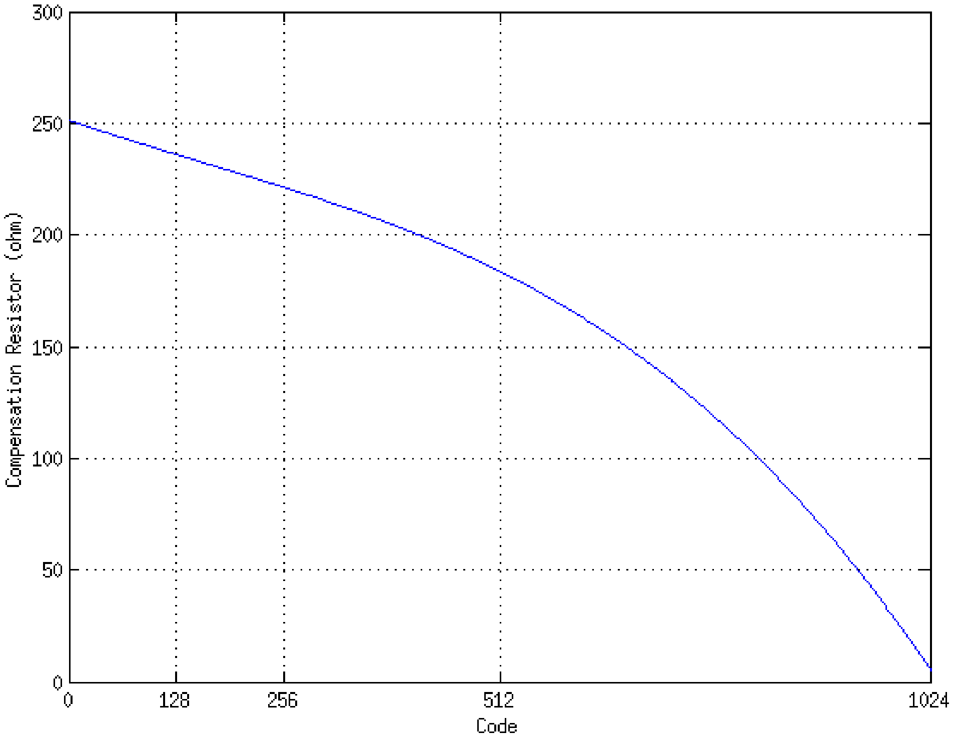


Figure 3.12 : Compensation resistors for SW1 and SW2.

Below in Figure 3.13 is the final compensation resistor profile, accounting for the string and the first two switch banks.

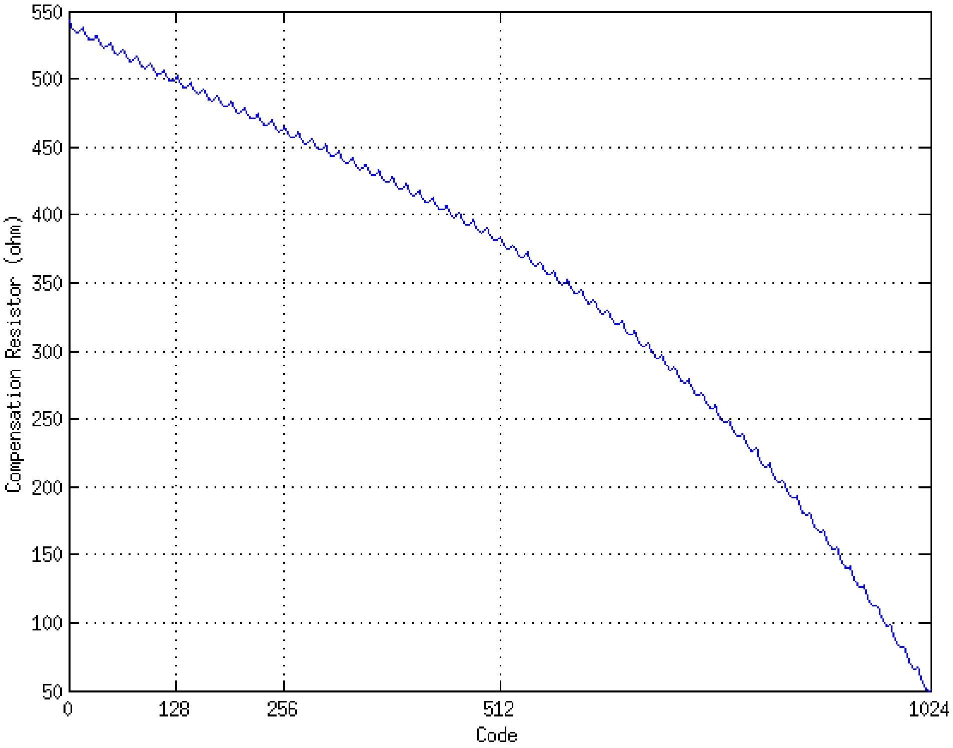


Figure 3.13 : Compensation resistors for string, SW1 and SW2.

Table 3.4 below shows comparison in SFDR when this set of compensation resistors is included in the design. Note that the following results are much lower than the ultimately achieved 87.5 dB SFDR - since many other design improvements are missing from this test-bench to only show the effect of resistive calibration.

Table 3.4 : Effect of compensation resistors on SFDR.

16-bit architecture with	**SFDR (dB)**
No compensation resistors	75.12
With compensation resistors	**78.78**

Second, and perhaps a more elegant way to compensate for switch bank nonlinearities (on resistance, charge injection, data feedthrough) is to use a constant V_{gs} switch architecture throughout the design as shown below in Figure 3.14 (figure shows the first switch bank, but the architecture is the same).

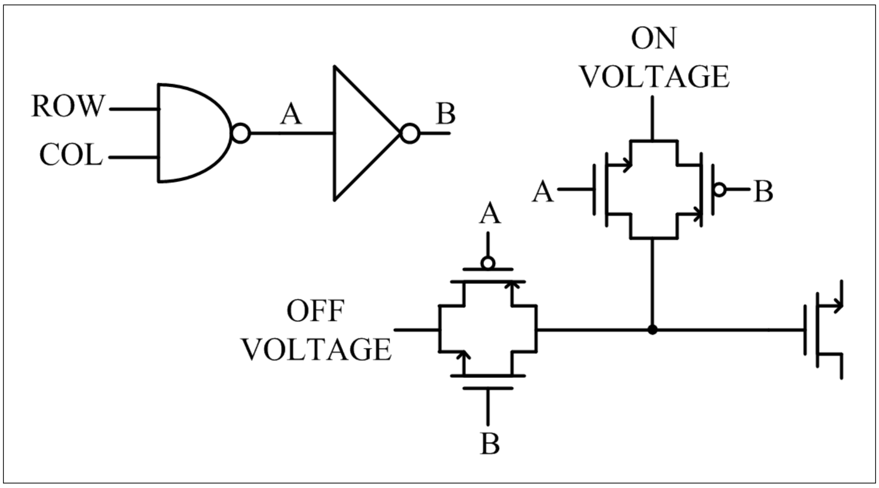


Figure 3.14 : Constant V_{GS} switch cell basic architecture.

Here, the switches at this bank are selected in an XY-decoded fashion with a NAND gate as explained in section 3.2. The NAND gate switches the true constant V_{gs} architecture that selects the main NMOS. Constant V_{gs} switches are transmission gates to accommodate for the range of the ON and OFF gate voltages. These voltages are selected from auxiliary strings that take their reference from the digitally programmable bandgap reference, just as the main string. For all switch banks to operate with constant V_{gs} voltages, 16 auxiliary string are required, all of which consists of 32 resistors. In addition to these auxiliary strings, such an architecture requires 2 separate switch banks (one PMOS, one NMOS) to feed the varying on and off voltages to the third switch bank. This solution’s main drawback is its voltage range limitations. Voltage ranges for the main string and constant V_{gs} auxiliary strings are determined experimentally to ensure all ON switches are in triode and all OFF switches are in cutoff in all possible code transitions as shown below in Table 3.5.kjhjkgkjhkjhkjhhkjh

Table 3.5 : DAC reference voltage ranges for constant V_{GS} switching.

String Range (V)	0.4 – 1
Vgs off Range (V)	0 – 0.6
Vgs on Range (V)	2.7 – 3.3

Allowing for true constant V_{gs} operation almost halves the range of the string (used to be 1 V), which worsens transient noise. Using 5 transistors instead of 1 for a single switch cell and 16 auxiliary strings is another problem, which becomes more apparent in the layout phase. Schematic level increase in performance given in Table 3.6 is practically the same as using compensation resistors, and it is an indicator that nonlinear charge injection and data feed-through are not dominant dynamic error mechanisms. In the experiment below, there are compensation resistors in place for the string but not for the switches.

Table 3.6 : Effect of constant V_{GS} switching on SFDR.

16-bit architecture with	**SFDR (dB)**
No special switching	75.12
Constant V_{gs} switching	**78.11**

Ultimately, this architecture is not used and is replaced by unique compensation resistors for the switch banks as explained in the previous subsection.

An experiment was conducted to get a better understanding of this tradeoff by editing out the overlap capacitances of all switches from the netlist in an effort to manually eliminate data-feedthrough. The results are below in Table 3.7. As it can be seen, code dependent data feedthrough seems to cause no performance problems.

Table 3.7 : Effects of data feedthrough on SFDR.

16-bit architecture with	**SFDR (dB)**
Unmodified netlist	75.12
Overlap capacitances manually removed	75.05

Figure 3.15 is a continuation of Figure 3.9, shows a comparison of using constant V_{gs} versus using compensation resistors to balance varying switch on resistances within the final design.

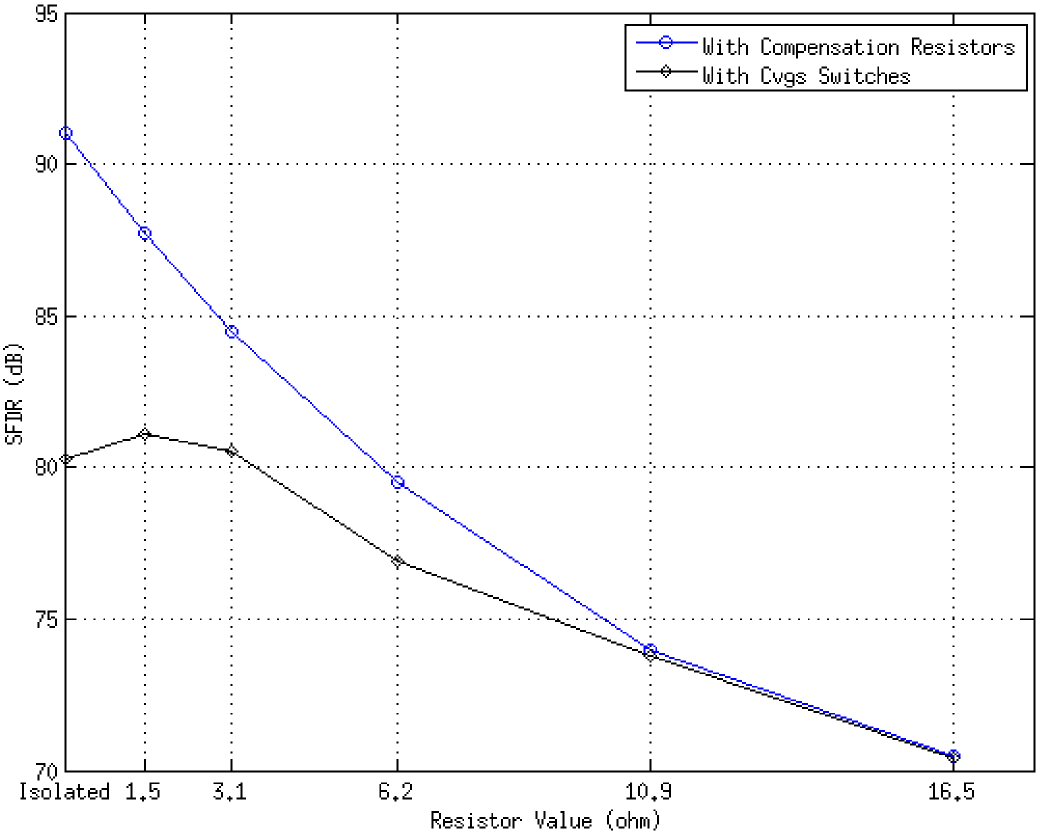


Figure 3.15 : SFDR vs. string resistance (2).

As it can be seen, the proposed resistive compensation architecture performs much better as a part of the whole design.

3.4 Interpolating OPAMP, Velocity Saturation, Capacitive Calibration

The OPAMP architecture used for this design is a PMOS folded-cascode, gain-boosted, Monticilli Class-AB output as shown below in Figure 3.16.

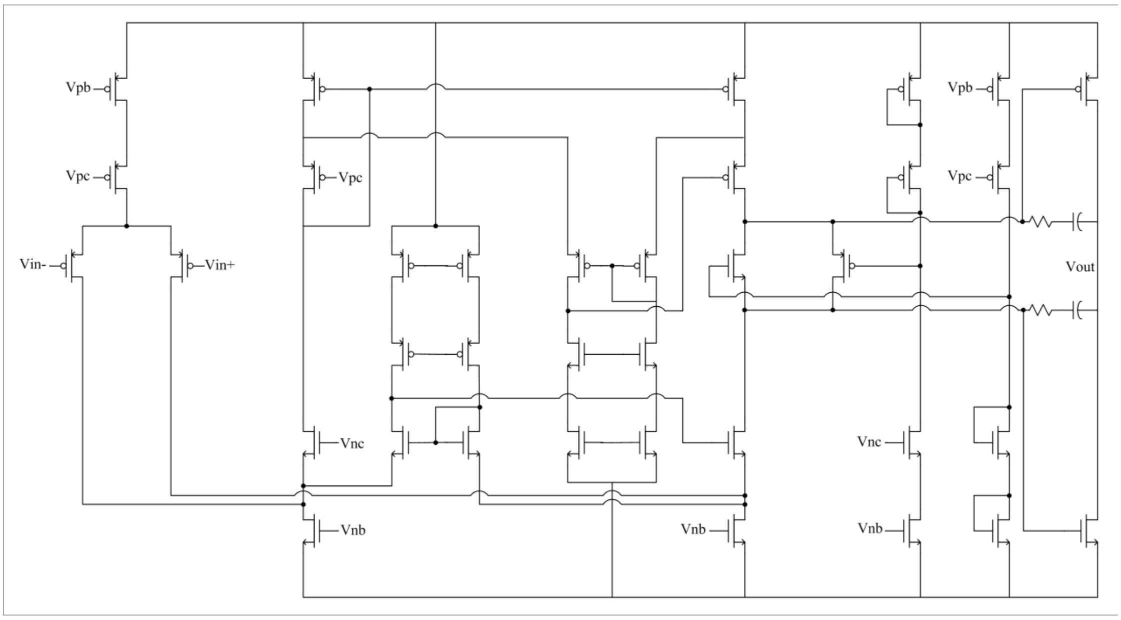


Figure 3.16 : Interpolating OPAMP.

The positive input PMOS is separated into 64 gates to interpolate the thermometer decoded 6 LSBs supplied from the third switch bank. Class-AB output transistors are operated at velocity saturation to allow for high linearity. To achieve this, these two transistors alone sink/source ~15mA. Other branch currents are given below in Figure 3.17. In order to preserve high gain, gain boosting of 1st stage cascade transistors are used.

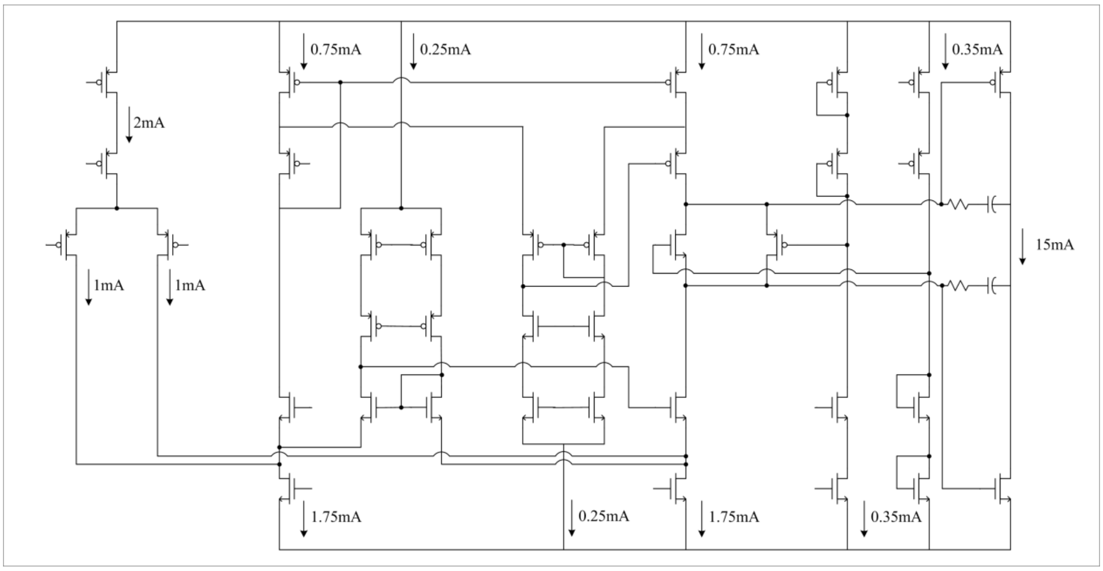


Figure 3.17 : Interpolating OPAMP currents.

Sweeping the input for the full scale and evaluating the operating regions for all the transistors determines the range of the OPAMP. This range is then optimized considering the design as a whole. Analysis of this range is given above in Figure 3.3. OPAMP operating range is determined to be 0.325 V - 1.325 V.

Frequency response of the 6-bit I-OPAMP for a 1 MHz (f_0), 32 MHz (f_s) full-scale signal is given below in Figure 3.18.

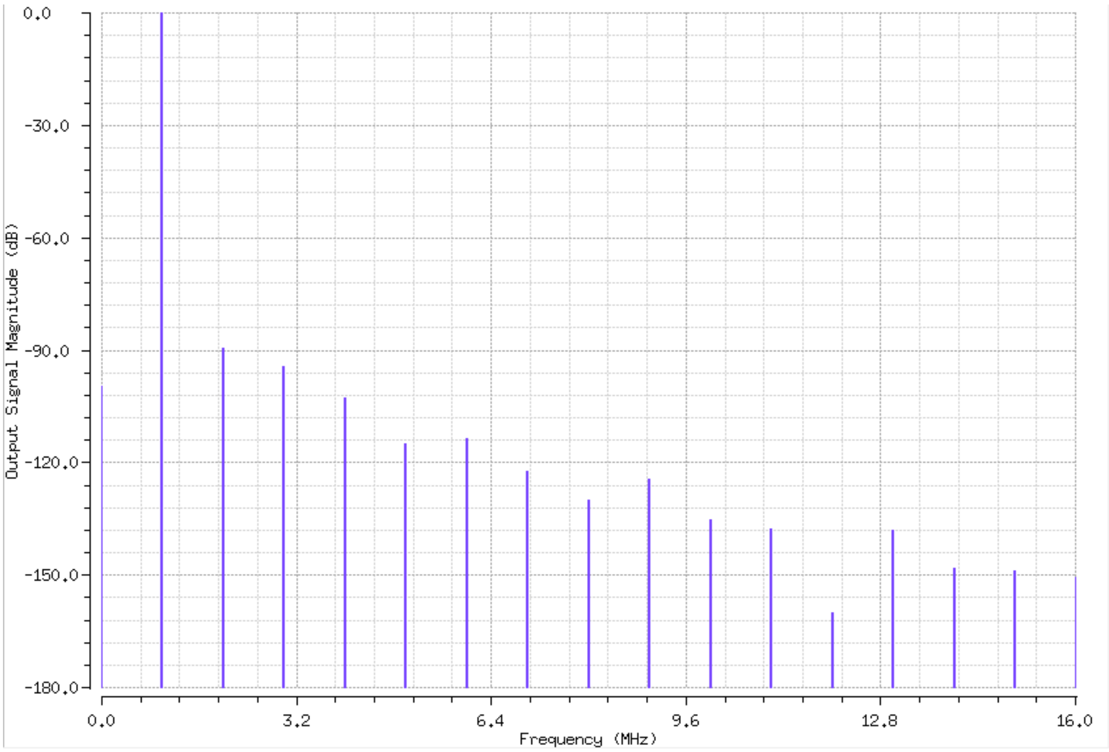


Figure 3.18 : Interpolating OPAMP frequency response (90 dB SFDR).

Driving the output transistors in their velocity saturation region (shown in Figures 3.19 and 3.20) by taking advantage of short channel effects is key to the performance of the OPAMP, and therefore the DAC.

Equation below shows that for short channel devices, I_D increases linearly with $V_{GS} - V_T$ rather than quadratically in the saturation region.

$$I_D = \mu_n C_{ox} \frac{W}{L} \left[(V_{GS} - V_T) V_{DS} - \frac{{V_{DS}}^2}{2} \right] \quad \textbf{(3.1)}$$

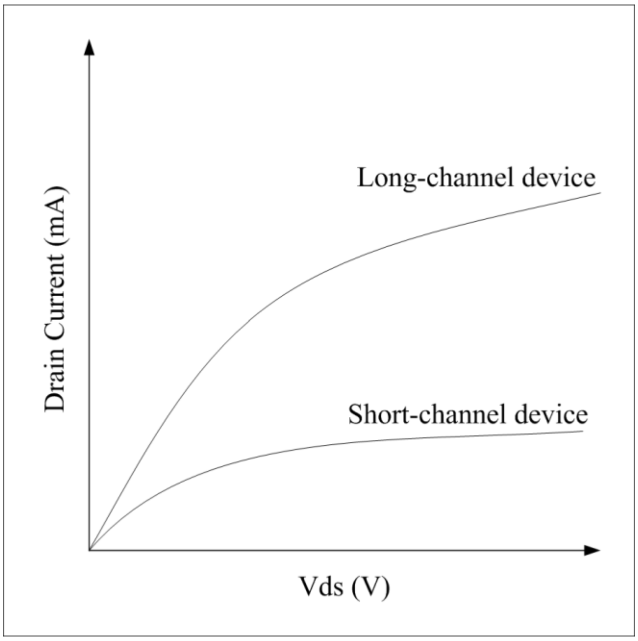


Figure 3.19 : Velocity saturation in short channel devices (I_D vs. V_{DS}).

