Austriamicrosystems AG,
Valencia Design Center

Escuela Técnica Superior de Ingenieros de Telecomunicación,
Universidad Politécnica de Valencia

Diseño de un Convertidor Analógico-Digital de Aproximaciones Sucesivas de bajo consumo y área reducida

Proyecto Fin de Carrera

Bueno Gimeno, Enrique

Tutores:

Herrero Bosch, Vicente (U. Politécnica de Valencia)

Poirier, Sébastien (Austriamicrosystems)

Valencia, 10 de Febrero de 2010

Resumen

El Proyecto Fin de Carrera aquí presentado se enmarca en el ámbito del diseño microelectrónico, concretamente en el área de los sistemas integrados mixtos.
El objeto del mismo es el estudio y diseño de un convertidor analógico-digital de aproximaciones sucesivas de bajo consumo y área reducida usando la tecnología CMOS de 0.35µm de *Austriamicrosystems*. Así como el aprendizaje y entrenamiento en el uso de las herramientas de diseño microelectrónico de *Cadence* y el kit de diseño de *Austriamicrosystems*.

El convertidor a implementar se encuentra dentro del grupo de los denominados convertidores analógico-digital de aproximaciones sucesivas (SAR ADC), los cuales se basan en el uso de un algoritmo de búsqueda por aproximaciones sucesivas (SAR) aplicado sobre un convertidor digital-analógico (DAC) y un comparador, para encontrar, tras varias iteraciones, el código digital de N bits que mejor representa la señal analógica de entrada.
Para este proyecto se ha optado por utilizar una topología SAR *fully-differential* de 12 bits, a la cual se le han impuesto especificaciones de: bajo consumo (~30µA) y área reducida (~0.4mm^2), todo ello con un objetivo de *throughput* de 10kSPS. Teniendo en cuenta estas especificaciones, muy restrictivas en cuanto a área se refiere, se ha optado por implementar una solución basada en un DAC capacitivo de dimensiones extremadamente reducidas, acompañado de un sistema de auto-calibración, para compensar los problemas de linealidad derivados del pobre *matching* entre las capacidades tan pequeñas del DAC. Para satisfacer las necesidades de consumo, la gran parte del esfuerzo se ha centrado en el diseño del comparador, ajustando su velocidad para conseguir los 10kSPS y un consumo verdaderamente reducido, además de incorporar un circuito de cancelación de *offset* para obtener la precisión que se requiere al trabajar en un ADC de 12 bits. En cuanto a la implementación del algoritmo de búsqueda SAR y del sistema de auto-calibración, se ha diseñado un bloque digital con la máquina de estados que genera la secuencia necesaria para la ejecución del algoritmo SAR junto al proceso de calibración del DAC.

La característica más relevante de este proyecto es la incorporación de un sistema de calibración del DAC, permitiendo de esta forma el uso de DAC's con requerimientos de *matching* menores, lo que posibilita la obtención de resoluciones mayores con un área más reducida. El uso del sistema de calibración no es gratuito, pues la complejidad del diseño aumenta considerablemente y se requiere de registros de memoria para almacenar información de calibración, sin embargo, el precio a pagar por el uso de un sistema de calibración es rídiculo en comparación con las ventajas, en forma de reducción del área del DAC, que éste aporta.

Índice

BLOQUE III: RESULTADOS Y CONCLUSIONES

Glosario

ADC : (*Analog Digital Converter*) Convertidor analógico-digital.

Convertidor A/D : Convertidor analógico-digital.

Convertidor D/A : Convertidor digital-analógico.

DAC : (*Digital Analog Converter*) Convertidor digital-analógico.

DNL : (*Differential Non Linearity*) Error de linealidad diferencial.

Fdt : Función de transferencia.

FSM : (*Finite State Machine*) Máquina de estados finitos.

Fully-differential : Tratamiento con señales diferenciales.

Fringe Effect : Efecto de bordes.

INL : (*Integral Non linearity*) Error de linealidad integral.

LSB : (*Least Significant Bit*) Bit menos significativo.

Matching : Nivel de exactitud entre dos dispositivos idénticos.

Missing codes : Códigos perdidos.

MPW : (*Multi-Project-Wafer*). Oblea destinada a la fabricación de diferentes proyectos (normalmente para testeo).