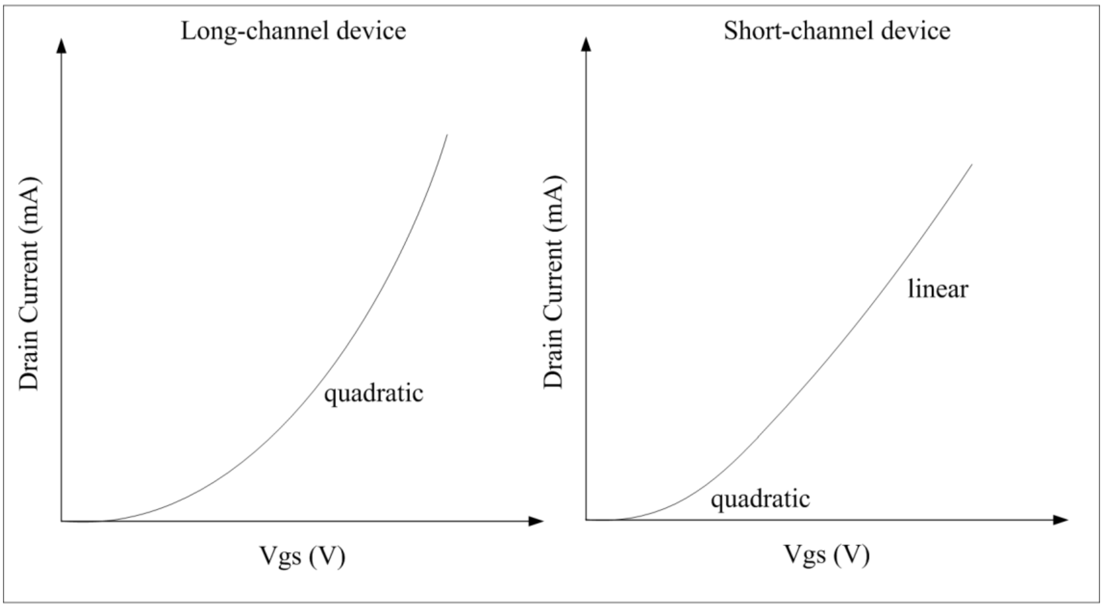


Figure 3.20 : Velocity saturation in short channel devices (I_D vs. V_{GS}).

As it can be seen, saturation region of short channel devices are more linear than for long-channel devices. Comparison of driving the output transistors in saturation versus velocity saturation is given in Table 3.8 for the same W/L ratio.

Table 3.8 : Short channel effects on SFDR.

OPAMP output stage transistor region	SFDR (dB)
Saturation (long-channel)	80.14
Velocity saturation (short-channel)	90.25

The last switch bank takes the 4 differential dual-outputs coming from the second switch bank, and creates 4 groups of 64 high or low node voltages to be resolved as 6 additional LSBs by the interpolating OPAMPs as shown below in Figure 3.21 and 3.22.

4 groups refer to the main output (VOUT), differential output (VOUT_D), and their inverses to be used by the dummy interpolating OPAMPs to balance load capacitance on the output busses for every code. At any time, somewhere between 0 and 64 switches connect the output busses to the interpolating OPAMP differential pairs for the unmodified architecture.

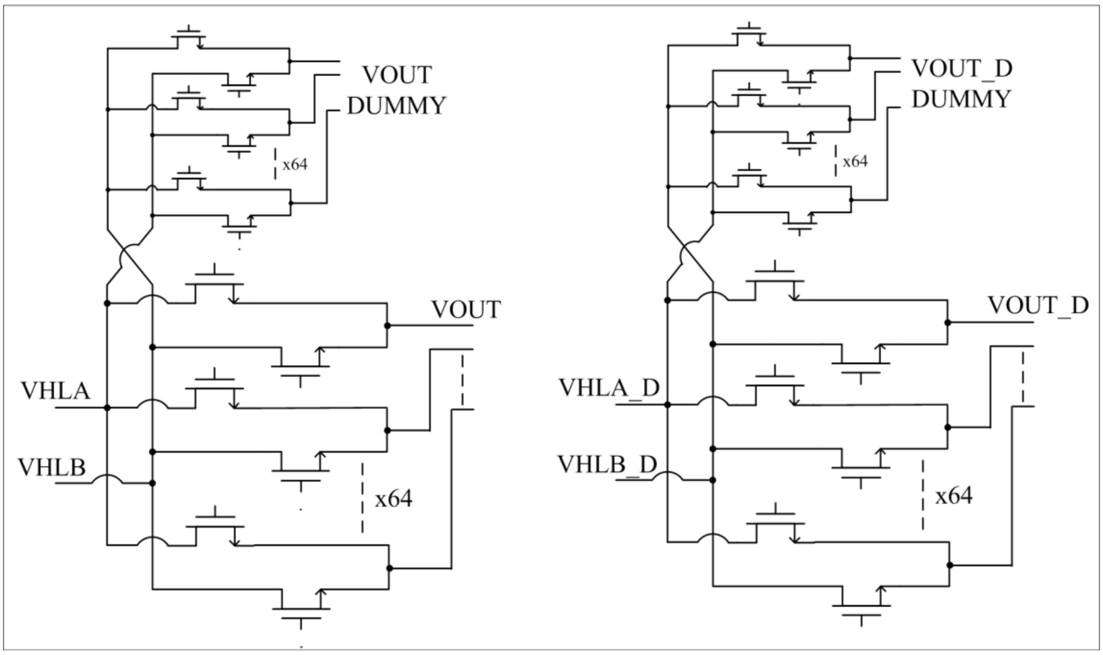


Figure 3.21 : Switch bank 3.

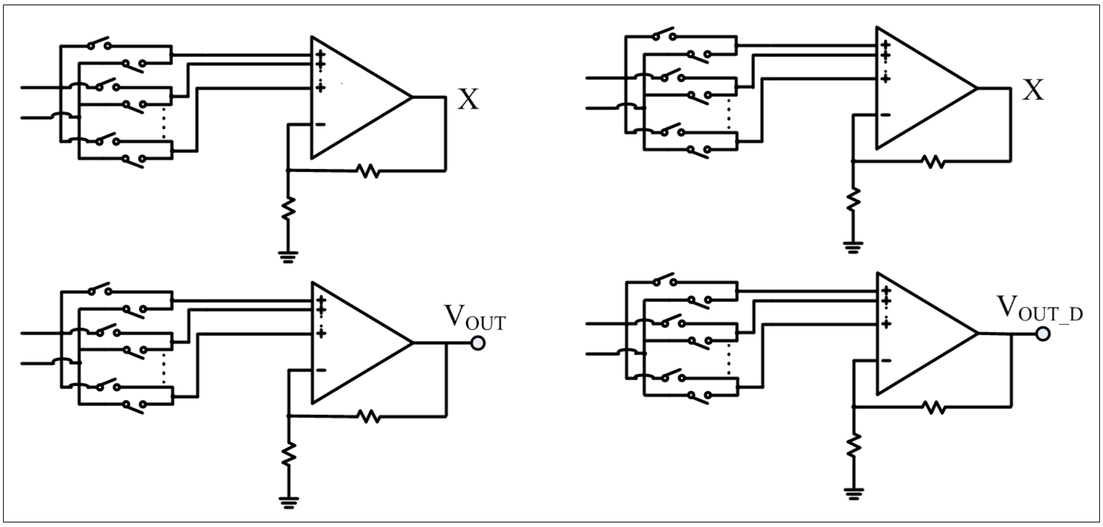


Figure 3.22 : Dummy interpolating OPAMPs and SW3.

With a dummy architecture employed, the capacitive load on the output bus is always from 64 switches and diff pair inputs which comes to about 640 fF. SFDR comparison of using dummy switches and OPAMPs are given below in Table 3.9.

Table 3.9 : Effect of dummy output stages on SFDR.

Final architecture	SFDR (dB)
Without capacitive compensation	78.78
With capacitive compensation	87.87

With the techniques outlined in the previous subchapters, SFDR for a 1 MHz (f_0), 32 MHz (f_s) full-scale signal is up to ~88 dB.

3.5 Pre-charge Stage and LSB Sensitivity

The following simulations concern a pre-charge block between first and second switch banks. At any point in time, 1 loop from the first stage is connected to the output bus via the second switch bank and the remaining 31 loops will float. The following Figure 3.23 shows this operation with just a few wires instead of all 32 for easy viewing. The top segment is the analog representation of the digital input. Middle segment show a few wires from VHLA (one of the output busses). As it can be seen, when the loop is connected, the signal follows the corresponding segment on the sine wave, and when it is disconnected it drops to some arbitrary voltage. The bottom segment on the plot are the corresponding looptaps of a few wires (same colors belong to same loops), which are not connected to any nodes.

Zooming into this plot in Figure 3.24 reveals that these nodes not only float at arbitrary voltages, but also drift due to charge leakage. Also there is quite a bit of glitches on these floating nodes due to switching activity on the string side.

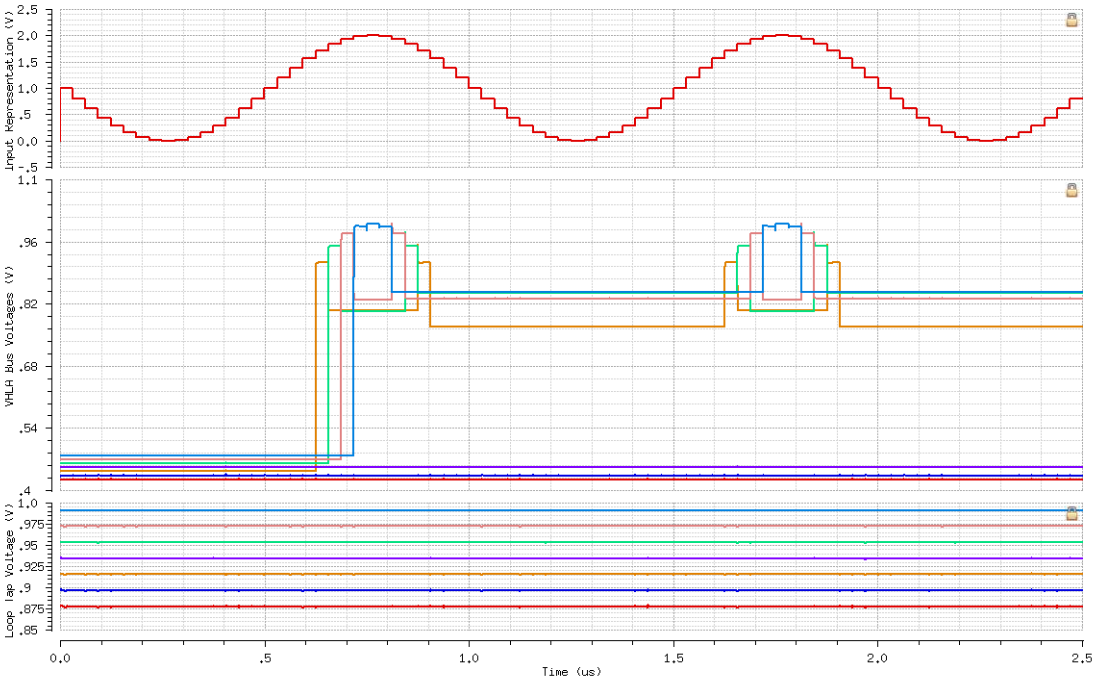


Figure 3.23 : Input to the second switch bank (no pre-charge).

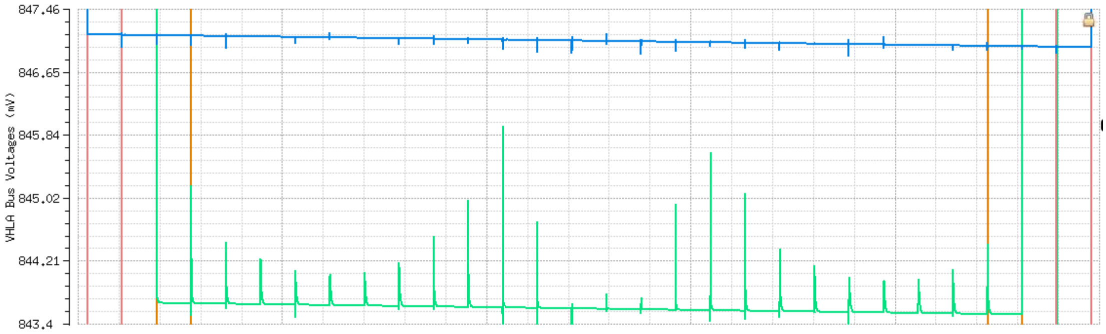


Figure 3.24 : Input to the second switch bank (no pre-charge) zoomed in.

Implementing a pre-charge stage as shown in Figure 3.25 connects the 31 floating loops to the tap voltages on the loops, therefore keeping the node ready at an appropriate voltage for an eventual connection. The following Figure 3.26 shows the same segment of the bus with a pre-charge stage in between stages.

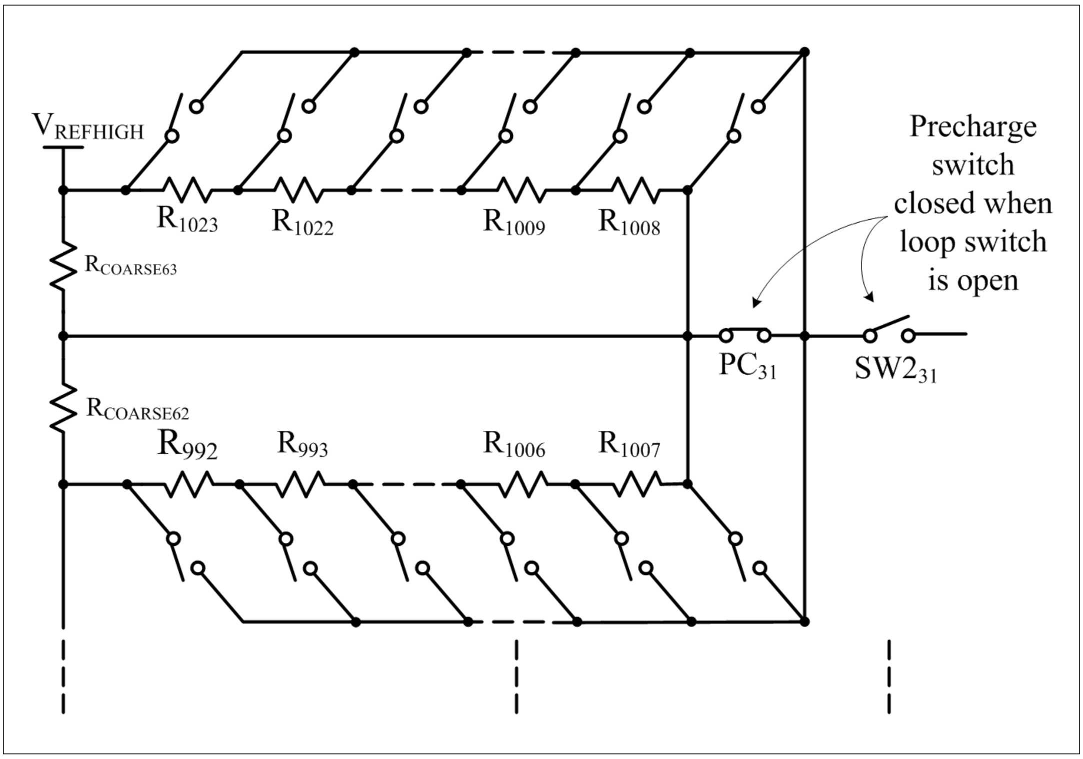


Figure 3.25 : Pre-charge switch placement.

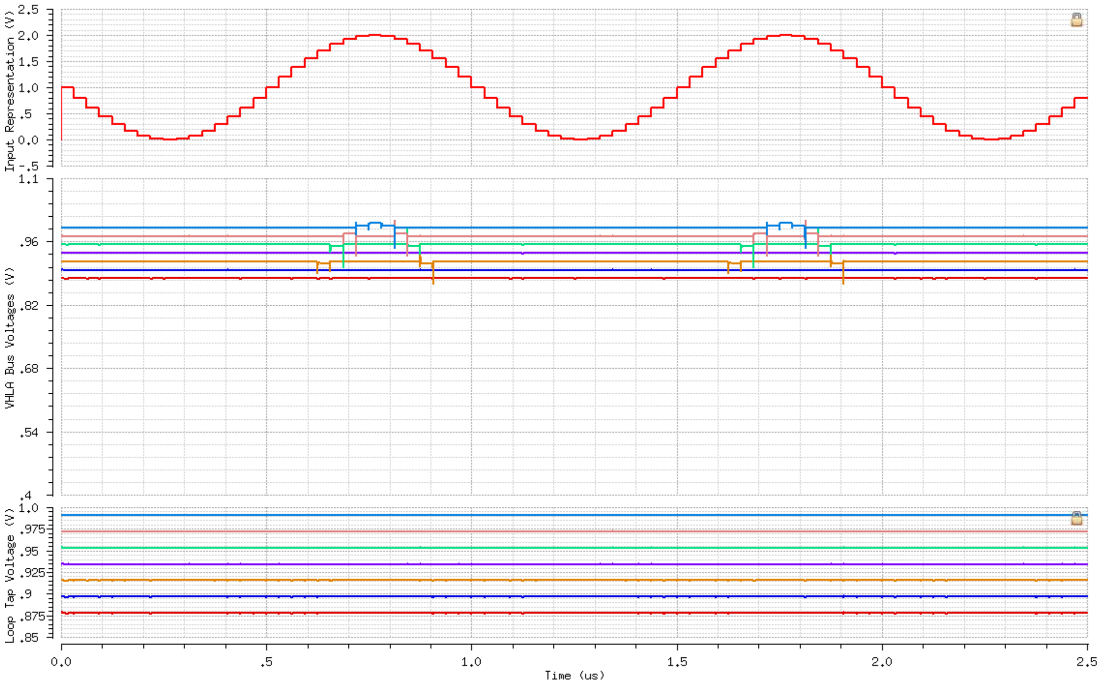


Figure 3.26 : Input to the second switch bank (with pre-charge).

Zooming into the same section in Figure 3.27 shows that disconnected loops remain at an appropriate and constant voltage until they are connected again. Glitches on disconnected nodes are largely reduced.

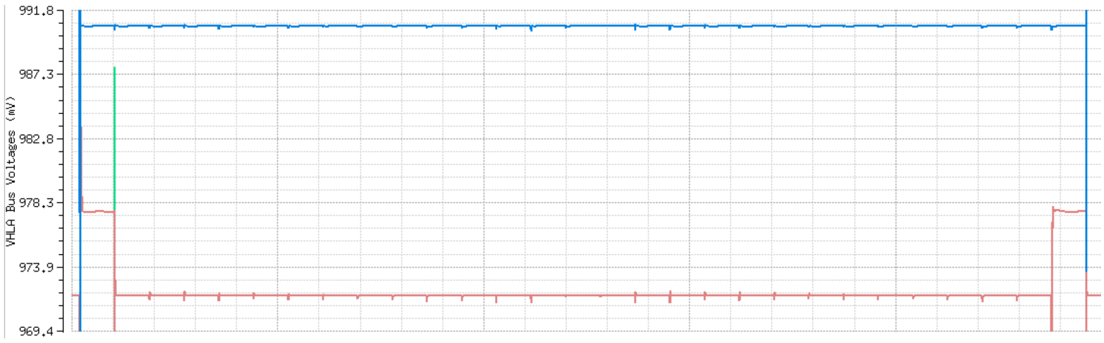


Figure 3.27 : Input to the second switch bank (with pre-charge) zoomed in.

The results from these two experiments tested for 8 LSB patterns in Table 3.10.

Table 3.10 : Effect of pre-charge on SFDR.

No Precharge	**SFDR (dB)**	**With Precharge**	**SFDR (dB)**
MEAN	87.87	MEAN	87.67
STD. DEV	2.53	STD. DEV	1.75

As it can be seen, a pre-charge stage doesn’t necessarily increase SFDR but majorly improves the LSB dependence. This can easily be explained by the fact that nonlinear glitches that are unique to different LSB not addressed by the pre-charge stage cause a larger deviation in SFDR results.

3.6 Instrumentation Amplifier

The final design at this point has a differential output which is the common mode of operation for DACs that are used as a part of a bigger mixed signal system. But for use as a discrete component, a single ended output is preferred. Therefore the final architecture is designed to offer both single ended and differential outputs. Figure 3.28 shows the 6-bit instrumentation amplifier used within the basic instrumentation amplifier configuration which is a common differential to single-ended converter architecture.

Compensation network of the OPAMPs in the first stage is optimized for the instrumentation amplifier configuration. This architecture by itself has practically the same SFDR as the differential output configuration, but within the design causes an added distortion as shown in Table 3.11. This is acceptable considering the single ended output is optional and the fact that any external differential-to-single-ended conversion would result in a similar performance drop anyway.

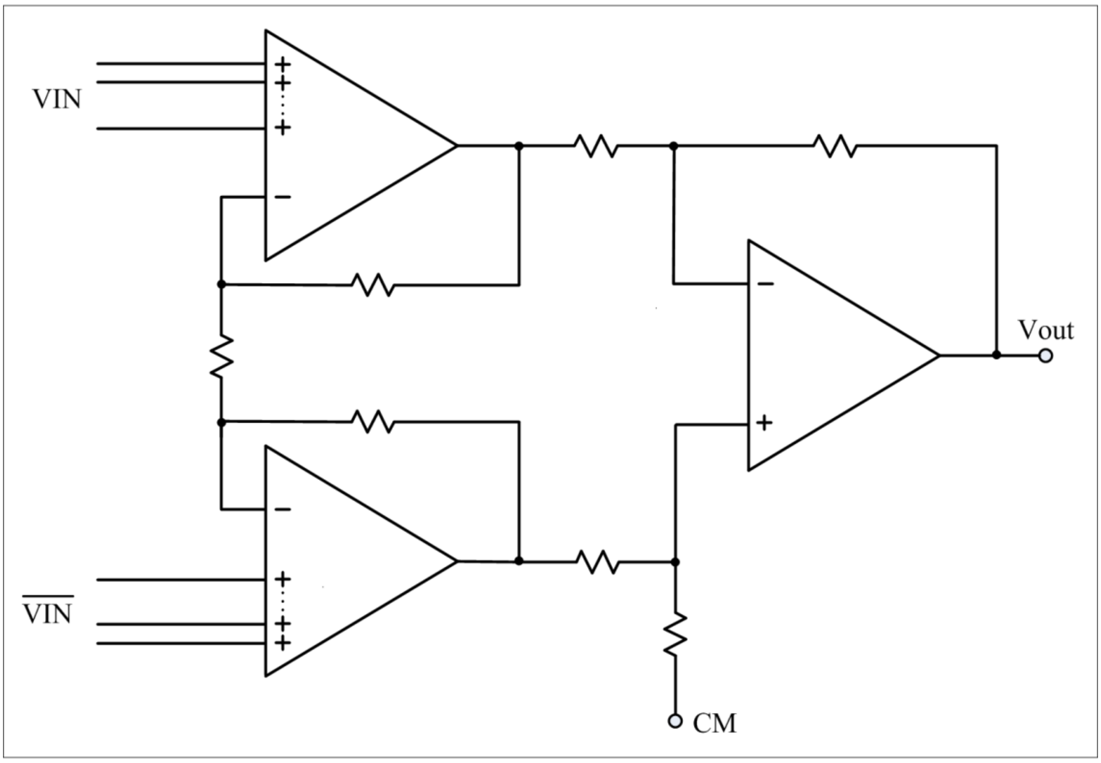


Figure 3.28 : Instrumentation amplifier configuration.

Table 3.11 : Instrumentation amplifier configuration.

Final architecture	SFDR (dB)
Differential output (previous sections)	87.87
Single-ended output (instrumentation)	84.37

3.7 Voltage References

Voltage references are supplied from a 8-bit digitally programmable internal band-gap reference with the Brokaw structure as the band-gap core as shown in Figure 3.29. OPAMP bias currents are generated with a 0tc architecture to ensure a 60ppm 1.24 V nominal output. All required voltages can be selected from the resistor string simultaneously. Voltage reference outputs are: STRING_REF_TOP (1.325 V), STRING_REF_BOTTOM (0.325 V) and AVDD (3.3 V). Please note that this block is a part of our design lab’s standard cell library.

All results up to this point were simulated with ideal voltage references. Table 3.12 below shows the final design SFDR for a 1 MHz (f_0), 32 MHz (f_s) full-scale signal, with references driven with real buffers.

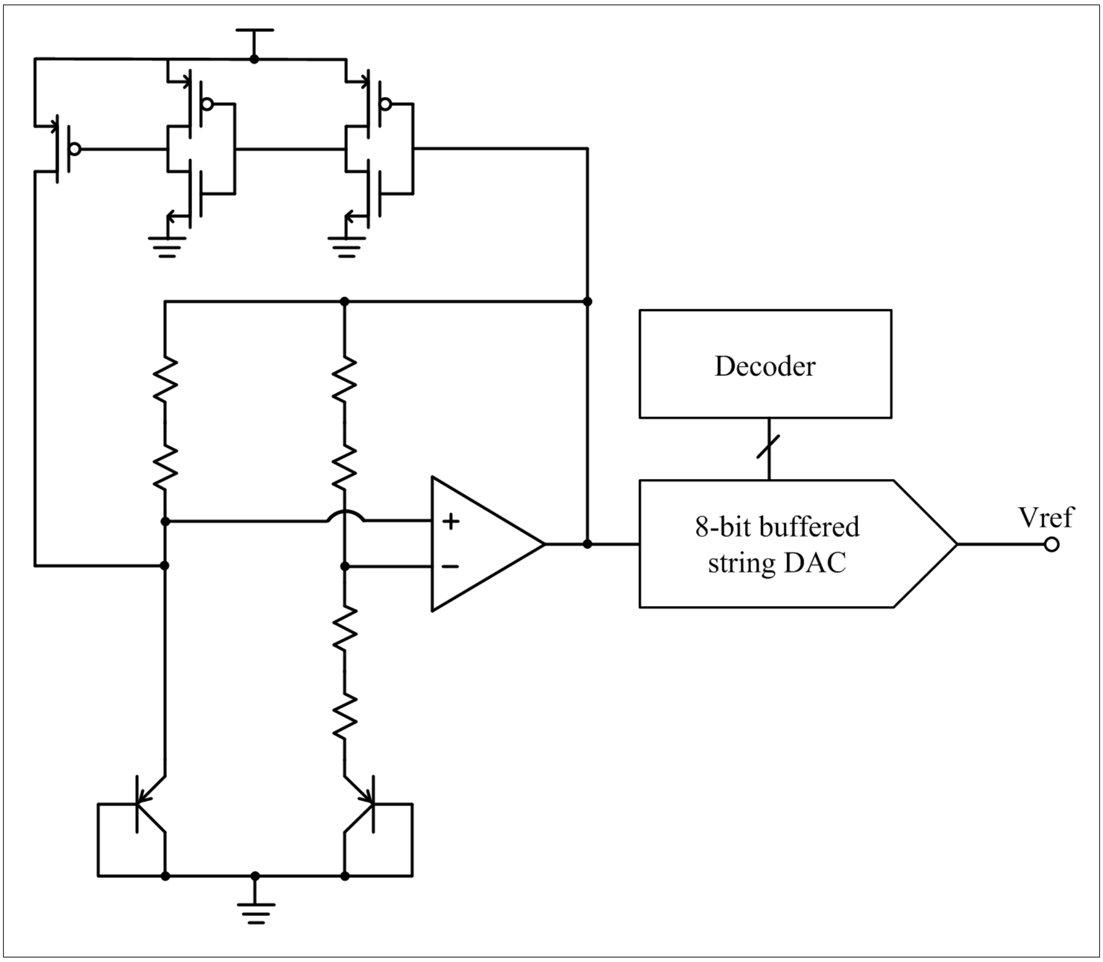


Figure 3.29 : Brokaw band-gap reference.

Table 3.12 : Real references.

Final architecture	**SFDR (dB)**
With ideal references	87.87
With real OPAMP driven references	87.04

A less than 1 dB drop with non-ideal voltage reference sources is perfectly acceptable.

3.8 Routine Simulations

In this section, results of a few routine simulations that operate the device at various conditions are presented.

3.8.1 Temperature sweep

The completed architecture SFDR vs. temperature is given below in Figure 3.30

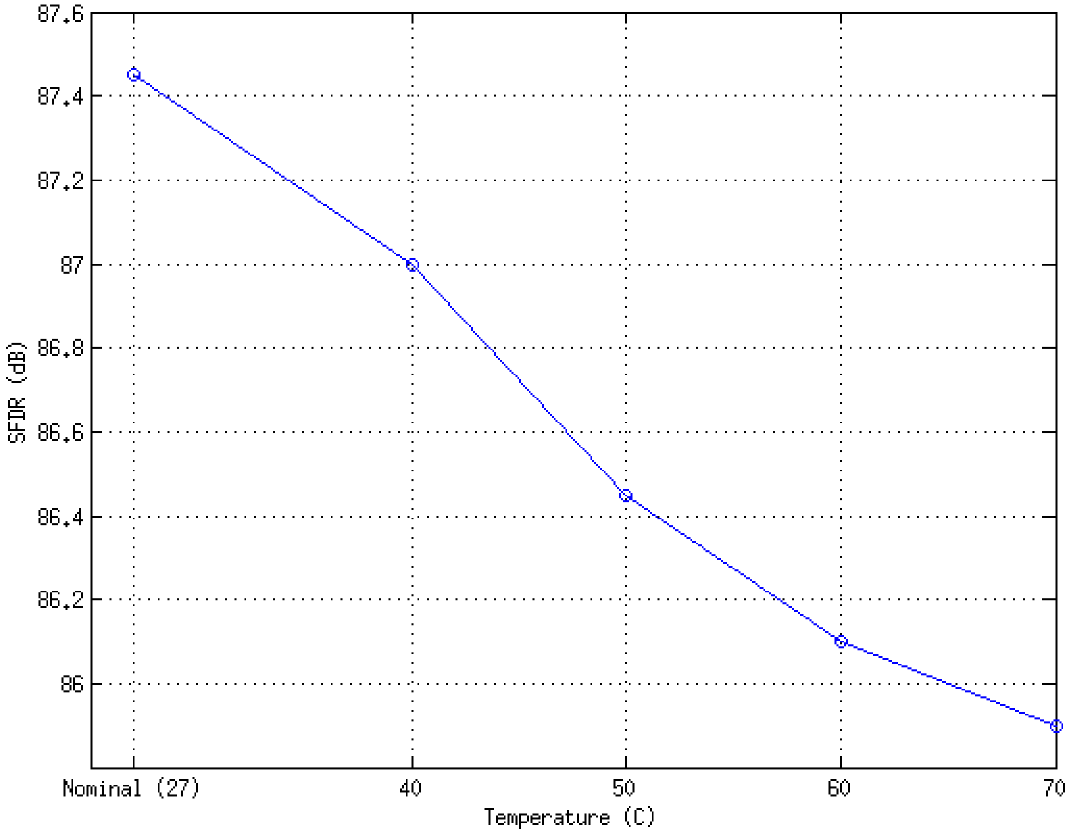


Figure 3.30 : SFDR vs. temperature.

3.8.2 Input frequency sweep

The completed architecture SFDR vs. input frequency is given below in Figure 3.31

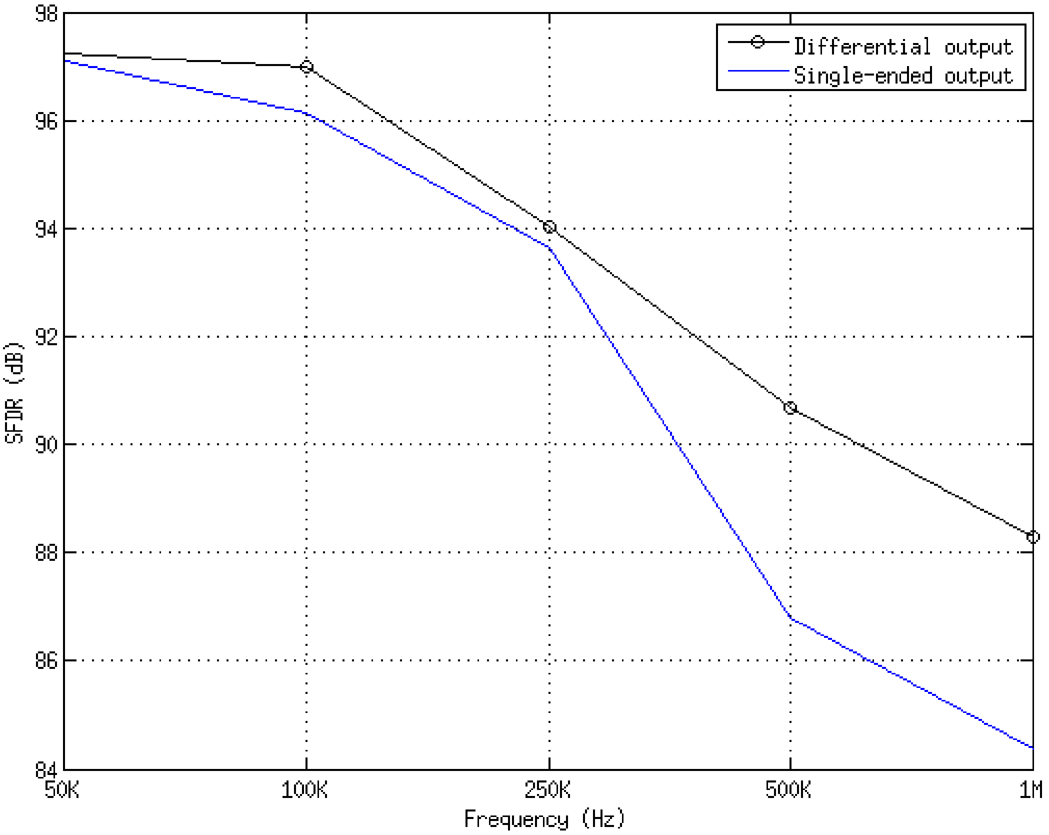


Figure 3.31 : SFDR vs. input frequency.

3.8.3 Output magnitude range sweep

The completed architecture SFDR vs. output magnitude and input common mode is given below in Figure 3.32 and Figure 3.33 respectively.

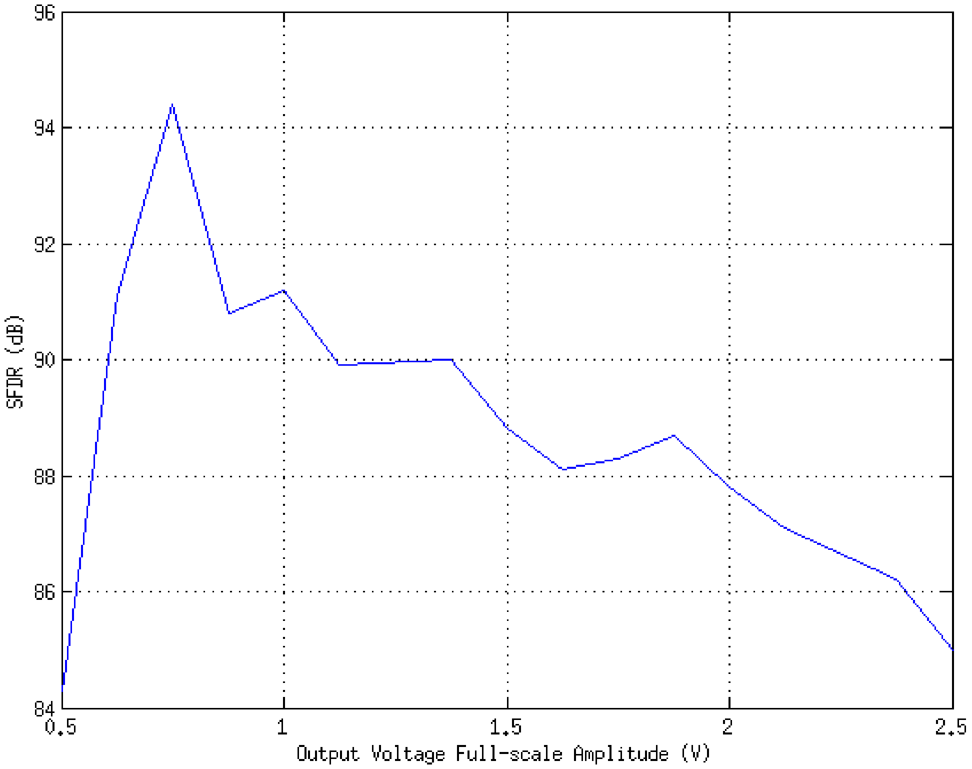


Figure 3.32: SFDR vs. output magnitude (nom: 2 V_{pp}).

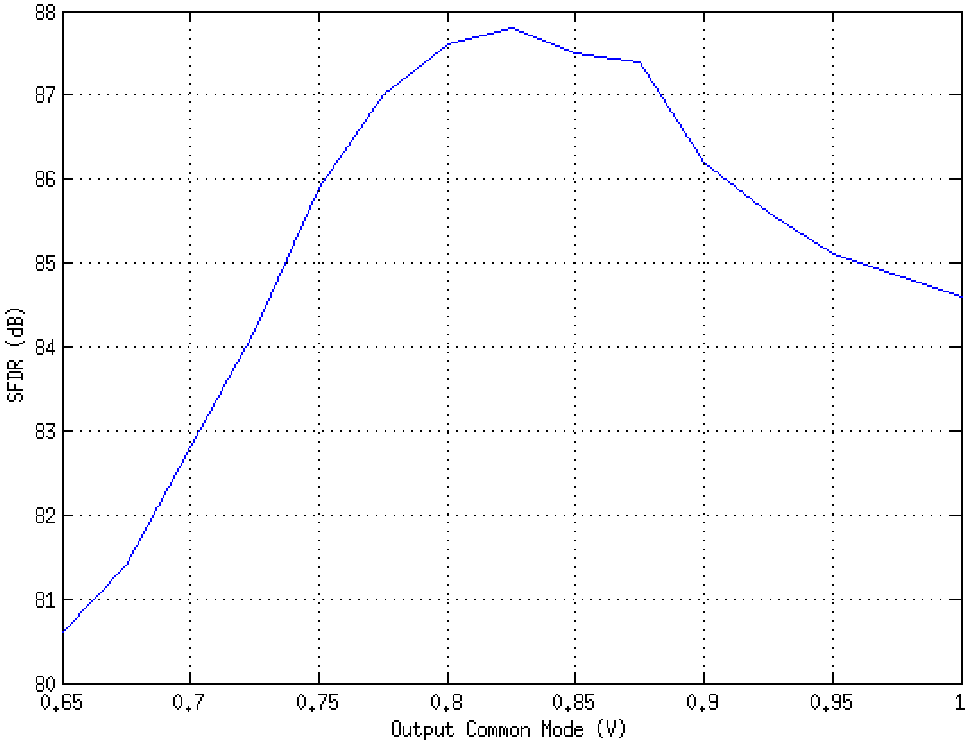


Figure 3.33 : SFDR vs. input common mode (nom: 0.825 V).

3.8.4 INL/DNL

Input for the integral non-linearity test is provided by an ideal 16-bit ADC written in VerilogAMS. Code transition happen every 1us which is ample time for 16-bit settling. Expression to generate the plots is given below.

$$INL = (VOUT_{IDEAL} - VOUT_{DAC})/1\,LSB \qquad \textbf{(3.2)}$$

$VOUT_{IDEAL}$ is generated by a 16-bit DAC written in VerilogAMS. The completed architecture INL is given below in Figure 3.34.

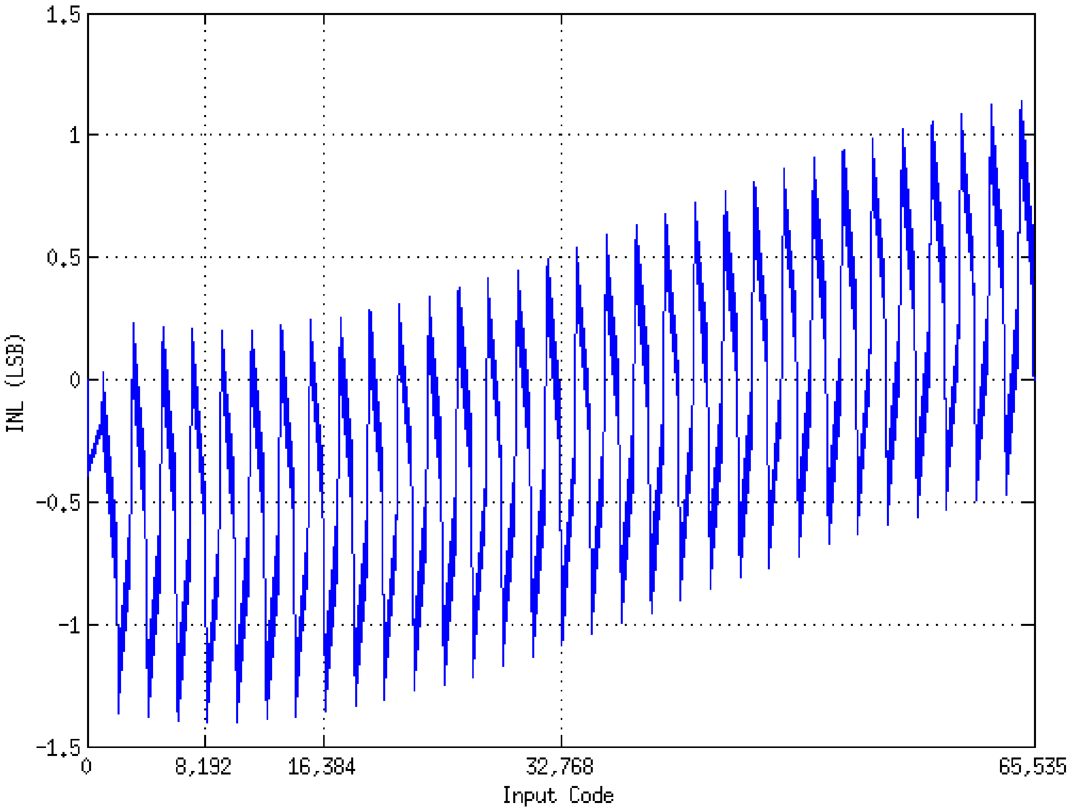


Figure 3.34 : 16-bit INL.