SAR : (*Successive Approximation Register*) Algoritmo de búsqueda por aproximaciones sucesivas.

SAR ADC : Convertidor analógico-digital de aproximaciones sucesivas.

Single- ended : Tratamiento con señales referenciadas a masa.

Toplevel : Nivel superior de la jerarquía de un diseño.

THD : (*Total Harmonic Distortion*). Distorsión armónica total.

Throughput : Tasa de transferencia.

Índice de figuras

Índice de tablas

BLOQUE I

INTRODUCCIÓN Y ESTADO DEL ARTE

CAPÍTULO 1

Introducción

1.1 ANTECEDENTES Y MOTIVACION

El entorno que nos rodea, por su propia naturaleza, es de carácter analógico, las señales presentes en él, y que los seres humanos somos capaces de captar, son analógicas, es decir, pueden tomar cualquier valor dentro de un rango continuo. Por esa razón, es obvio pensar, que para procesar cualquier información de nuestro entorno será necesario trabajar en el dominio continuo. Sin embargo, surge la posibilidad de poder discretizar las señales, es decir, permitir que puedan tomar valores sólo de un conjunto discreto de valores posibles, y no de todo el rango continuo, esto abre un mundo de nuevas posibilidades dentro del campo del tratamiento de señales, al poder tratarlas y procesarlas ahora en el dominio digital.

El procesamiento de señales en el dominio digital, ofrece grandes ventajas frente al procesamiento analógico: gran inmunidad frente al ruido y a la distorsión, mayor flexibilidad y modularidad, menores requerimientos de consumo y computación... Además de reducir drásticamente la complejidad de los sistemas, lo que hace posible la aplicación de complicadas técnicas de procesamiento, muy difíciles o inviables de implementar en el dominio analógico. Desde el punto de vista de la industria, el procesamiento digital también supone grandes beneficios, pues la menor complejidad de los sistemas se traduce en: tiempos de desarrollo menores, reducción de costes y una menor dependencia de las habilidades del diseñador.

Todo esto no sería posible sin los interfaces que realizan la función de pasarela entre el mundo analógico y digital, y viceversa. Es aquí donde se hace relevante la función de los convertidores analógico-digital (ADC) y digital-analógico (DAC).

En la siguiente figura, se muestra un esquema típico de un sistema de tratamiento de señales en el dominio digital.

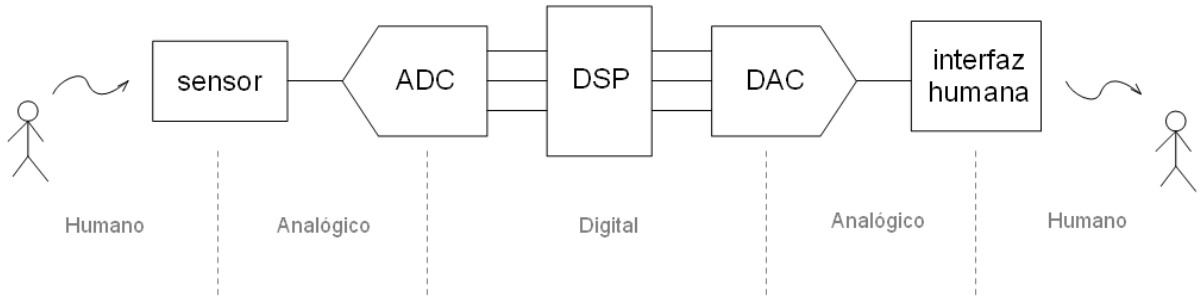


Figura 1.1: Diagrama de bloques de un sistema típico de procesamiento digita de señal

En la actualidad, son muchas las aplicaciones que basan su funcionamiento en el esquema mostrado anteriormente, ya sea en el mercado de la automoción, de las comunicaciones, de la electrónica de consumo o para aplicaciones médicas o de la industria aeronáutica y militar. En todas ellas, se hace imprescindible el uso de los convertidores analógico-digital y digital-analógico, lo que lleva a que el mercado de los convertidores sea uno de los más fructíferos del sector del silicio.

Como ejemplo, en la tabla 1.1 se observa el número de convertidores A/D y D/A vendidos en el año 2005 y su distribución por sectores de negocio.