Note that, what seems like 1 LSB jumps are in fact integrated over hundreds of codes.

The completed architecture DNL is given below in Figure 3.35

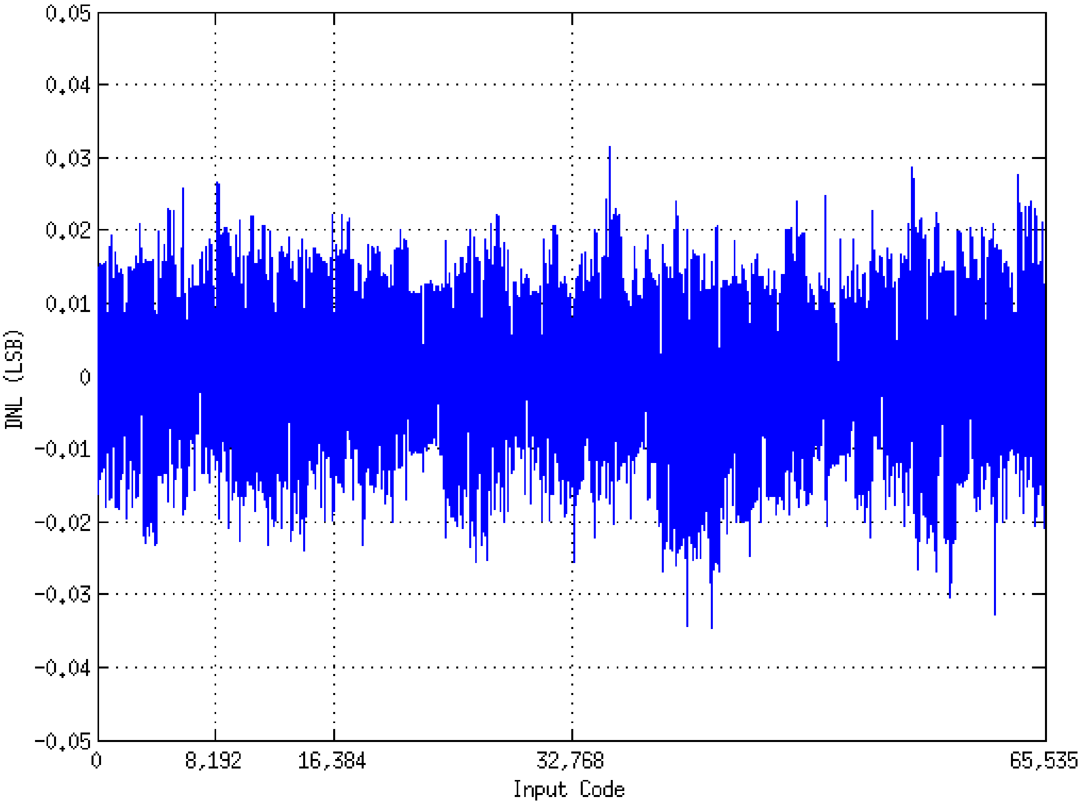


Figure 3.35 : 16-bit DNL.

3.8.5 Mismatch

Mismatch analysis for all switch bank, current source and differential pair transistors are given in this section. Unfortunately, resistor mismatch models for the resistors used in the design (N+ polys resistors with silicide) are not included in TSMC process models, but minimum resistor area criteria for true 10-bit performance is met. That is acceptable anyways, since DC trimming is assumed in this work.

The completed architecture mismatch histogram for 20 runs is given below in Figure 3.36.

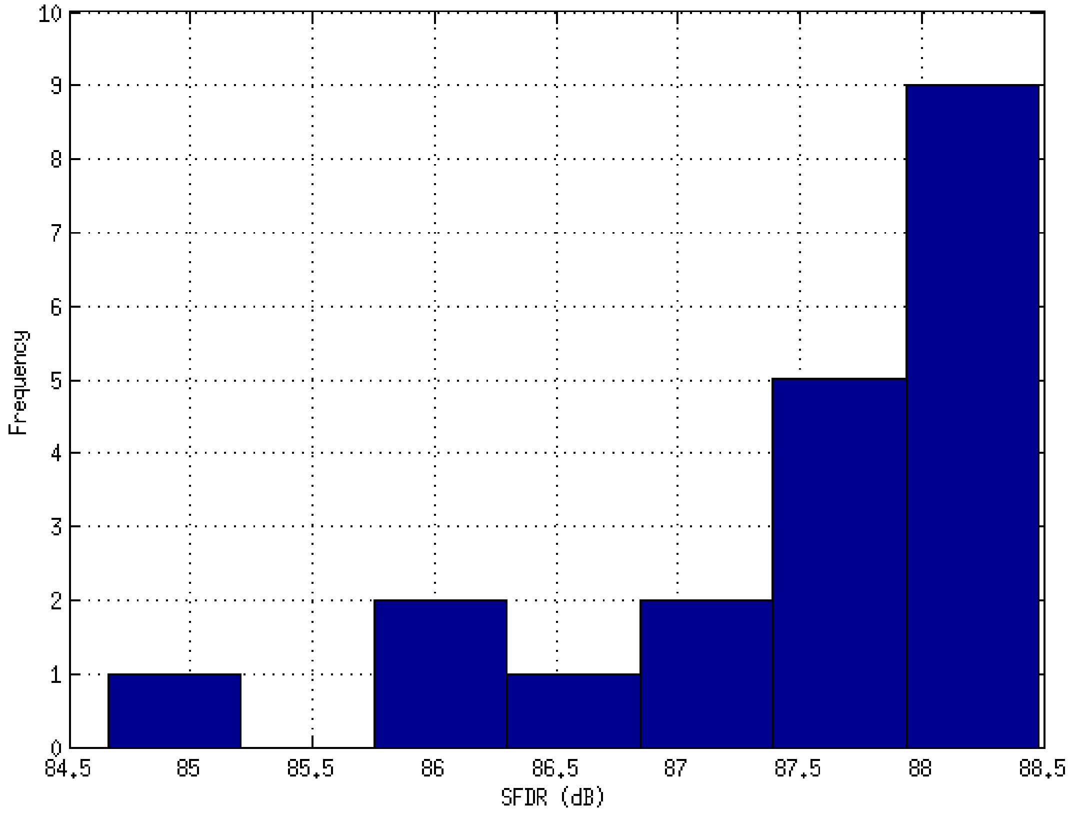


Figure 3.36 : Element mismatch SFDR histogram.

3.8.6 Corner analysis (schematic)

PVT corners analysis for SS, FF, TT process variation, -40-25-85 temperature variation and 3.2-3.3-3.4 supply voltage variation is given below in Table 3.13.

Table 3.13 : Schematic corners.

Corner	Supply (V)	Process	Temperature (°C)	SFDR (dB)
1 (N)	3.3	TT	27	88.74
2	3.3	FF	27	87.00
3	3.3	SS	27	80.25
4	3.3	TT	-40	90.00
5	3.3	FF	-40	85.87
6	3.3	SS	-40	86.82
7	3.3	TT	85	83.79
8	3.3	FF	85	85.72
9	*3.3*	*SS*	*85*	*77.26*
10	3.2	TT	27	83.93
11	3.2	FF	27	87.97
12	*3.2*	*SS*	*27*	*76.49*
13	3.2	TT	-40	91.26
14	3.2	FF	-40	88.25
15	3.2	SS	-40	81.13
16	3.2	TT	85	80.08
17	3.2	FF	85	86.68
18	*3.2*	*SS*	*85*	*74.12*
19	3.4	TT	27	87.34
20	3.4	FF	27	86.21
21	3.4	SS	27	84.17
22	3.4	TT	-40	89.02
23	3.4	FF	-40	84.43
24	3.4	SS	-40	90.59
25	3.4	TT	85	86.07
26	3.4	FF	85	84.96
27	3.4	SS	85	80.37

Most corners are satisfied for over 80 dB SFDR, except SS process corners at high temperature and low supply voltage (just below the 80 dB mark), which is acceptable. The design is optimized for the TT corner, which is statistically much more likely to occur than a SS corner. We did not want to maximize SS corner performance at the expense of TT corner performance. SFDR is maintained at over 80 dB at 85°C.

4. LAYOUT

4.1 Floorplan

Symbolic floor plan for the design is shown below in Figure 4.1. Approximate size of the blocks is in scale relative to each other. Size of the device as it is shown below is expected to be 1.75mm x1.75mm. Multiple incremental versions were made for every stage of this design over the past year. Layout screenshots in this report are the final iterations.

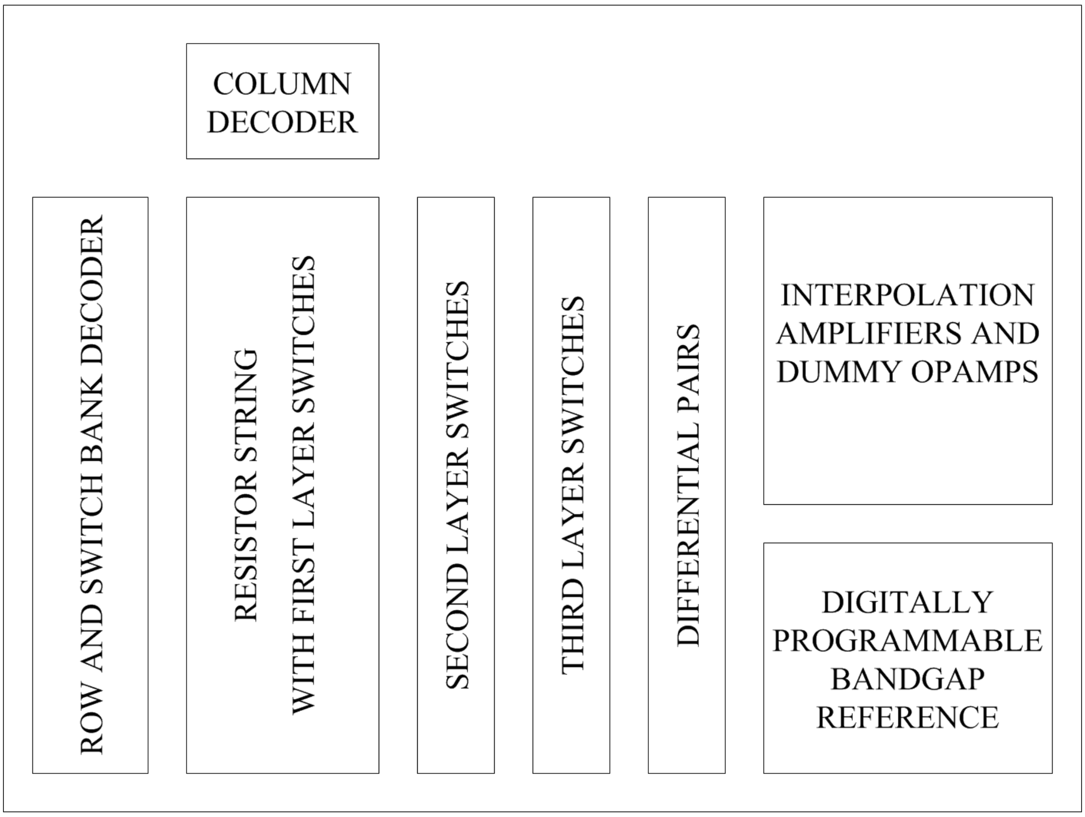

COLUMN DECODER
ROW AND SWITCH BANK DECODER
RESISTOR STRING WITH FIRST LAYER SWITCHES
SECOND LAYER SWITCHES
THIRD LAYER SWITCHES
DIFFERENTIAL PAIRS
INTERPOLATION AMPLIFIERS AND DUMMY OPAMPS
DIGITALLY PROGRAMMABLE BANDGAP REFERENCE

Figure 4.1 : Symbolic floorplan.

4.2 String and Switch Bank 1

Various views of the DRC and LVS clean finalized main string is shown below in Figures 4.2, 4.3, 4.4 and 4.5.

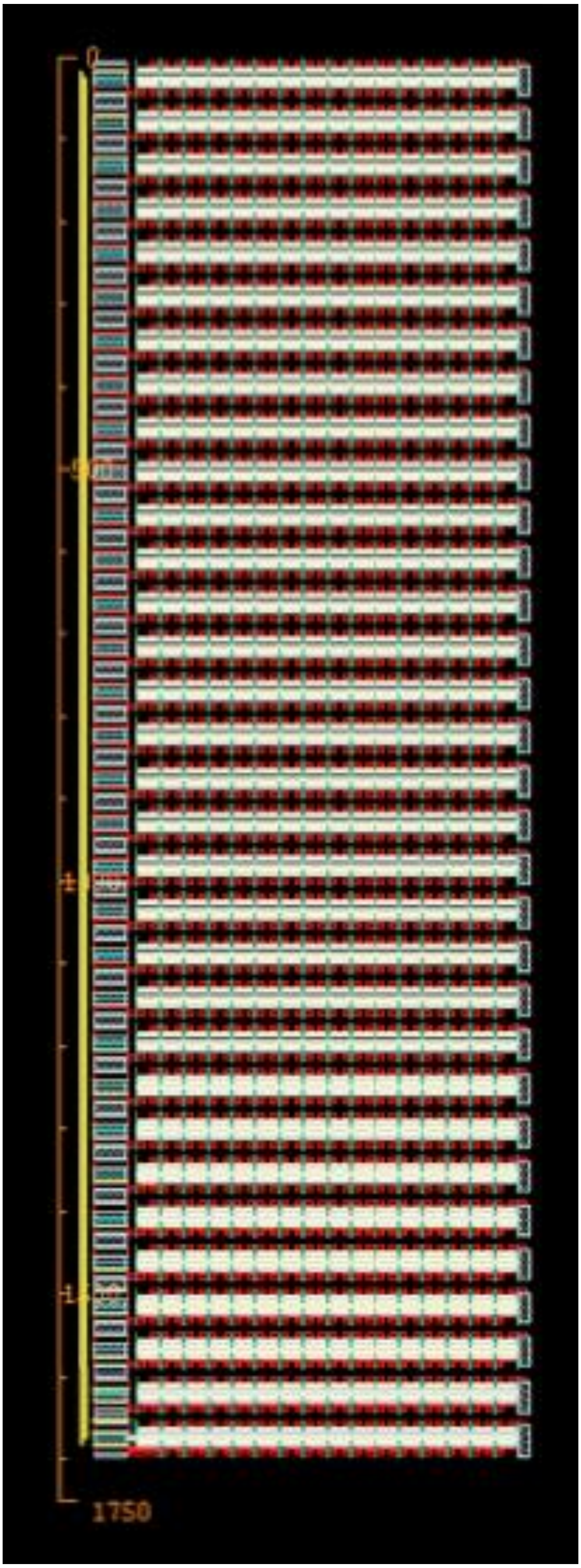


Figure 4.2 : Resistor string w/ compensation resistors and SW1.

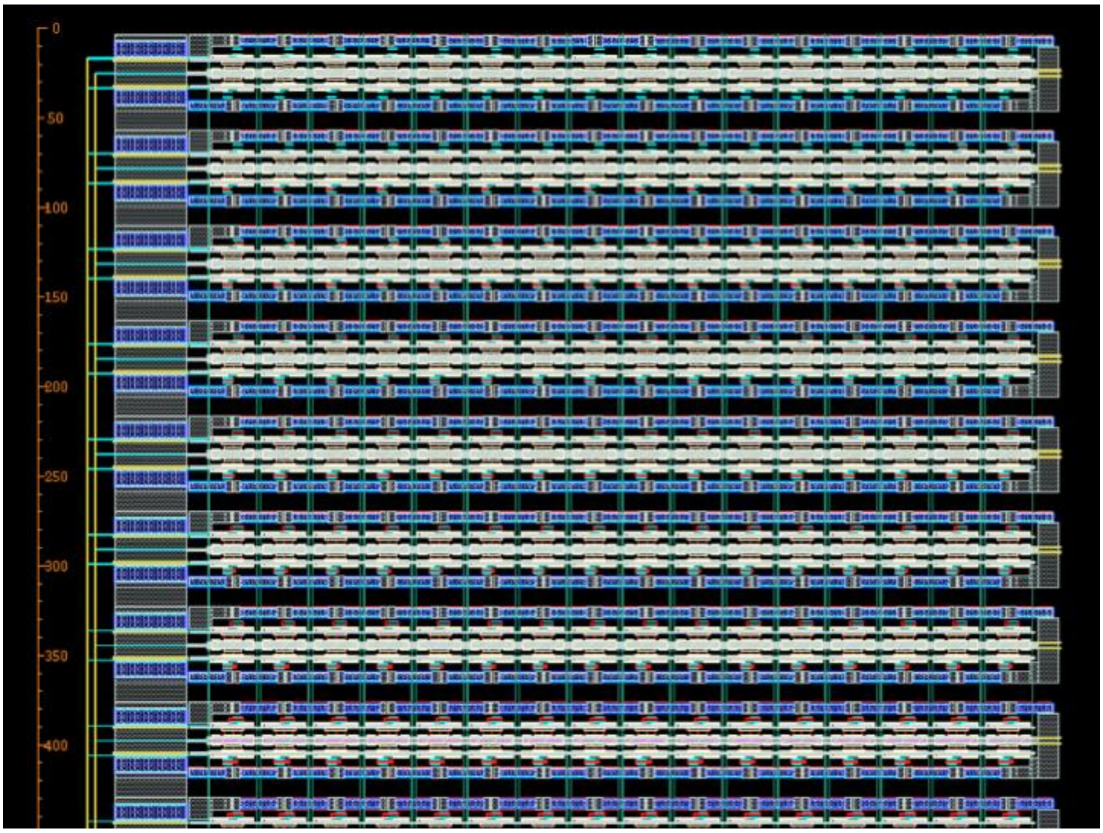


Figure 4.3 : Resistor string w/ compensation resistors and SW1 (zoom 1).

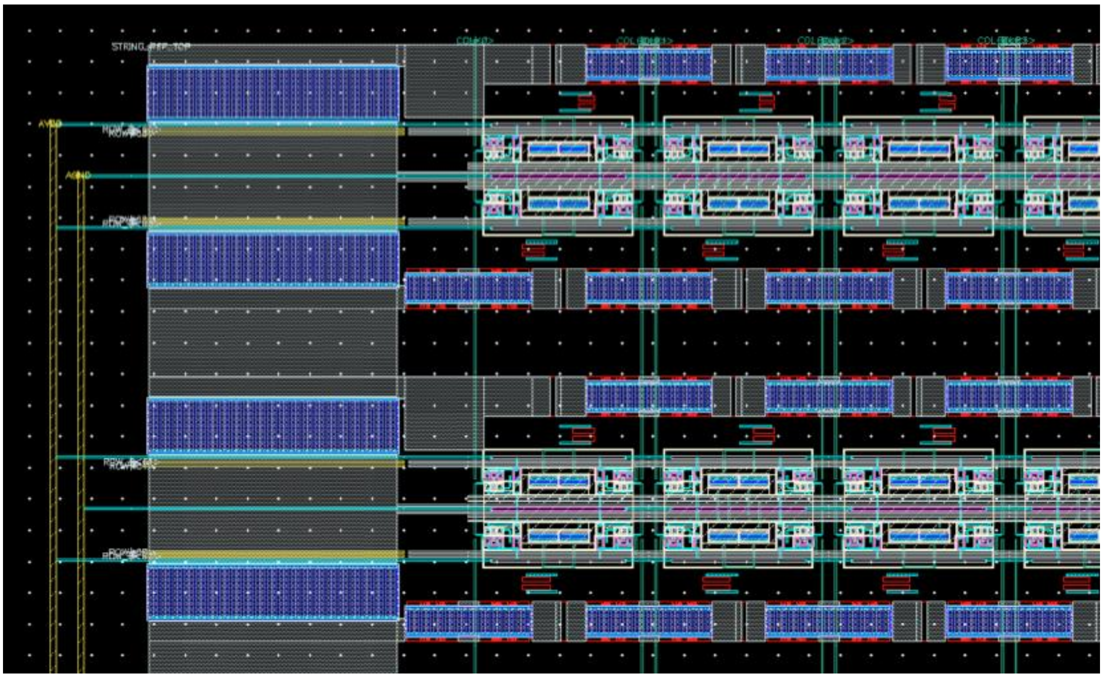


Figure 4.4 : Resistor string w/ compensation resistors and SW1 (zoom 2).