Al hablar de convertidores analógico-digital, se deben tener en cuenta varios parámetros básicos que los describen, como son: la arquitectura que implementan, la resolución, la velocidad, el consumo y el área que ocupan. Cada arquitectura tiene su propia identidad y características que la definen, lo que hace que en función de los requerimientos necesarios de la aplicación, sea mejor el uso de un convertidor de un tipo u otro.

A modo de resumen, se presenta en la tabla 1.2 las características típicas de cada arquitectura, así como los aspectos más y menos interesantes de cada una de ellas.

Volumen del mercado de los convertidores A/D y D/A (2005)	
Sector	**Porcentaje de ventas (millones $)**
Automoción	(1.7% , 1.8%)
Telecomunicaciones fijas	(13.14% , 10.7%)
Telecomunicaciones móviles	(7.1% , 5.4%)
Dispositivos	(0.7% , 1.5%)
Infraestructuras	(6.4% , 3.9%)
Informática	(9.0% , 10.0%)
Ordenadores	(1.0% , 2.1%)
Periféricos y equipamiento de oficina	(8.0% , 7.9%)
Electrónica de consumo	(21.1% , 34.2%)
Línea blanca	(1.2% , 1.5%)
Línea marrón	(20.0% , 34.0%)
Industria y Medicina	(43.4% , 34.9%)
Militar y Aeronáutica	(4.3% , 2.8%)
TOTAL	**1285.0 , 833.5**

Tabla 1.1: Distribución de las ventas de convertidores A/D y D/A por sectores (2005)

Fundamentalmente, existen cuatro tipos de arquitecturas para un convertidor A/D. Por un lado, los convertidores denominados Sigma-Delta, empleados en aplicaciones con altos requerimientos en resolución y no grandes velocidades, como ocurre en el caso del tratamiento digital de audio, y por otro, los convertidores Pipeline y Flash, que se caracterizan por sus grandes velocidades pero con una resolución no muy elevada. Todos ellos requieren de áreas grandes y consumos considerables, lo que hace que sean convertidores utilizados sólo en aplicaciones muy específicas, en donde la resolución o la velocidad son aspectos críticos.

En medio de estos dos grupos se encuentran los denominados convertidores de tipo SAR, los cuales se caracterizan por tener velocidades y resoluciones medias; pero que al mismo tiempo, requiren de un área reducida para su implementación y un consumo muy bajo. Estas últimas características son las causantes de que esta arquitectura se haya convertido en una de las más utilizadas en la gran mayoría de las aplicaciones.

En este proyecto se impone el objetivo de implementar un ADC de tipo SAR, por su gran versatilidad en multitud de aplicaciones, pero intentando maximizar sus puntos fuertes, es decir, conseguir un consumo muy reducido y un área extremadamente pequeña.

	SAR	Pipeline	Flash	Sigma Delta
Velocidad	< 5 MSPS	< 300 MSPS	< 1.5 GSPS	< 16 kSPS
Resolución	< 13 bits	< 11 bits	< 10 bits	< 24 bits
Consumo	Muy bajo	Alto	Alto	Alto
Área	Pequeño	Medio	Grande	Grande
Desventajas	No destaca ni en velocidad ni en resolución	Consumo elevado	Baja resolución, alto consumo	Muy lento y alto consumo
Ventajas	Bajo consumo, área reducida y gran versatilidad	Rápido	Muy rápido	Muy alta resolución

Tabla 1.2: Arquitecturas de ADC´s: características, ventajas e inconvenientes

1.2 OBJETIVOS

El objeto de este proyecto es el estudio, diseño y simulación de un convertidor analógico-digital de aproximaciones sucesivas de bajo consumo y área reducida, usando la tecnología CMOS de 0.35µm de *Austriamicrosystems*. Así como el aprendizaje y entrenamiento en el uso de las herramientas de diseño microelectrónico de *Cadence* y el kit de diseño de *Austriamicrosystems*.

Para la implementación de este SAR ADC, los esfuerzos se centran en conseguir un área reducida (~0.4mm^2) y un bajo consumo (~30µA). Para conseguir un área pequeña manteniendo una resolución aceptable (12 bits), es necesaria la implementación de un sistema de calibración en el ADC, con la finalidad de compensar los posibles errores de linealidad derivados del uso de un área tan pequeña; el estudio y diseño de este sistema de calibración constituye el objetivo prioritario para este proyecto.

Además del propio diseño del ADC, con este proyecto se intenta obtener una visión general del flujo de diseño de un circuito microelectrónico, pasando por todas las fases: estudio de la topología y simulaciones teóricas, diseño del esquemático, implementación de modelos *Verilog-A/AMS* para simulación, generación y extracción del *layout*, simulaciones de *corners* y *montecarlo* y simulaciones de *toplevel*. Recalcar en este punto, que el objetivo marcado no es el de cubrir hasta el final todas las etapas hasta llegar a la fabricación, pero si tener una visión general de todas ellas y centrarse en aquellos aspectos del diseño más interesantes a la hora de trabajar con este tipo de circuitos.

1.3 CONTENIDOS

El presente Proyecto Fin de Carrera se encuentra divido en tres bloques y doce capítulos. El primero de los bloques aglutina los capítulos del uno al cinco, en los cuales se introduce el proyecto realizado y se hace un estudio del estado del arte actual de los convertidores analógico-digital. En el segundo bloque, del capítulo seis al diez, se describe de forma exhaustiva cada uno de los bloques diseñados para este ADC, centrándose en el sistema de calibración, la parte más innovadora de este proyecto. Y finalmente, en el bloque número tres, que agrupa los dos últimos capítulos, se recoge toda la información relativa a las simulaciones realizadas y los resultados obtenidos, así como las conclusiones finales del proyecto.

En las siguientes líneas se describe, de forma más detallada, el contenido de cada uno de los doce capítulos de este documento.

Este primer capítulo se dedica a describir la motivación, objetivos y contenidos del proyecto.

Los capítulos segundo y tercero hacen referencia a cuestiones generales de los convertidores. En el capítulo segundo se focaliza la atención hacia el funcionamiento teórico de un convertidor, así como a describir los parámetros de medida que definen su calidad. En el capítulo tercero, se profundiza en el estudio de los convertidores de aproximaciones sucesivas y de los DAC's capacitivos.

El capítulo número cuatro describe el flujo de diseño y la planificación seguida para realizar este proyecto, así como las herramientas que han sido empleadas para ello.

En el quinto capítulo se detallan las especificaciones técnicas del convertidor a diseñar, centrándose en todos aquellos aspectos relevantes a tener en cuenta, así como el tipo de topología a implementar y la necesidad de un sistema de calibración para lograr los requerimientos establecidos.

En el sexto capítulo se describe con profundidad la topología de ADC que se va a implementar, resaltando sus aspectos más significativos y haciendo una descripción genérica de cada bloque que la compone y de su funcionalidad.

El capítulo séptimo habla sobre el sistema de calibración que implementa este ADC, resume las necesidades para la calibración y describe paso a paso la secuencia seguida para medir los errores de *matching* del DAC y corregirlos durante el proceso de conversión.

El capítulo número ocho trata sobre el bloque principal de la arquitectura SAR, el convertidor digital-analógico (DAC). En este capítulo se justifica la topología de DAC empleada, así como todas las medidas tomadas para reducir su área al máximo e incorporar la posibilidad de calibrar sus capacidades. También se describen las características de su *layout* y los efectos de los parásitos.

En el noveno capítulo se presenta el bloque digital que implementa la lógica de control. Se define la máquina de estados que ejecuta el algoritmo de búsqueda SAR junto a al proceso de auto-calibración, y se detallan los aspectos más significativos de la descripción *Verilog* y su síntesis.

En el capítulo décimo se detalla el diseño del comparador, describiendo su topología y el sistema de compensación de *offset* que incorpora. Al final del capítulo se presentan algunas gráficas obtenidas tras su simulación y los parámetros que caracterizan al mismo.

En el undécimo capítulo se recopilan los resultados de todos los análisis y simulaciones realizados.

Y finalmente, en el último capítulo se ofrecen las conclusiones y los resultados obtenidos.

Además de todos los capítulos anteriormente citados, al final del documento se adjuntan cuatro anexos con información referida a: los esquemáticos diseñados, los modelos *Verilog-A/AMS* empelados para las simulaciones, el *layout* del DAC y el modelo *MATLAB* del sistema de calibración.