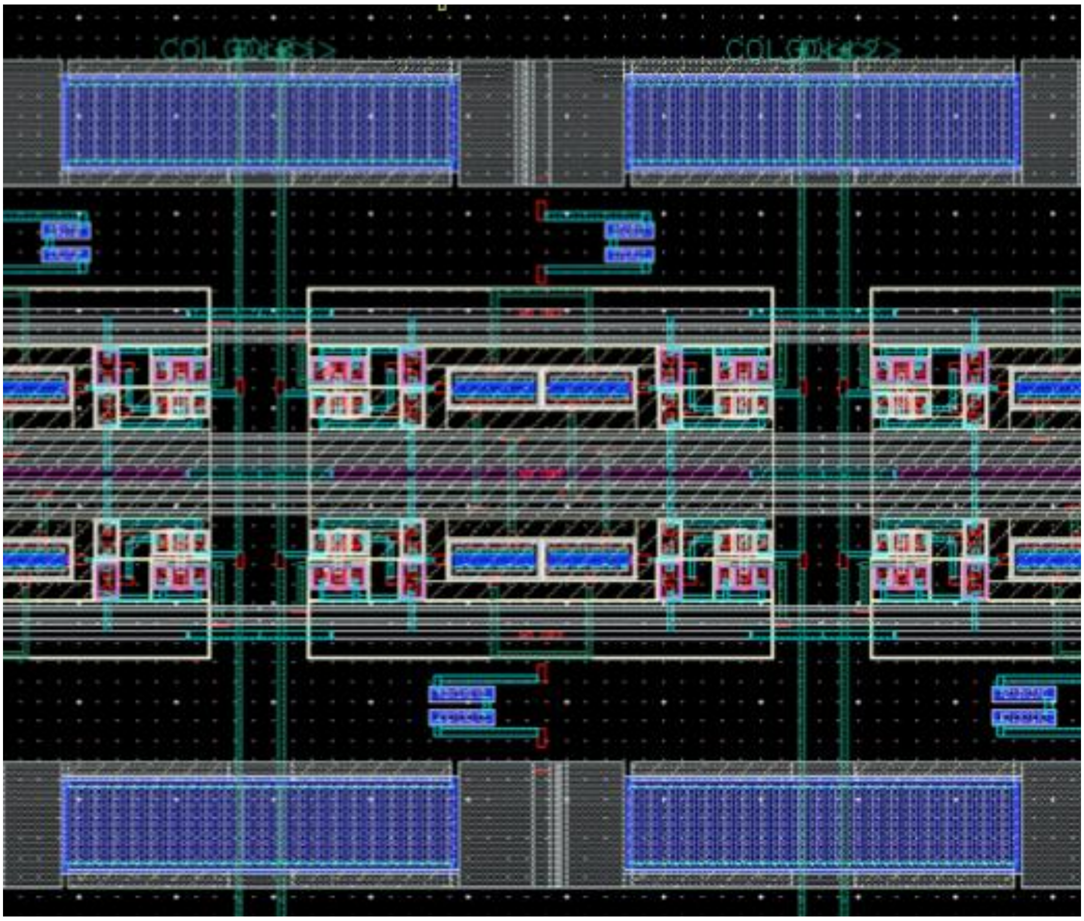


Figure 4.5 : Resistor string w/ compensation resistors and SW1 (zoom 3).

DRC and LVS clean finalized SW1 separate from the string is shown below in Figure 4.6. Four of these cells arranged symmetrically can be roughly seen in Figure 4.5 (center).

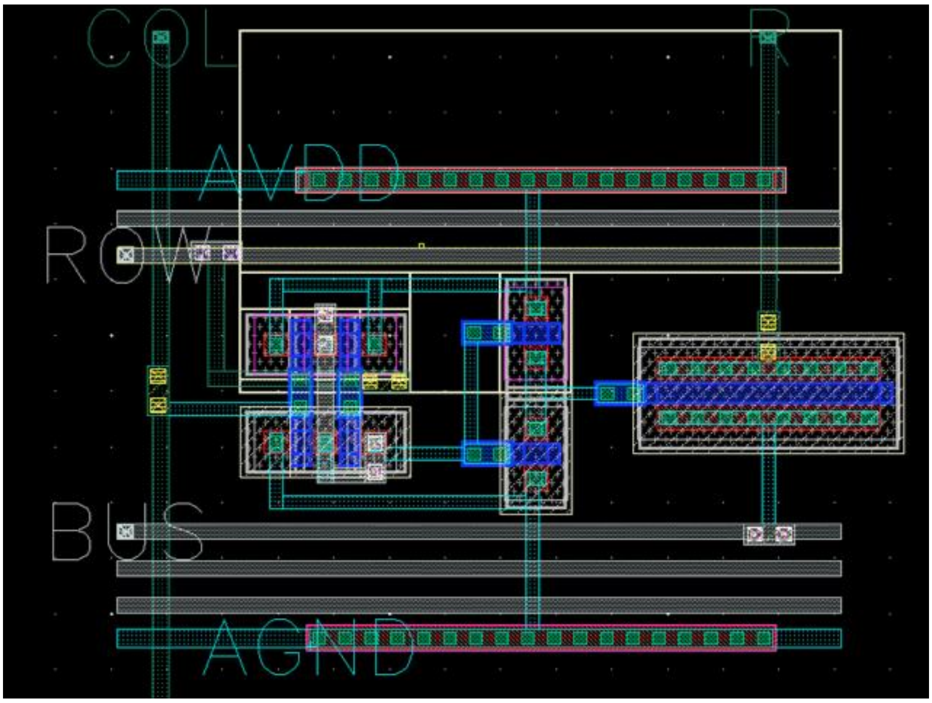


Figure 4.6 : SW1.

4.3 Switch Bank 1

DRC and LVS clean finalized the second switch bank is shown below in Figure 4.7. This structure connects to the 4 output busses (VHLA_OUT, VHLB_OUT, VHLA_D_OUT, VHLB_D_OUT), then repeated 32 times for the 32 main string loops shown below in Figure 4.8.

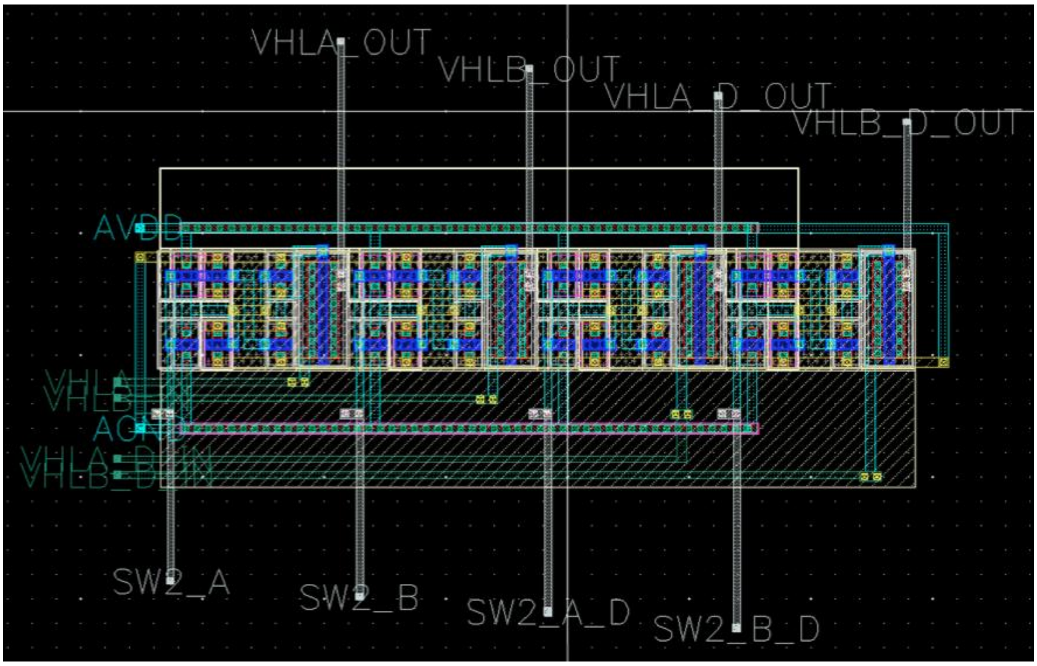


Figure 4.7 : SW2 (one cell).

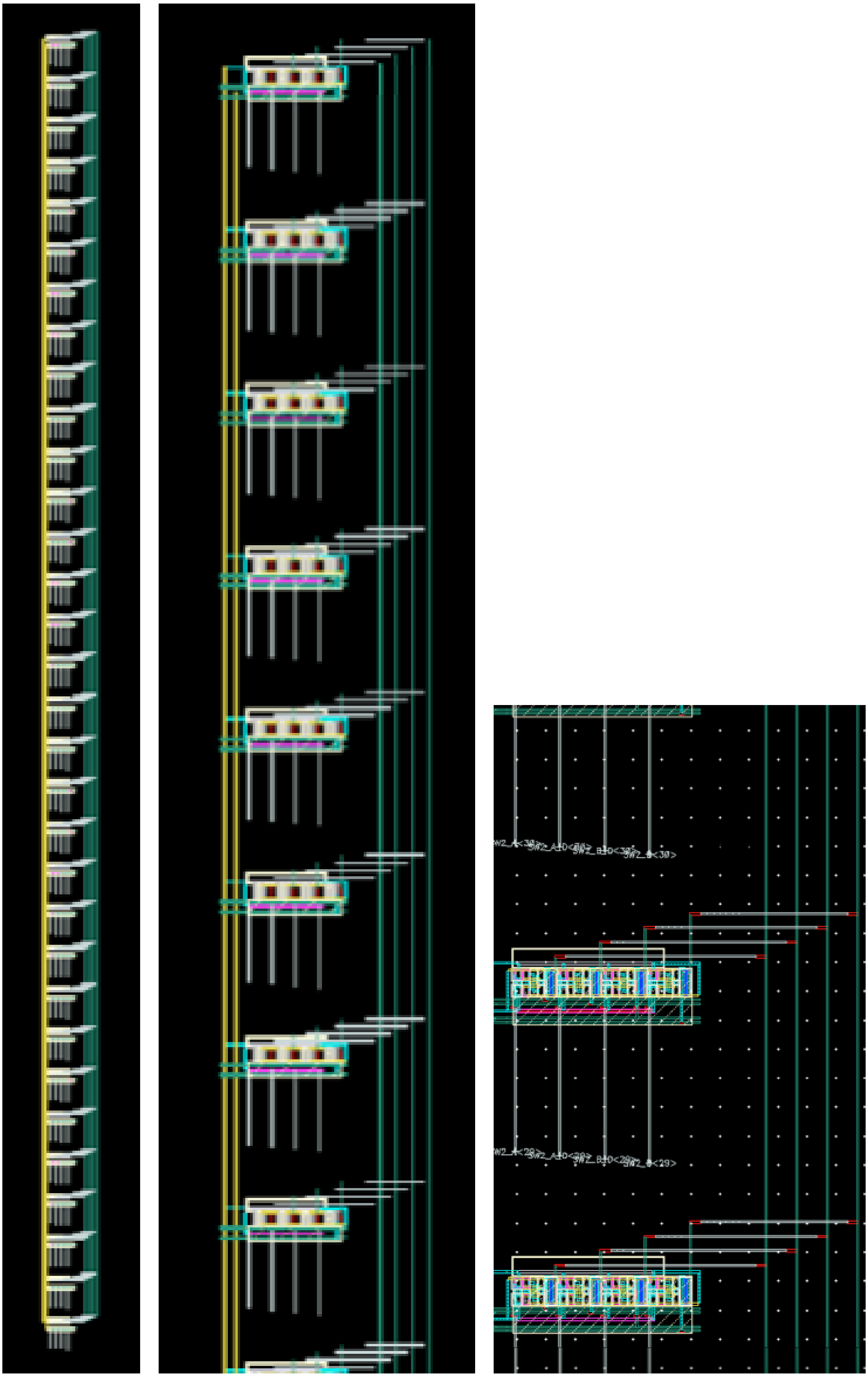


Figure 4.8 : SW2 a) whole block b) zoom 1 c) zoom 2.

4.4 Switch Bank 3

DRC and LVS clean finalized third switch bank is shown below in Figure 4.9. Just like SW2, the whole block is a thin strip.This structure is repeated 2 times for differential output, then repeated 64 times for the 6 LSBs.

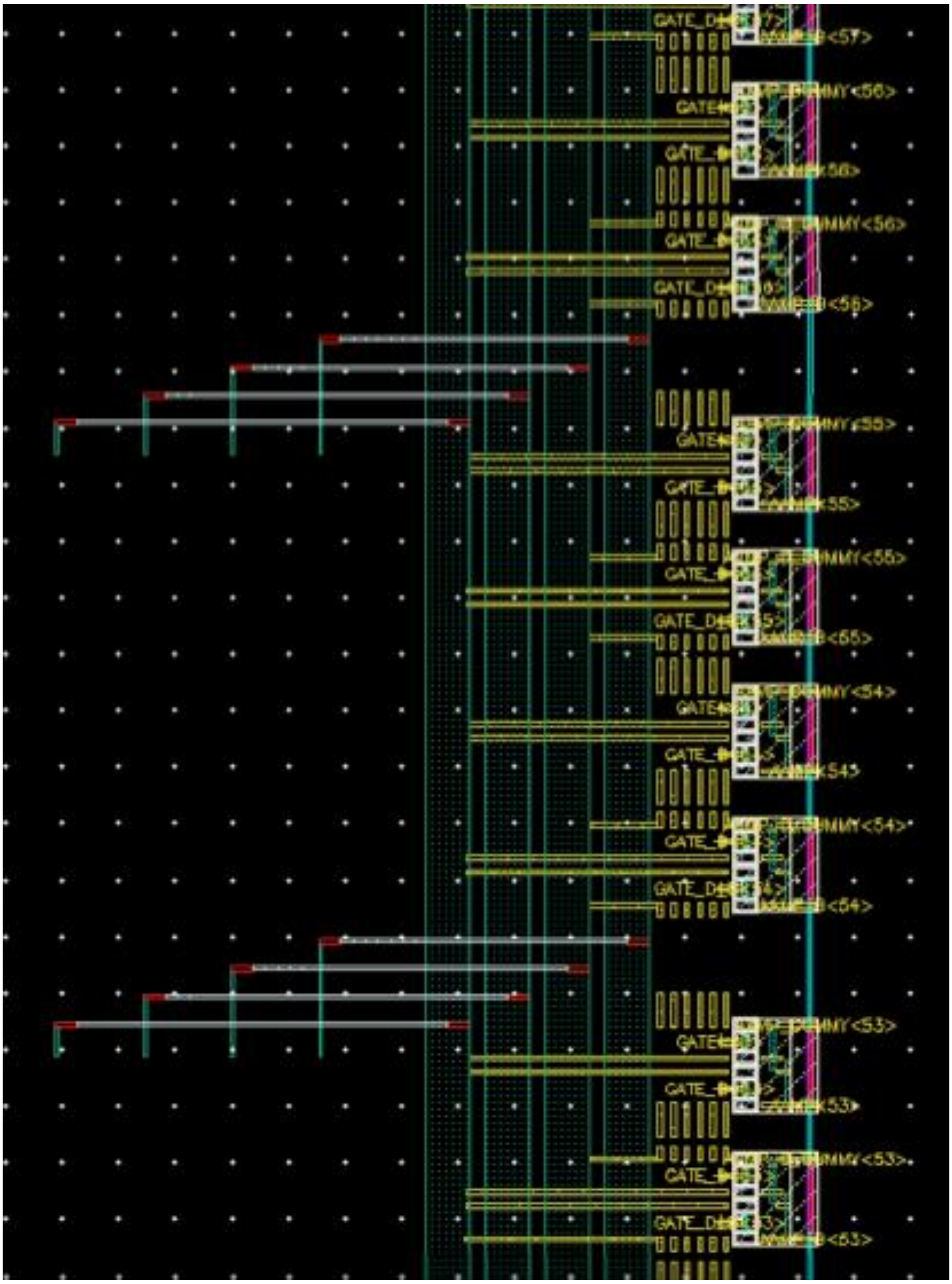


Figure 4.9 : SW3 whole block zoomed.

This block attaches to the 2nd switch bank as shown below in Figure 4.10.

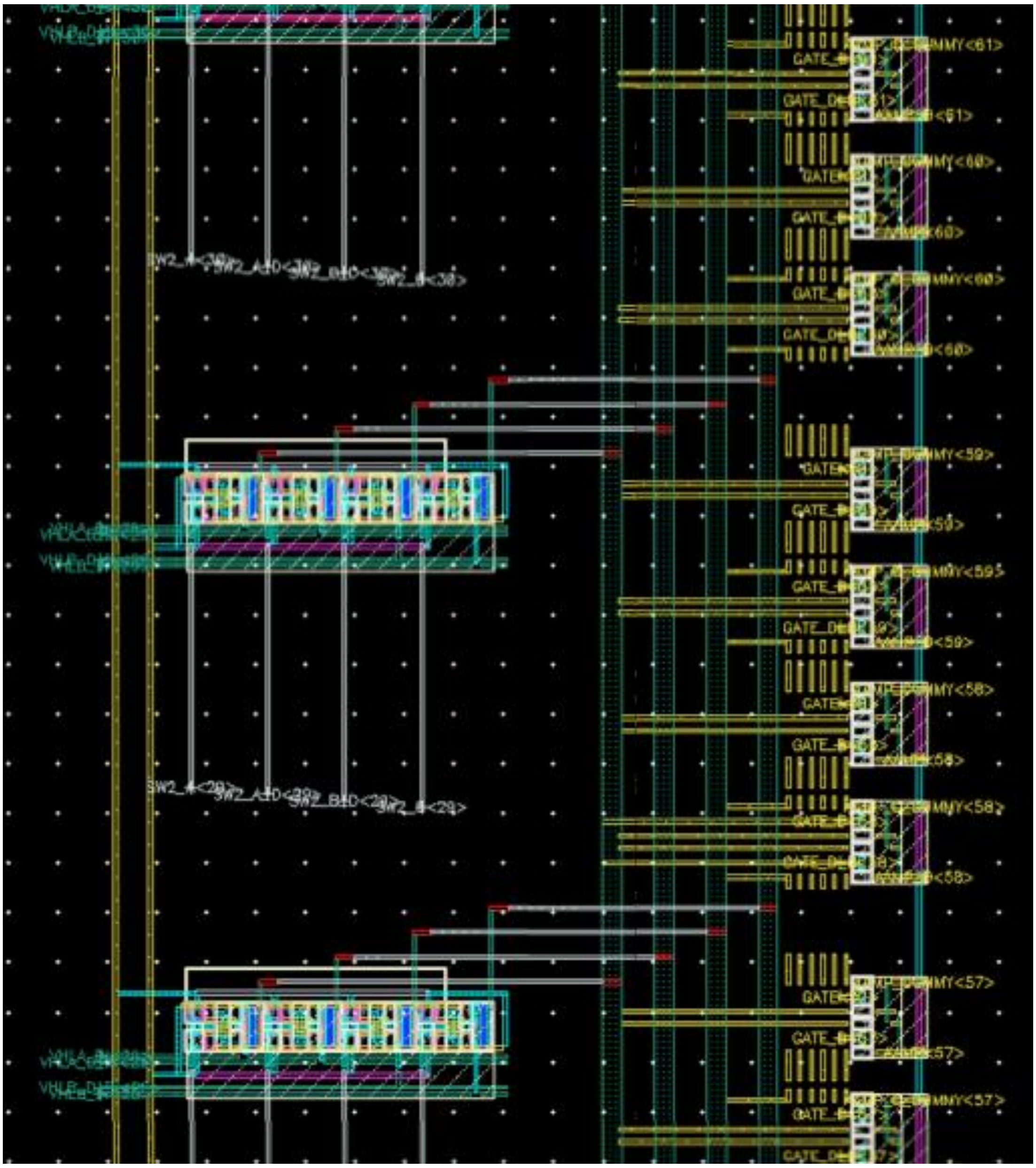


Figure 4.10 : SW2 and SW3 whole block zoomed.

4.5 Interpolating OPAMP

DRC and LVS clean finalized interpolating OPAMP is shown below in Figure 4.11.

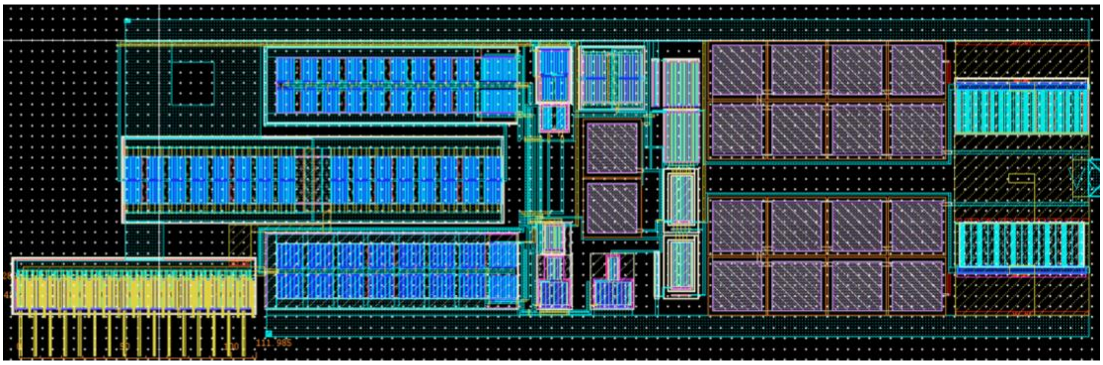


Figure 4.11 : Interpolating OPAMP.

Connecting the compact differential pair (bottom left corner of Figure 4.11) of the interpolating OPAMP to the long switch 3 bank caused performance issues due to unequal connection lengths. So the differential pair has been redesigned as part of switch bank 2 and 3 as shown in Figure 4.12.