CAPÍTULO 2

Introducción a los Convertidores Analógico-Digital

2.1 PROCESO DE CONVERSIÓN ANALÓGICO-DIGITAL

Un proceso de conversión analógico-digital es aquel que permite partir de una señal continua y llegar a otra señal discreta equivalente. De tal forma que, si posteriormente se aplica el proceso inverso, es posible recuperar la señal continua original a partir de la señal discreta sin haber sufrido en la transformación ningún tipo de pérdida de información.

Desde el punto de vista de un convertidor analógico-digital ideal, el proceso necesario para convertir una señal analógica (continua) en una señal digital (discreta), consta de tres fases: muestreo, cuantificación y codificación.

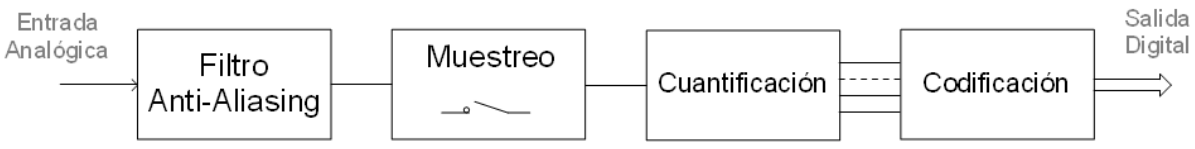


Figura 2.1: Proceso de conversión analógico-digital

Durante la fase de **muestreo** se discretiza la señal en el eje temporal, es decir, la señal pasa de ser de tiempo-continuo a ser de tiempo-discreto. O lo que es lo mismo, se pasa de tener una amplitud de la señal para un conjunto infinito de valores temporales, rango de tiempo continuo, a tener una amplitud de la señal sólo para ciertos instantes de tiempo, conjunto discreto de valores temporales.

El parámetro más importante a definir en esta etapa es lo que se denomina período de muestreo (T_s) o frecuencia de muestreo ($f_s = 1/T_s$). El período de muestro se define como el tiempo transcurrido entre dos muestras consecutivas de la señal.

Idealmente, el muestreo genera una secuencia de deltas cuya amplitud es igual a la de la señal en los instantes de muestreo. Si consideramos un muestreo uniforme de periodo T_s, la señal muestreada resultante sería:

$$x^*(t) = x^*(n \cdot T_s) = \sum x(t) \cdot \delta(t - n \cdot T_s) \qquad \textbf{(2.1)}$$

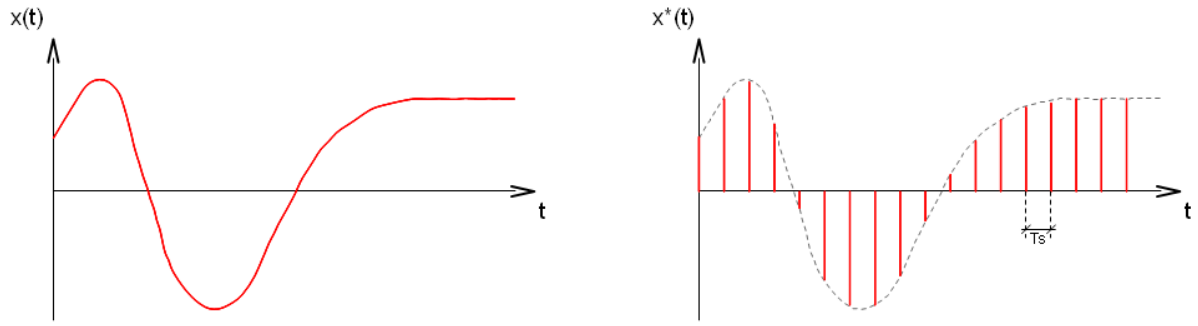


Figura 2.2: Muestreado de una señal

Directamente relacionada con la frecuencia de muestreo, se encuentra la frecuencia máxima que puede tener la señal de entrada para poder ser muestreada y posteriormente recuperada sin problemas. Se demuestra a partir del teorema de *Nyquist*, que la frecuencia de muestreo mínima f_s necesaria, para poder discretizar una señal de frecuencia máxima f_{max} y que posteriormente pueda ser recuperada sin perder información es de:

$$f_s > 2 \cdot f_{max} \qquad \textbf{(2.2)}$$

Si se representa la respuesta en frecuencia de la señal muestreada, se puede observar lo siguiente:

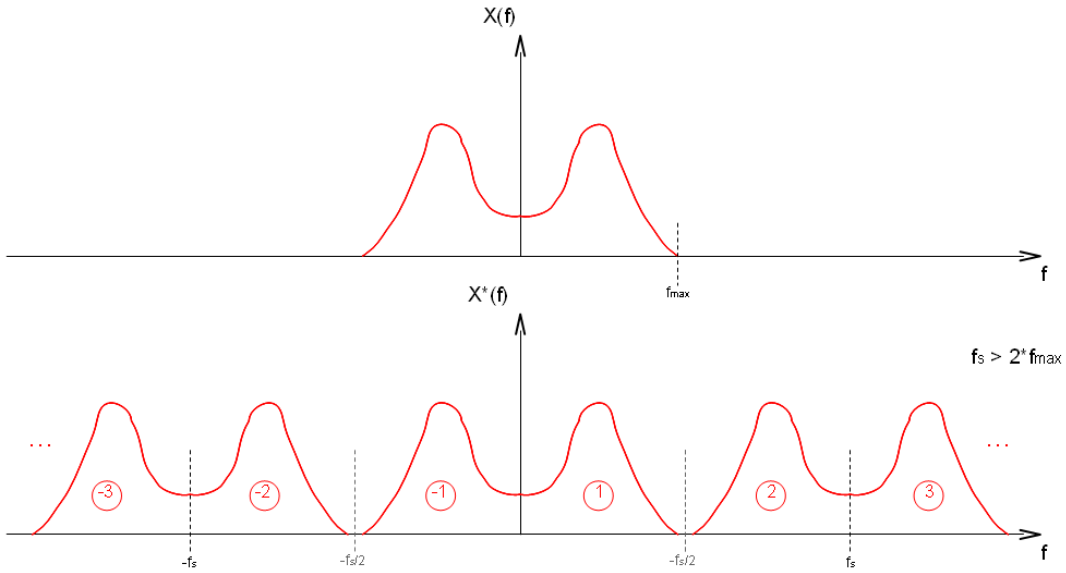


Figura 2.3: Respuesta en frecuencia de una señal muestreada con fs>2·fmax

$$TL[x^*(n\cdot T_s)] = \sum_{-\infty}^{+\infty} X(s - j\cdot n\cdot \omega_s) = \sum_{-\infty}^{+\infty} x(n\cdot T_s)\cdot e^{-n\cdot s\cdot T_s} \qquad \textbf{(2.3)}$$

el espectro de una señal muestreada equivale al espectro de la señal original escalado por $1/f_s$ y repetido de forma periódica cada f_s. Según lo observado, es evidente afirmar que para el caso en que no se cumpla el teorema de *Nyquist*, es decir, para una: $f_s < 2\cdot f_{\max}$, se va a producir solapamiento en el espectro (*aliasing*), por lo que la información contenida en la porción de espectro que quede solapada se perderá, y por lo tanto, recuperar a posteriori la señal analógica original va a ser imposible.