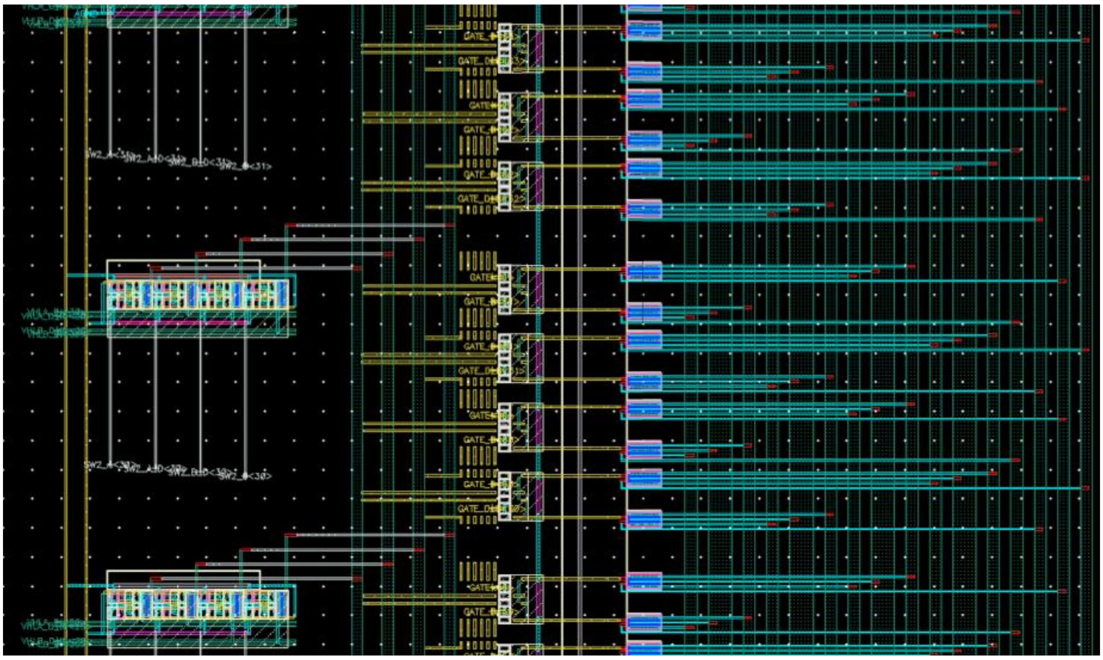


Figure 4.12 : Differential pair (rightmost), SW3 (middle) and SW2 (left most).

Interpolating OPAMPs are attached to this layout as shown in Figure 4.13, forming the second main layout piece (switch bank 2, switch bank3, differential pair, interpolating OPAMPs), first main layout piece being (string, compensation resistors, switch 1 bank).

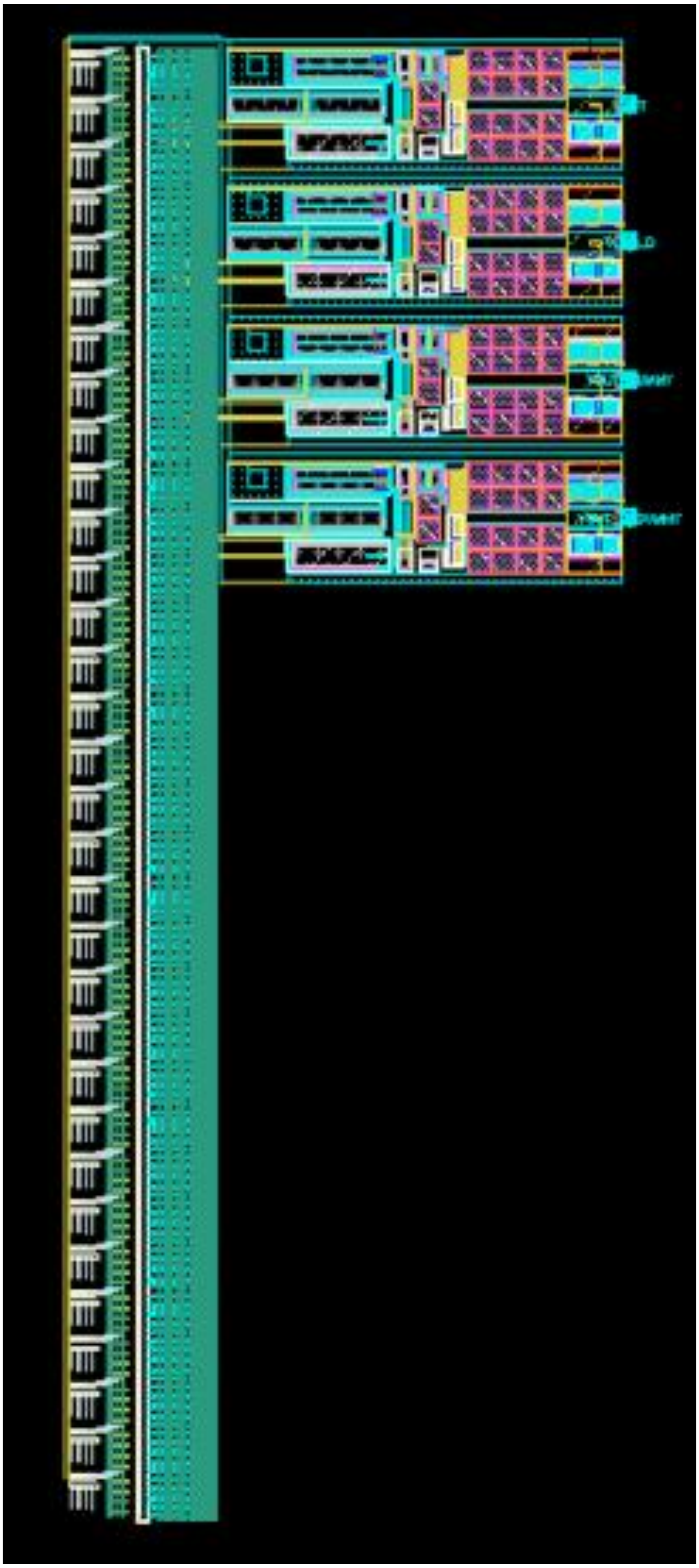


Figure 4.13 : Interpolating OPAMPs (rightmost), differential pair, SW3 and SW2 (left most).

4.6 Complete DAC Core Layout

DRC and LVS clean finalized differential output layout is given below in Figure 4.14.

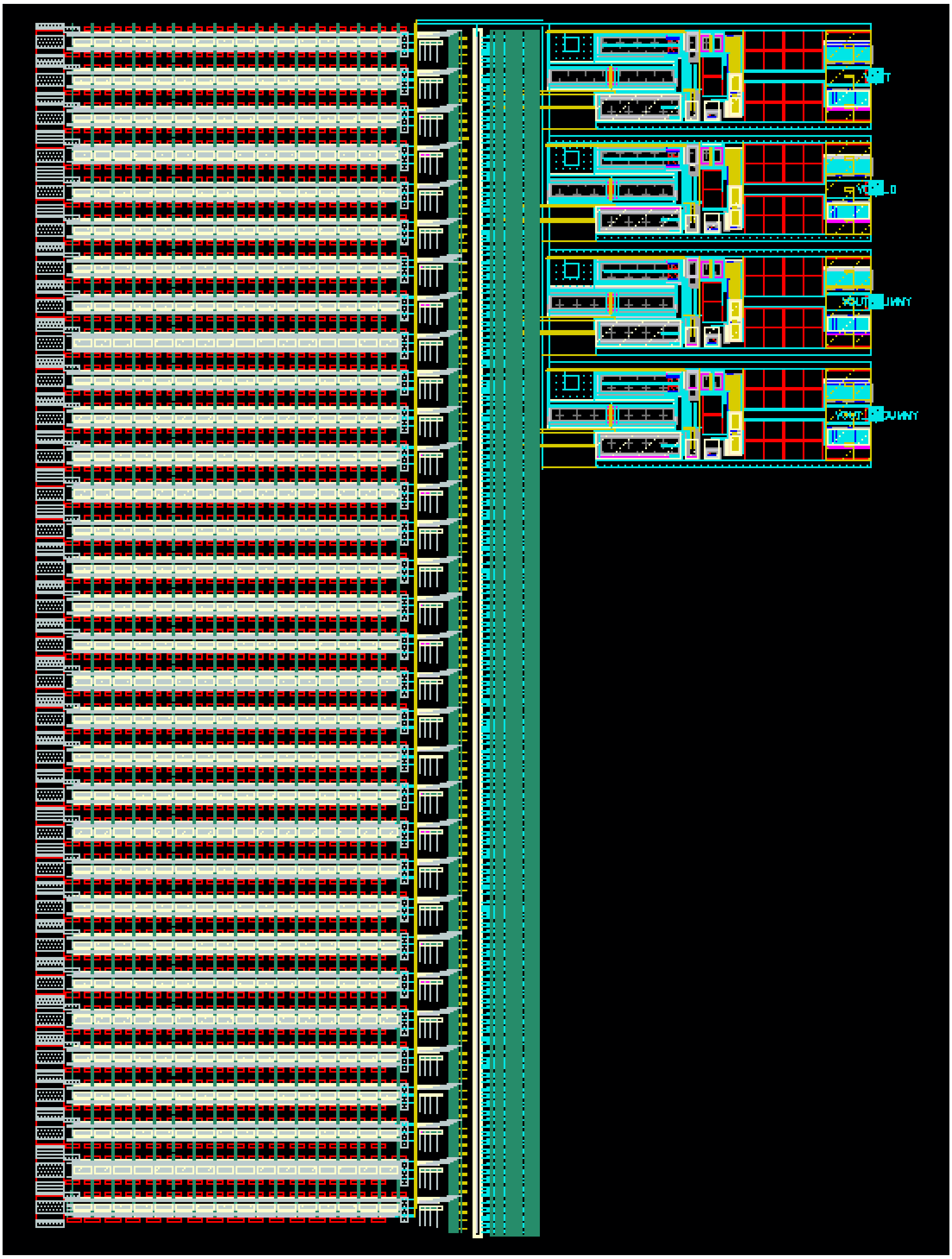


Figure 4.14 : Interpolating OPAMPs (rightmost), differential pair, switch bank 3 and switch bank 2, string and switch bank 1 (left most).

4.7 Instrumentation Amplifier Configuration

DRC and LVS clean finalized single-ended output layout is given below in Figure 4.15.

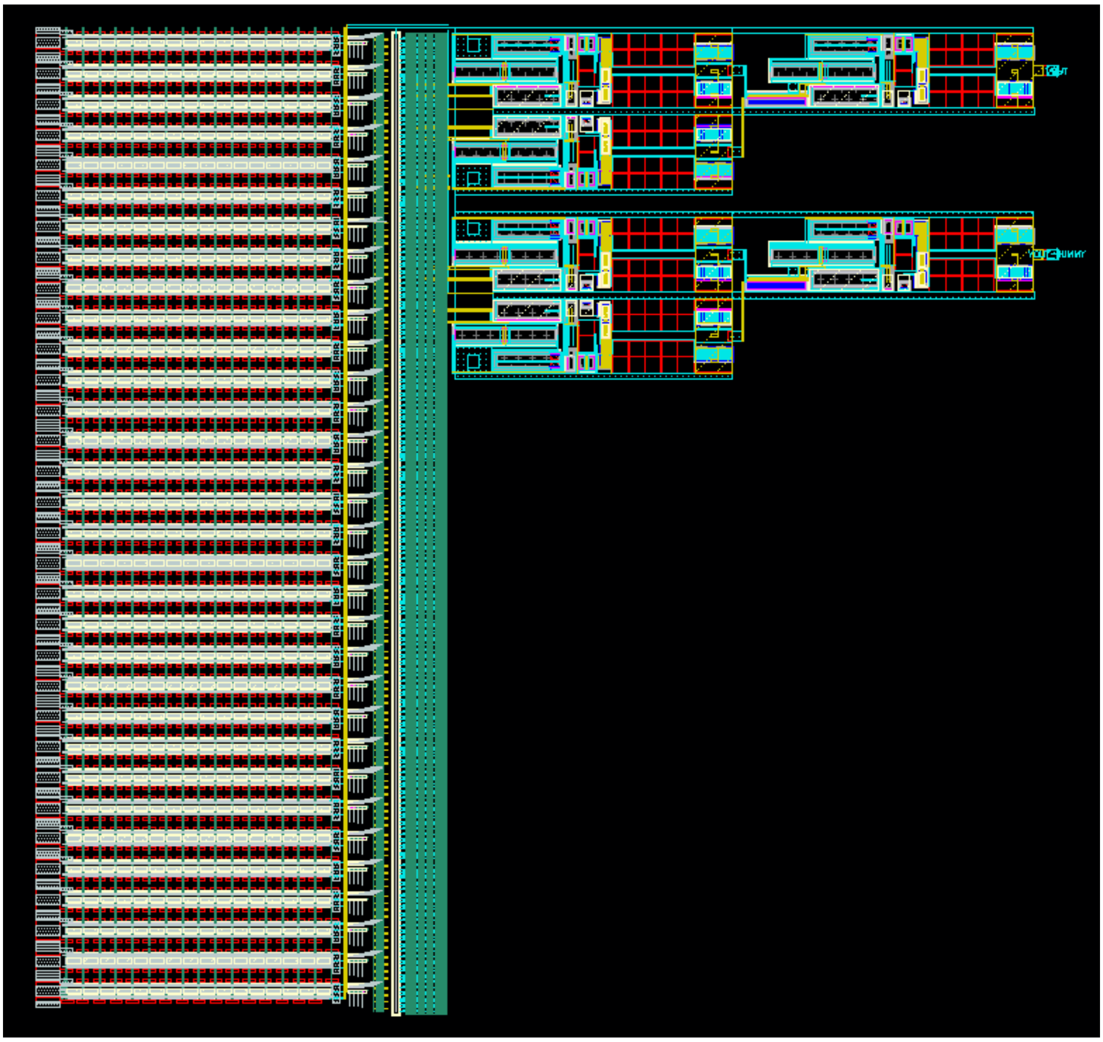


Figure 4.15 : Instrumentation amplifier configuration DAC core.

4.8 Bandgap Reference

DRC and LVS clean finalized bandgap reference core and its digitally programmable resistor string is shown below in Figures 4.16 and 4.17. Please note that this block is a part of our design lab’s standard cell library.

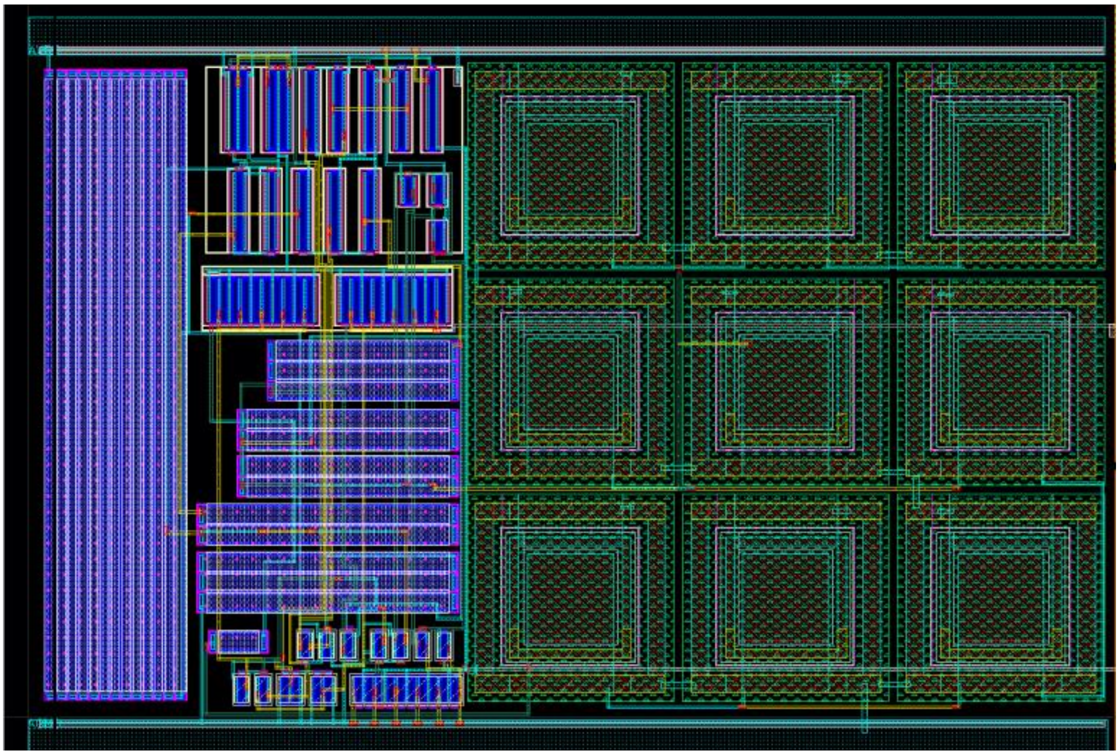


Figure 4.16 : Brokaw bandgap reference core.

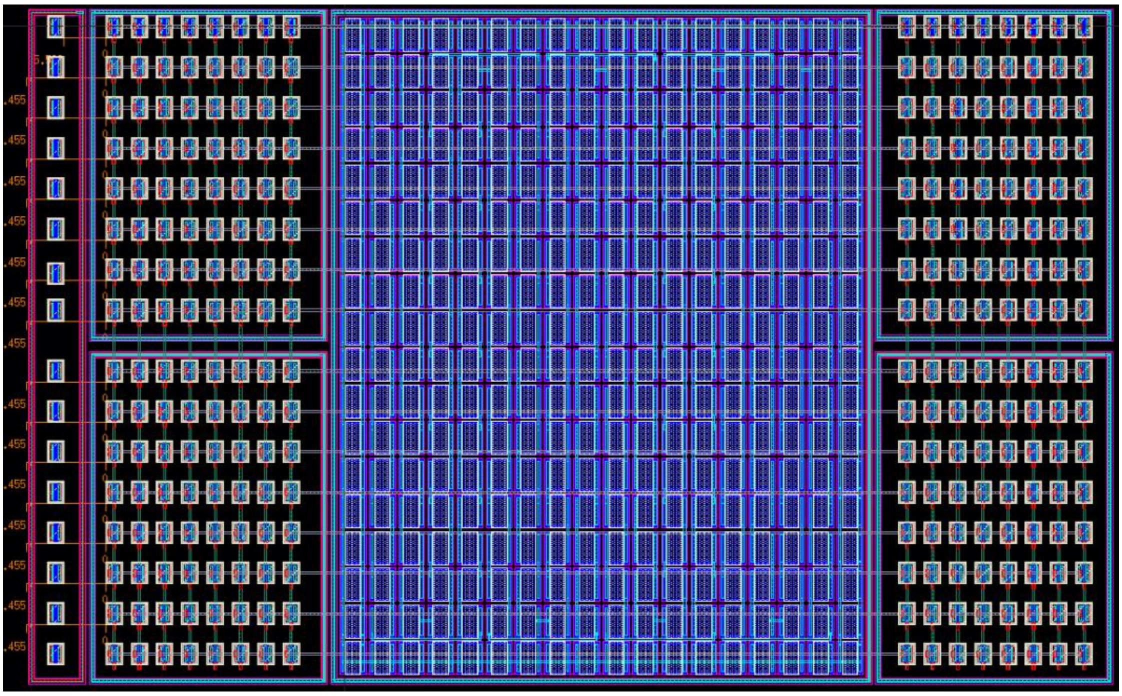


Figure 4.17 : Digitally programmable resistor string.

4.9 Decoder

Place-and-routed DAC decoder is shown below in Figure 4.18.

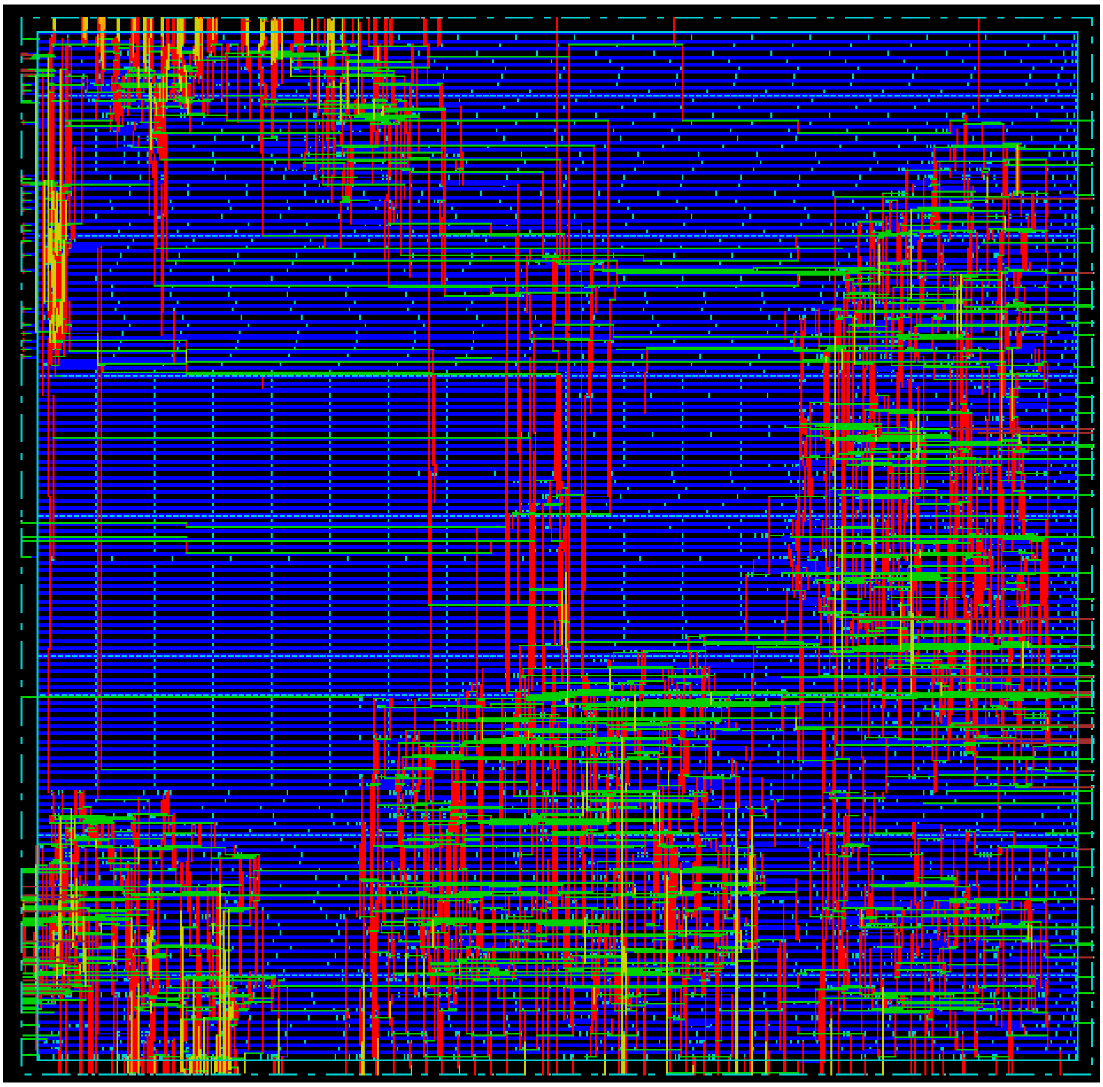


Figure 4.18 : Decoder.

4.10 Extracted Results

Table 4.1 compares the schematic and layout-extracted SFDR results of the differential output and single ended full DAC simulations. PVT corners analysis for SS, FF, TT process variation, -40-25-85 temperature variation and 3.2-3.3-3.4 supply voltage variation is given below in Table 4.2 for the single ended DAC.

Table 4.1 : Single ended vs. differential output (layout).

Schematic	SFDR (dB)	Layout Extracted	SFDR (dB)
Differential output	87.87	Differential output	84.39
Single-ended output	84.37	Single-ended output	83.82

Table 4.2 : Layout corners.

Corner	Supply (V)	Process	Temperature (°C)	SFDR (dB)
1 (N)	3.3	TT	27	86.31
2	3.3	FF	27	84.4
3	3.3	SS	27	82.34
4	3.3	TT	-40	88.17
5	3.3	FF	-40	86.11
6	3.3	SS	-40	84.24
7	3.3	TT	85	84.82
8	3.3	FF	85	85.01
9	*3.3*	*SS*	*85*	*77.13*
10	3.2	TT	27	86.27
11	3.2	FF	27	86.48
12	*3.2*	*SS*	*27*	*75.64*
13	3.2	TT	-40	88.11
14	3.2	FF	-40	88.34
15	*3.2*	*SS*	*-40*	*77.37*
16	3.2	TT	85	81.39
17	3.2	FF	85	85.06
18	*3.2*	*SS*	*85*	*72.38*
19	3.4	TT	27	86.25
20	3.4	FF	27	81.58
21	3.4	SS	27	86.14
22	3.4	TT	-40	88.12
23	3.4	FF	-40	83.43
24	3.4	SS	-40	88.00
25	3.4	TT	85	84.79
26	3.4	FF	85	82.43
27	3.4	SS	85	83.11

Most corners are satisfied for over 80 dB SFDR, except SS process corners at high temperature and low supply voltage (just below the 80 dB mark), which is acceptable. SFDR with input frequency for the single-ended output design and the differential output design is givenbelow in Figure 4.19.

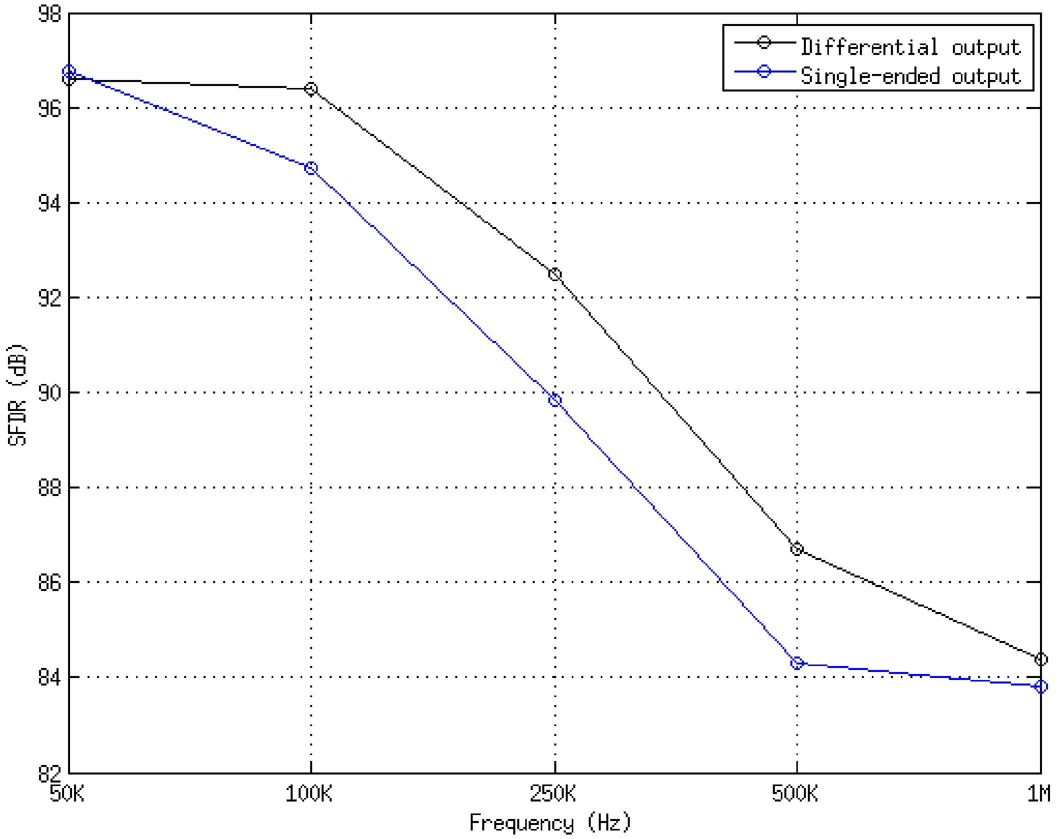


Figure 4.19 : Final layout SFDR vs. output frequency.

The design is is completed to success. Tape-out is expected to be April 2015. Post tape-out testing preparations have been made by measuring the TI DAC8580, which is a high performance 16-bit voltage output DAC with similar applications to the design proposed in this work.

5. CONCLUSION

SFDR performance of the final design in schematic and layout is given again in Figure 5.1 below along with the performance of the design referenced in [7] and the basic resitor string architecture given in Figure 1.10 under the same conditions.

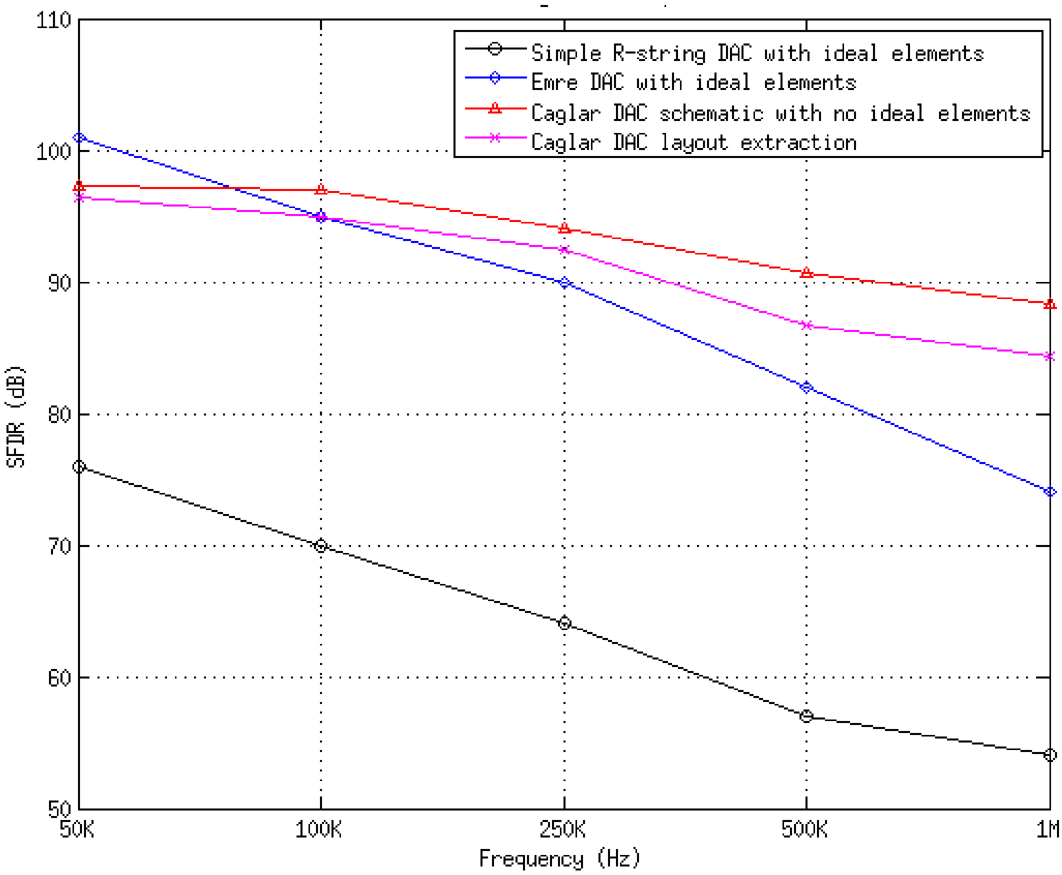


Figure 5.1 : SFDR vs. output frequency summary.

This work, realised what E. Topcu modelled in [7] in schematic and layout, prioritized the error mechanisms proposed in [7] along with new error mechanisms found during my research, investigated architectures that can reduce these effects and their trade-offs with other error mechanisms, and proposed a layout-complete DAC-core design that surpasses all previous theoretical work and current state-of-the-art by a significant amount in dynamic linearity performance.

Tape-out is expected to be April 2015 at TSMC. Post-tape-out testing preparations have been made by measuring the TI DAC8580 to specifications – a state-of-the-art high performance 16-bit voltage output DAC with similar applications to the design proposed in this work.

REFERENCES

[1] **Maloberti, F.** (2007). *Data Converters*, Springer, Dordreicht, Netherlands.

[2] **Andrew G. F. Dingwall, Victor Zazzu.** (1985). A 16-bit Resistor String DAC with Full Calibration at Final Test, *IEEE International Test Conference Proceeding,* 75-85.

[3] **Url-1** < *http://www.analog.com/static/imported-files/data_sheets/AD5516.pdf*>, date retrieved 15.12.2014.

[4] **Url-2** <*http://www.analog.com/static/imported-files/data_sheets/DAC8420.pdf*

>, date retrieved 15.12.2014.

[5] **Yilmaz, A.** (2001). LSB Interpolation Circuit and Method for Segmented digital to Analog Converter, *US Patent*, US6246351 dated 12.06.2001.

[6] **Kuyel, T.** (2007). High Speed, High Resolution Voltage Output Digital to Analog Converter and Method, *US Patent*, US7283082, 2007.

[7] **Topcu, E.** (2012). Yüksek Çözünürlüklü Direnç Dizesi Tipli Sayısaldan Analoga Dönüştürücülerde Dinamik Hata Mekanizmalarının İncelenmesi, *MSc Thesis*, ITU, Istanbul, Turkey.

[8] **Url-3** <*http://www.ti.com/lit/ds/symlink/dac8580.pdf*>, date retrieved

15.12.2014.

[9] **Kuyel, T.** (2006). All Digital Calibration of String DAC Linearity Using Area Efficient PWL Approximation: Eliminating Digital Division and Hardware Search, *US Patent*, US7002496 dated 21.02.2006.

[10] **Kuyel, T., Yilmaz, A.** (2006). All Analog Calibration of String DAC Linearity Application to High Voltage Process, *US Patent,* US6897794 dated 2004.

[11] **Zumbahlen, H.** (2008). Linear Circuit Design Handbook, *Newnes.*

APPENDICES

APPENDIX A: Decoder Verilog Code
APPENDIX B: Ideal ADC-DAC Testbench VerilogAMS Code
APPENDIX C: Calibration Resistor Generating Script

APPENDIX A: Decoder Verilog Code

```
//Verilog HDL for "CO_THESIS2", "decoder_16b2_1" "verilog"
//
//This code is used to synthesize a decoder for switching Resistor string outputs,
//                                                    Second switch bank
//                                                    Interpolator amplifier's inputs
//Code includes:
//              Precharge output
//              Clock input (operates at posedge clk)

`timescale 10ps/1ps

module decoder_16b2_2 (DIN, DIN6, CLK, ROW, ROW_D, COL, COL_D,
SW2_A, SW2_AD, SW2_B, SW2_BD, ENDBIT, ENDBIT_D, SWPC_A,
SWPC_B, SWPC_AD, SWPC_BD, DOUT_64, DOUT_64D, DOUTDUMMY_64,
DOUTDUMMY_64D);

//digital 16 bit input
input  [9:0]    DIN;     //10 bit part
input  [5:0]     DIN6;           //6 bit part

//Clock input
input CLK;

//******DECODER OUTPUTS****************************

//First switch bank
output [63:0]   ROW;            //row output
output [63:0]   ROW_D;                  //row output's inverse
output [15:0]   COL;            //column output
output [15:0]   COL_D;                  //column output's inverse
output          ENDBIT;         //one bit output for 1024th code
output          ENDBIT_D;       //inverse of ENDBIT

reg [63:0]   ROWreg;            //row output
reg [63:0]   ROW_Dreg;                  //row output's inverse
reg [15:0]   COLreg;            //column output
reg [15:0]   COL_Dreg;                  //column output's inverse
reg         ENDBITreg;          //one bit output for 1024th code
reg         ENDBIT_Dreg;        //inverse of ENDBIT


//Second switch bank
output [31:0]   SW2_A;                  //output for 2nd switch bank, A
output [31:0]   SW2_AD;                 //inverse of SW2_A
output [31:0]   SW2_B;                  //output for 2nd switch bank, B
output [31:0]   SW2_BD;                 //inverse of SW2_B
output [31:0]   SWPC_A;                 //precharge switch for output A
output [31:0]   SWPC_AD;   //inverse of SWPC_A
output [31:0]   SWPC_B;                 //precharge switch for output B
output [31:0]   SWPC_BD;    //inverse of SWPC_B

reg [31:0]   SW2_Areg ;                 //output for 2nd switch bank, A
reg [31:0]   SW2_ADreg ;                //inverse of SW2_A
reg [31:0]   SW2_Breg ;                 //output for 2nd switch bank, B
reg  [31:0]   SW2_BDreg ;               //inverse of SW2_B
reg  [31:0]   SWPC_Areg ;               //precharge switch for output A
reg  [31:0]   SWPC_ADreg ;  //inverse of SWPC_A
reg [31:0]   SWPC_Breg ;                //precharge switch for output B
reg  [31:0]   SWPC_BDreg ;  //inverse of SWPC_B

//Third switch bank
output [63:0]   DOUT_64;     //6 to 64 bit decoder output
output [63:0]   DOUT_64D;  //inverse of DOUT_64
output [63:0]   DOUTDUMMY_64; //dummy output
output [63:0]   DOUTDUMMY_64D;          //inverse of dummy output

reg  [63:0]   DOUT_64reg ;   //6 to 64 bit decoder output
reg  [63:0]   DOUT_64Dreg ;//inverse of DOUT_64
reg  [63:0]   DOUTDUMMY_64reg ;          //dummy output
reg  [63:0]   DOUTDUMMY_64Dreg ;         //inverse of dummy output

//wires used for assignments and connections
wire [9:0]    din_msb;
wire [5:0]    din_6b;
wire [15:0]   din_all;
wire [63:0]   ROW;
wire [63:0]   ROW_D;
wire [15:0]   COL;
wire [15:0]   COL_D;
wire          ENDBIT;
wire          ENDBIT_D;
wire [31:0]   SW2_A;
wire [31:0]   SW2_AD;
wire [31:0]   SW2_B;
wire [31:0]   SW2_BD;
wire [31:0]   SWPC_A;
wire [31:0]   SWPC_AD;
wire [31:0]   SWPC_B;
wire [31:0]   SWPC_BD;
wire [63:0]   DOUT_64;
wire [63:0]   DOUT_64D;
wire [63:0]   DOUTDUMMY_64;
wire [63:0]   DOUTDUMMY_64D;


// registers to store values for decoding
reg     whatsup;
reg [4:0]    a;
reg [5:0]    b;
reg [6:0]    c;
reg [63:0]   row_bin;
reg [63:0]   row_bin2;
reg [63:0]   row_temp;
reg [63:0]   row_i;
reg [63:0]   row_i2;
reg [63:0]   row_shiftone;
reg [63:0]   row_shiftone2;
reg [63:0]   row_final;
reg [63:0]   row_final2;
reg [63:0]   row_end;
reg [15:0]   col_bin;
reg [15:0]   col_bin2;
reg [15:0]   col_temp;
reg [15:0]   col_i;
reg [15:0]   col_i2;
reg [15:0]   col_shiftone;
reg [15:0]   col_shiftone2;
reg [15:0]   col_final;
reg [15:0]   col_final2;
reg [15:0]   col_end;
reg [9:0]    din_msb2;
reg          end_bit;
reg [31:0]   sw2_bin_A;
reg [31:0]   sw2_temp;
reg [31:0]   sw2_i_A;
reg [31:0]   sw2_final_A;
reg [31:0]   sw2_final_two_A;
reg [31:0]   sw2_bin_B;
reg [31:0]   sw2_i_B;
reg [31:0]   sw2_final_B;
reg [31:0]   sw2_final_two_B;
reg [63:0]   dout_64;
reg [63:0]   dout_64d;
reg [63:0]   dout_temp;
reg [63:0]   dout_final;
reg [63:0]   dout_final_z;

reg [9:0]    d;
reg [9:0]    din_msb_d;
reg [9:0]    din_msb_d2;
reg [63:0]   row_bin_d;
reg [63:0]   row_bin_d2;
reg [63:0]   row_i_d;
reg [63:0]   row_i_d2;
reg [63:0]   row_shiftone_d;
reg [63:0]   row_shiftone_d2;
reg [63:0]   row_final_d;
reg [63:0]   row_final_d2;
reg [63:0]   row_end_d;
reg [15:0]   col_bin_d;
reg [15:0]   col_bin_d2;
reg [15:0]   col_i_d;
reg [15:0]   col_i_d2;
reg [15:0]   col_shiftone_d;
reg [15:0]   col_shiftone_d2;
reg [15:0]   col_final_d;
reg [15:0]   col_final_d2;
reg [15:0]   col_end_d;
reg          end_bit_d;
reg [31:0]   sw2_bin_A_d;
reg [31:0]   sw2_i_A_d;
reg [31:0]   sw2_final_A_d;
reg [31:0]   sw2_final_two_A_d;
reg [31:0]   sw2_bin_B_d;
reg [31:0]   sw2_i_B_d;
reg [31:0]   sw2_final_B_d;
reg [31:0]   sw2_final_two_B_d;

assign din_msb = DIN[9:0];
assign din_6b = DIN6[5:0];
assign din_all = {DIN, DIN6};

always @ (din_msb, din_6b, din_all)
begin

//initial assignments for division
a = 5'b10000;           //16
b = 6'b100000;                  //32
c = 7'b1000000;                 //64

d = 10'b1111111111;
//initial assignment for decoding
row_temp = 64'hFFFFFFFFFFFFFFFF;
col_temp = 16'hFFFF;
sw2_temp = 32'hFFFFFFFF;
dout_temp = 64'hFFFFFFFFFFFFFFFF;

din_msb_d = d-din_msb;

//starting decoding code
  if(din_msb == 10'b0000000000) begin  //when input is zero

// normal
    end_bit = 1'b1;                 //make endbit 1
    end_bit_d = 1'b0;
    row_bin = din_msb / a;            //row = input/16
       if(row_bin%2==0) begin       //if (row is even)
       col_bin = din_msb % a;       //column = input%16
```

```
          end else begin             //else
          col_bin = 4'hF - (din_msb%a);//column = 15 - (input%16)
          end

      sw2_bin_A = din_msb / b;                  //swich A number = input/32

//this operation creates decoder output for rows and columns. Decoding operation is
the same for other outputs


      row_i = (row_temp << (row_bin));                          //shift row_temp by row
number Example: 11111111111111->11111111000000
      row_shiftone = row_i << 1;                      //shift  row_i by 1
            11111111000000->11111110000000
      row_final = (row_i ^ row_shiftone);             //xOR these two
                  00000001000000

      col_i = (col_temp << (col_bin));                //shift column by column number
      col_shiftone = col_i << 1;                      //shift col_i by 1
      col_final = (col_i ^ col_shiftone);             //xOR these two

      row_end = row_final;
      col_end = col_final;

//decoding of switch bank A
      sw2_i_A = (sw2_temp << (sw2_bin_A));
      sw2_final_A = sw2_i_A << 1;
      sw2_final_two_A = (sw2_i_A ^ sw2_final_A);

//assign switch A to switch B
      sw2_i_B = sw2_i_A;
      sw2_final_B = sw2_final_A;
      sw2_final_two_B = sw2_final_two_A;


//normal end

//Decoding operation for differential output
//in differential operation we decode the previous code from the input and OR these
two decoded outputs
//Example: 00000001000000|00000000100000 -> 00000001100000

//diff
      din_msb_d2 = din_msb_d-1'b1;                    //for defining previous code

      row_bin_d = din_msb_d / a;
      if(row_bin_d%2==0) begin
          col_bin_d = din_msb_d % a;
          end else begin
          col_bin_d = 4'hF - (din_msb_d%a);
```

```
    end

    row_bin_d2 = din_msb_d2 / a;
    if(row_bin_d2%2==0) begin
        col_bin_d2 = din_msb_d2 % a;
        end else begin
        col_bin_d2 = 4'hF - (din_msb_d2%a);
    end

//decode row
    row_i_d = (row_temp << (row_bin_d));
    row_shiftone_d = row_i_d << 1;
    row_final_d = (row_i_d ^ row_shiftone_d);
//decode column
    col_i_d = (col_temp << (col_bin_d));
    col_shiftone_d = col_i_d << 1;
    col_final_d = (col_i_d ^ col_shiftone_d);

//decode to open previous switch row and coulumn
    row_i_d2 = (row_temp << (row_bin_d2));
    row_shiftone_d2 = row_i_d2 << 1;
    row_final_d2 = (row_i_d2 ^ row_shiftone_d2);

    col_i_d2 = (col_temp << (col_bin_d2));
    col_shiftone_d2 = col_i_d2 << 1;
    col_final_d2 = (col_i_d2 ^ col_shiftone_d2);

    row_end_d = row_final_d|row_final_d2;          // OR operation for differential
row output
    col_end_d = col_final_d|col_final_d2;   // OR operation for differential column
output

//decode process for other switch banks
    sw2_bin_A_d = din_msb_d / b;
    sw2_bin_B_d = din_msb_d2 / b;
    sw2_i_A_d = (sw2_temp << (sw2_bin_A_d));
    sw2_final_A_d = sw2_i_A_d << 1;
    sw2_final_two_A_d = (sw2_i_A_d ^ sw2_final_A_d);
    sw2_i_B_d = (sw2_temp << (sw2_bin_B_d));
    sw2_final_B_d = sw2_i_B_d << 1;
    sw2_final_two_B_d = (sw2_i_B_d ^ sw2_final_B_d);

// diff end

  end else if (din_msb == 10'b1111111111) begin // FOR CODE 1023

//normal

    end_bit = 1'b0;
    end_bit_d = 1'b1;
    din_msb2 = din_msb-1'b1;

    row_bin = din_msb / a;
    if(row_bin%2==0) begin
        col_bin = din_msb % a;
        end else begin
        col_bin = 4'hF - (din_msb%a);
    end

    row_bin2 = din_msb2 / a;
    if(row_bin2%2==0) begin
        col_bin2 = din_msb2 % a;
        end else begin
        col_bin2 = 4'hF - (din_msb2%a);
    end

    sw2_bin_A = din_msb / b;
    sw2_bin_B = din_msb2 / b;

    row_i = (row_temp << (row_bin));
    row_shiftone = row_i << 1;
    row_final = (row_i ^ row_shiftone);

    col_i = (col_temp << (col_bin));
    col_shiftone = col_i << 1;
    col_final = (col_i ^ col_shiftone);

    row_i2 = (row_temp << (row_bin2));
    row_shiftone2 = row_i2 << 1;
    row_final2 = (row_i2 ^ row_shiftone2);

    col_i2 = (col_temp << (col_bin2));
    col_shiftone2 = col_i2 << 1;
    col_final2 = (col_i2 ^ col_shiftone2);

    row_end = row_final|row_final2;
    col_end = col_final|col_final2;

    sw2_i_A = (sw2_temp << (sw2_bin_A));
    sw2_final_A = sw2_i_A << 1;
    sw2_final_two_A = (sw2_i_A ^ sw2_final_A);
    sw2_i_B = (sw2_temp << (sw2_bin_B));
    sw2_final_B = sw2_i_B << 1;
    sw2_final_two_B = (sw2_i_B ^ sw2_final_B);

// normal end

// diff

    row_bin_d = din_msb_d / a;
```

```
        if(row_bin_d%2==0) begin
        col_bin_d = din_msb_d % a;
        end else begin
        col_bin_d = 4'hF - (din_msb_d%a);
        end



    row_i_d = (row_temp << (row_bin_d));
    row_shiftone_d = row_i_d << 1;
    row_final_d = (row_i_d ^ row_shiftone_d);

    col_i_d = (col_temp << (col_bin_d));
    col_shiftone_d = col_i_d << 1;
    col_final_d = (col_i_d ^ col_shiftone_d);

    row_end_d = row_final_d;
    col_end_d = col_final_d;

    sw2_bin_A_d = din_msb_d / b;
    sw2_i_A_d = (sw2_temp << (sw2_bin_A_d));
    sw2_final_A_d = sw2_i_A_d << 1;
    sw2_final_two_A_d = (sw2_i_A_d ^ sw2_final_A_d);
    sw2_i_B_d = sw2_i_A_d;
    sw2_final_B_d = sw2_final_A_d;
    sw2_final_two_B_d = sw2_final_two_A_d;


// diff end

  end else begin // FOR REST

//normal

    end_bit = 1'b0;
    end_bit_d = 1'b0;
    din_msb2 = din_msb-1'b1;

    row_bin = din_msb / a;
    if(row_bin%2==0) begin
        col_bin = din_msb % a;
        end else begin
        col_bin = 4'hF - (din_msb%a);
    end

    row_bin2 = din_msb2 / a;
    if(row_bin2%2==0) begin
        col_bin2 = din_msb2 % a;
        end else begin
        col_bin2 = 4'hF - (din_msb2%a);
    end

    sw2_bin_A = din_msb2 / b;
    sw2_bin_B = din_msb / b;

    row_i = (row_temp << (row_bin));
    row_shiftone = row_i << 1;
    row_final = (row_i ^ row_shiftone);

    col_i = (col_temp << (col_bin));
    col_shiftone = col_i << 1;
    col_final = (col_i ^ col_shiftone);

    row_i2 = (row_temp << (row_bin2));
    row_shiftone2 = row_i2 << 1;
    row_final2 = (row_i2 ^ row_shiftone2);

    col_i2 = (col_temp << (col_bin2));
    col_shiftone2 = col_i2 << 1;
    col_final2 = (col_i2 ^ col_shiftone2);

    row_end = row_final|row_final2;
    col_end = col_final|col_final2;

    sw2_i_A = (sw2_temp << (sw2_bin_A));
    sw2_final_A = sw2_i_A << 1;
    sw2_final_two_A = (sw2_i_A ^ sw2_final_A);
    sw2_i_B = (sw2_temp << (sw2_bin_B));
    sw2_final_B = sw2_i_B << 1;
    sw2_final_two_B = (sw2_i_B ^ sw2_final_B);

// normal end

// diff
    din_msb_d2 = din_msb_d-1'b1;

    row_bin_d = din_msb_d / a;
    if(row_bin_d%2==0) begin
        col_bin_d = din_msb_d % a;
        end else begin
        col_bin_d = 4'hF - (din_msb_d%a);
    end

    row_bin_d2 = din_msb_d2 / a;
    if(row_bin_d2%2==0) begin
        col_bin_d2 = din_msb_d2 % a;
        end else begin
        col_bin_d2 = 4'hF - (din_msb_d2%a);
    end
```

```
      row_i_d = (row_temp << (row_bin_d));
      row_shiftone_d = row_i_d << 1;
      row_final_d = (row_i_d ^ row_shiftone_d);

      col_i_d = (col_temp << (col_bin_d));
      col_shiftone_d = col_i_d << 1;
      col_final_d = (col_i_d ^ col_shiftone_d);

      row_i_d2 = (row_temp << (row_bin_d2));
      row_shiftone_d2 = row_i_d2 << 1;
      row_final_d2 = (row_i_d2 ^ row_shiftone_d2);

      col_i_d2 = (col_temp << (col_bin_d2));
      col_shiftone_d2 = col_i_d2 << 1;
      col_final_d2 = (col_i_d2 ^ col_shiftone_d2);

      row_end_d = row_final_d|row_final_d2;
      col_end_d = col_final_d|col_final_d2;

      sw2_bin_A_d = din_msb_d2 / b;
      sw2_bin_B_d = din_msb_d / b;

      sw2_i_A_d = (sw2_temp << (sw2_bin_A_d));

      sw2_final_A_d = sw2_i_A_d << 1;
      sw2_final_two_A_d = (sw2_i_A_d ^ sw2_final_A_d);
      sw2_i_B_d = (sw2_temp << (sw2_bin_B_d));
      sw2_final_B_d = sw2_i_B_d << 1;
      sw2_final_two_B_d = (sw2_i_B_d ^ sw2_final_B_d);

// diff end

   end

//thermometer decoder for third switches
   dout_64 = dout_temp << (din_6b);
   dout_64d = ~(dout_64);


   if(row_bin2%2==0) begin
   dout_final = dout_64d;
   end else begin
   dout_final = ~(dout_64);
   end

   whatsup = (din_all/c)%2;

   if(whatsup==0) begin
   dout_final_z = ~(dout_final);
   end else begin
```

```
  dout_final_z = dout_final;
  end

end


always@ (posedge CLK)
begin

ROWreg <= row_end;
ROW_Dreg <= row_end_d;
COLreg <= col_end;
COL_Dreg <= col_end_d;
ENDBITreg <= end_bit;
ENDBIT_Dreg <= end_bit_d;

SW2_Areg <= sw2_final_two_A;
SW2_ADreg <= sw2_final_two_A_d;
SW2_Breg <= sw2_final_two_B;
SW2_BDreg <= sw2_final_two_B_d;

SWPC_Areg <= ~(sw2_final_two_A);
SWPC_ADreg <= ~(sw2_final_two_A_d);
SWPC_Breg <= ~(sw2_final_two_B);
SWPC_BDreg <= ~(sw2_final_two_B_d);

DOUT_64reg <= dout_final_z;
DOUT_64Dreg <= ~(dout_final_z);
DOUTDUMMY_64reg <= dout_final_z;
DOUTDUMMY_64Dreg <= ~(dout_final_z);

end


//assigns register values to outputs
assign ROW = ROWreg;
assign ROW_D = ROW_Dreg;
assign COL = COLreg;
assign COL_D = COL_Dreg;
assign ENDBIT = ENDBITreg;
assign ENDBIT_D = ENDBIT_Dreg;

assign SW2_A = SW2_Areg;
assign SW2_AD = SW2_ADreg;
assign SW2_B = SW2_Breg;
assign SW2_BD = SW2_BDreg;

assign SWPC_A = SWPC_Areg;
assign SWPC_AD = SWPC_ADreg;
assign SWPC_B = SWPC_Breg ;
assign SWPC_BD = SWPC_BDreg;

assign DOUT_64 = DOUT_64reg;
assign DOUT_64D = DOUT_64Dreg;
assign DOUTDUMMY_64 = DOUTDUMMY_64reg;
assign DOUTDUMMY_64D = DOUTDUMMY_64Dreg;

endmodule
```

APPENDIX B: Ideal ADC-DAC Testbench VerilogAMS Code

```
`include "discipline.h"
`include "constants.h"

module adc_16b (vin, vclk, dout);
input vin;
input vclk;
output [15:0] dout;

electrical vin, vclk;
electrical [15:0] dout;

parameter real trise = 20p;
parameter real tfall = 20p;
parameter real tdel = 0;
parameter real vlogic_high = 3.0;
parameter real vlogic_low  = 0;
parameter real vtrans_clk     = 1.5;
parameter real vref        = 2.0;

`define NUM_ADC_BITS    16
  real unconverted;
  real halfref;

  real vd[0:`NUM_ADC_BITS-1];
  integer i;

  analog begin

    @ ( initial_step ) begin
       halfref = vref / 2;
    end

    @ (cross(V(vclk) - vtrans_clk, 1)) begin
      unconverted = V(vin);
      for (i = (`NUM_ADC_BITS-1); i >= 0 ; i = i - 1) begin
        vd[i] = 0;
        if (unconverted > halfref) begin
          vd[i] = vlogic_high;
          unconverted = unconverted - halfref;
     end else begin
          vd[i] = vlogic_low;
     end
        unconverted = unconverted * 2;
      end
    end

    V(dout[0])  <+ transition( vd[0],  tdel, trise, tfall );
```

```
      V(dout[1])  <+ transition( vd[1],  tdel, trise, tfall );
      V(dout[2])  <+ transition( vd[2],  tdel, trise, tfall );
      V(dout[3])  <+ transition( vd[3],  tdel, trise, tfall );
      V(dout[4])  <+ transition( vd[4],  tdel, trise, tfall );
      V(dout[5])  <+ transition( vd[5],  tdel, trise, tfall );
      V(dout[6])  <+ transition( vd[6],  tdel, trise, tfall );
      V(dout[7])  <+ transition( vd[7],  tdel, trise, tfall );
      V(dout[8])  <+ transition( vd[8],  tdel, trise, tfall );
      V(dout[9])  <+ transition( vd[9],  tdel, trise, tfall );
      V(dout[10]) <+ transition( vd[10], tdel, trise, tfall );
      V(dout[11]) <+ transition( vd[11], tdel, trise, tfall );
      V(dout[12]) <+ transition( vd[12], tdel, trise, tfall );
      V(dout[13]) <+ transition( vd[13], tdel, trise, tfall );
      V(dout[14]) <+ transition( vd[14], tdel, trise, tfall );
      V(dout[15]) <+ transition( vd[15], tdel, trise, tfall );

`undef NUM_ADC_BITS
   end
endmodule

module dac_16b (vd, vout);
input [15:0] vd;
output vout;
electrical [15:0] vd;
electrical vout;
parameter real vref  = 2;
parameter real trise = 20p;
parameter real tfall = 20p;
parameter real tdel  = 0;
parameter real vtrans  = 1.5;

    real out_scaled; // output scaled as fraction of 256
    real out_scaled_last; // output scaled as fraction of 256

    analog begin
        out_scaled = 0;
        out_scaled_last = 0;
        out_scaled = out_scaled + ((V(vd[15]) > vtrans) ? 32768 : 0);
        out_scaled = out_scaled + ((V(vd[14]) > vtrans) ? 16384 : 0);
        out_scaled = out_scaled + ((V(vd[13]) > vtrans) ? 8192 : 0);
        out_scaled = out_scaled + ((V(vd[12]) > vtrans) ? 4096 : 0);
        out_scaled = out_scaled + ((V(vd[11]) > vtrans) ? 2048 : 0);
        out_scaled = out_scaled + ((V(vd[10]) > vtrans) ? 1024 : 0);
        out_scaled = out_scaled + ((V(vd[9]) > vtrans) ? 512 : 0);
        out_scaled = out_scaled + ((V(vd[8]) > vtrans) ? 256 : 0);
        out_scaled = out_scaled + ((V(vd[7]) > vtrans) ? 128 : 0);
        out_scaled = out_scaled + ((V(vd[6]) > vtrans) ? 64 : 0);
        out_scaled = out_scaled + ((V(vd[5]) > vtrans) ? 32 : 0);
        out_scaled = out_scaled + ((V(vd[4]) > vtrans) ? 16 : 0);
        out_scaled = out_scaled + ((V(vd[3]) > vtrans) ? 8 : 0);
        out_scaled = out_scaled + ((V(vd[2]) > vtrans) ? 4 : 0);
        out_scaled = out_scaled + ((V(vd[1]) > vtrans) ? 2 : 0);
        out_scaled = out_scaled + ((V(vd[0]) > vtrans) ? 1 : 0);
                out_scaled_last = vref*out_scaled/65535;
        V(vout) <+ transition( out_scaled_last, tdel, trise, tfall );
        // V(vout) <+ vref*out_scaled/65535;
    end
endmodule
```

APPENDIX C: Calibration Resistor Generating Script

```
clear
clc

% etting some variables up
x32 = zeros(1,32);
x1025 = zeros(1,1025);

for i=1:1:33
   x33(i) = (i-1)*32;
end

for i=1:1:1025
   x1025(i) = i;
end
R = 1.56021;
TOTALCODE=1024;
TOTALCOARSE=64;
TOTALFINE=16;

% calculate switch calibration
ron = [580.85 561.11 543.25 527.01 512.14 498.55 486.05 474.51 463.87 453.96
444.74 436.14 428.07 420.54 413.46 406.81 400.56 394.65 389.05 383.76 378.72
373.95 369.41 365.09 360.98 357.05 353.29 349.69 346.24 342.94 339.76 336.72
333.79];
ron = fliplr(ron);
p = polyfit(x33, ron, 3);
R_EQ = polyval(p, x1025);
R_STRING_COMPENSATION = (max(R_EQ)+5)-R_EQ;

% calculate string calibration
% Initialize arrays
R_TOP = zeros(1,TOTALCODE);
R_FINEBOT = zeros(1,TOTALCODE);
R_COARSE = zeros(1,TOTALCODE);
R_FINETOP = zeros(1,TOTALCODE);
R_BOT = zeros(1,TOTALCODE);
R_A = zeros(1,TOTALCODE);
R_B = zeros(1,TOTALCODE);
R_C = zeros(1,TOTALCODE);
R_D = zeros(1,TOTALCODE);
R_E = zeros(1,TOTALCODE);
R_1 = zeros(1,TOTALCODE);
R_2 = zeros(1,TOTALCODE);
R_3 = zeros(1,TOTALCODE);
R_4 = zeros(1,TOTALCODE);
R_5 = zeros(1,TOTALCODE);
R_EQ = zeros(1,TOTALCODE);
for code=1:1:TOTALCODE
```

```
% Initial model
R_TOP(code) = (TOTALCOARSE - ceil((code)/TOTALFINE)) *
((R*(TOTALFINE*R))/(R+(TOTALFINE*R)));
R_FINETOP(code) = ((mod((TOTALFINE-mod((code), TOTALFINE)),
TOTALFINE)))*R;
R_COARSE(code) = R;
R_FINEBOT(code) = ((TOTALFINE-mod((TOTALFINE-mod((code),
TOTALFINE)), TOTALFINE)))*R;
R_BOT(code) = floor((code-1)/TOTALFINE) *
((R*(TOTALFINE*R))/(R+(TOTALFINE*R)));

% Change naming for clarity
R_D(code) = R_TOP(code);
R_A(code) = R_FINEBOT(code);
R_C(code) = R_COARSE(code);
R_B(code) = R_FINETOP(code);
R_E(code) = R_BOT(code);

% Delta-star transformation
R_1(code) = (R_B(code)*R_C(code))/(R_A(code)+R_B(code)+R_C(code));
R_2(code) = (R_A(code)*R_C(code))/(R_A(code)+R_B(code)+R_C(code));
R_3(code) = (R_A(code)*R_B(code))/(R_A(code)+R_B(code)+R_C(code));
R_4(code) = R_D(code);
R_5(code) = R_E(code);

% Equivalent string resistance
R_EQS(code) = R_3(code) +
((R_4(code)+R_1(code))*(R_5(code)+R_2(code)))/(R_1(code)+R_2(code)+R_4(cod
e)+R_5(code));
end
R_STRING_COMPENSATION_S2 = (max(R_EQS)+5)-R_EQS;

R_STRING_COMPENSATION_S = [R_STRING_COMPENSATION_S2(1024)
R_STRING_COMPENSATION_S2];
ALLCOMP_2 = R_STRING_COMPENSATION + R_STRING_COMPENSATION
+ R_STRING_COMPENSATION_S;

ALLCOMP = ALLCOMP_2 - (min(ALLCOMP_2)-50);

%resistor values
for i=1:1:6000
    x600(i) = 0.1*i;
end

R_entered = [11.578 20.014 29.93 40.028 49.953 60.043 69.967 80.057 89.982
100.072 119.921 139.936 159.95 179.643 199.979 224.956 249.933 275.075 300.052
333.299 366.711 399.958 450.077 500.031 599.938];
R_measured = [11.89888114  20.56805221 30.76722265 41.13615569 51.33526126
61.70427199 71.90325744 82.27231138 92.47142493 102.8405815 123.2723198
143.8065149 164.3745548 184.9423316 205.5114433 231.1782384 256.8453169
282.6843016 308.3506975 342.5190721 376.8585981 411.0254462 462.5735913
513.8666127 616.5343298];
L_generated = [700 1210 1810 2420 3020 3630 4230 4840 5440 6050 7250 8460
9670 10860 12090 13600 15110 16630 18140 20150 22170 24175 27210 30230
36270];

p_E = polyfit(R_measured, R_entered, 3);
p_L = polyfit(R_entered, L_generated, 3);

R_E_func = polyval(p_E, x600);
R_L_func = polyval(p_L, x600);

% total resistance an length generation
res_index = round(ALLCOMP*5);
W = zeros(1,1025);

for i=1:1:1025
W(i) = round((R_L_func(round(R_E_func(res_index(i))*10))/10000)*1000)/100;
end

fid = fopen('compensation_resistors_real3', 'w');
fprintf(fid, '.SUBCKT compensation_resistors_real3 ');
for i = 0:1:(TOTALCOARSE-1)
for j = 0:1:(TOTALFINE-1)
fprintf(fid, 'res_in<%d> ', i*TOTALFINE+j);
end
fprintf(fid, '\n');
end
fprintf(fid, 'res_in<%d> ', TOTALCODE);
for i=0:1:(TOTALCOARSE-1)
for j=0:1:(TOTALFINE-1)
fprintf(fid, 'res_out<%d> ', i*TOTALFINE+j);
end
fprintf(fid, '\n');
end
fprintf(fid, 'res_out<%d> \n\n', TOTALCODE);
fprintf(fid, ' R0 (res_in<0> res_out<0>) rnpo1w l=%.2fu w=420n mf=(1) m=1
mismatchflag=1\n', max(W));
for i=1:1:1024
fprintf(fid, ' R%d (res_in<%d> res_out<%d>) rnpo1w l=%.2fu w=420n mf=(1) m=1
mismatchflag=1\n',i,i,i,W(i));
end
fprintf(fid, '\n');
fprintf(fid, '.END');
fclose(fid)
```

CURRICULUM VITAE

Name Surname: Çağlar Özdağ

Place and Date of Birth: 22nd December 1988, Ankara

Address: Şair Latifi Sok. 27/1, Moda, Kadıköy, Istanbul, TURKEY

E-Mail: caglarozdag@gmail.com

B.Sc.: University of British Columbia, Vancouver, CANADA

List of Publications and Patents:

Çağlar Özdağ, Türker Küyel, Güneş Karabulut Kurt, Alphan Salarvan "Digital-to-analog Converter Effects on Orthogonal Frequency Division Multiplexing Performance", IEEE SIU 2013, pp 1-28