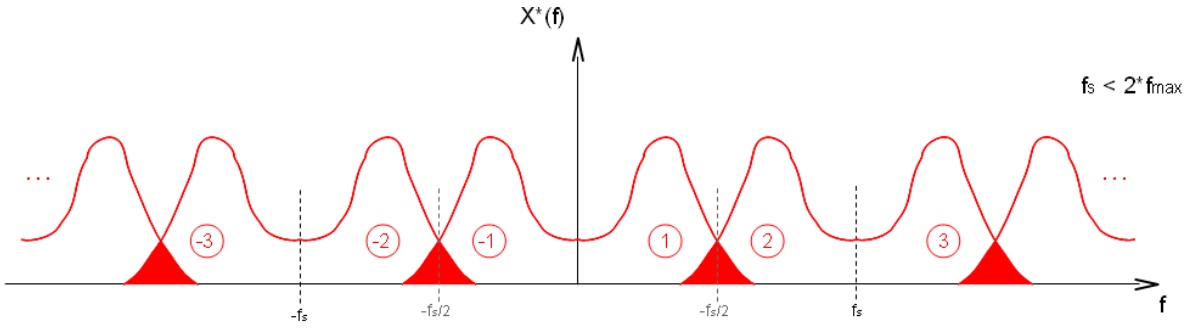


Figura 2.4: Respuesta en frecuencia de una señal muestreada con fs<2·fmax

En la práctica, debido al problema existente con el solapamiento, es imprescindible utilizar un filtro paso-bajo analógico de frecuencia de corte $f_s/2$ (filtro *anti-aliasing*) antes de muestrear, para de esta forma eliminar el ruido y todas aquellas señales indeseadas que se encuentren fuera del espectro de *Nyquist*, y que si no se suprimen van a provocar solapamiento al realizar el muestreo.

La etapa de **cuantificación** es la que se encarga de discretizar la señal en amplitud. Después de la fase de muestreo, se tiene una señal discreta en el dominio temporal pero con unos valores de amplitud continuos; con la cuantificación se consigue discretizar la amplitud de la señal, y que esta pase de variar dentro de un rango de valores continuos a variar en un conjunto de valores discretos. El rango dinámico de la señal de entrada se divide en un conjunto discreto de intervalos, los denominados intervalos de cuantificación (Q_k). Cada intervalo de cuantificación pasa a ser representado por un único valor, que normalmente se corresponde con el valor intermedio del intervalo, de esta forma la señal queda discretizada en amplitud.

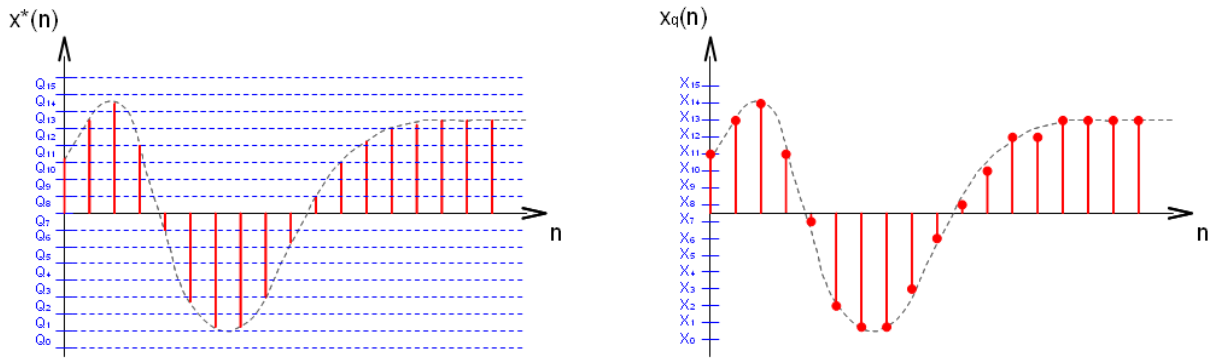


Figura 2.5: Cuantificación de una señal

$$Q = \frac{margen \quad dinámico}{número \quad de \quad intervalos} = \frac{V_{fs}}{M} \tag{2.4}$$

$$X_k = (k + 1/2) \cdot Q \quad , \quad k = 0, ..., M-1 \tag{2.5}$$

Como es evidente, a diferencia de lo que ocurre con el muestreo, durante el proceso de cuantificación es inevitable la pérdida de información. Al asignarle a cada valor de la señal de entrada un intervalo se está sustituyendo el valor original por el valor de representación de dicho

intervalo, con lo cual se está cometiendo un error igual a la diferencia entre esos dos valores. Dado que los valores de representación se encuentran en el centro de los intervalos, es claro que el error de cuantificación queda acotado entre:

$$k \cdot Q < \varepsilon_q < (k+1) \cdot Q \quad , \quad k = 0, ..., M-1 \tag{2.6}$$

A medida que Q *se reduce, el error cometido disminuye, hasta llegar al límite teórico de:* $Q = 0 \ (M = \infty) \rightarrow \varepsilon_q = 0$

El error de cuantificación se suele modelar como una pequeña señal de ruido que se suma a la entrada, esta señal de ruido varía entre $-Q/2$ y $+Q/2$ y presenta la siguiente característica:

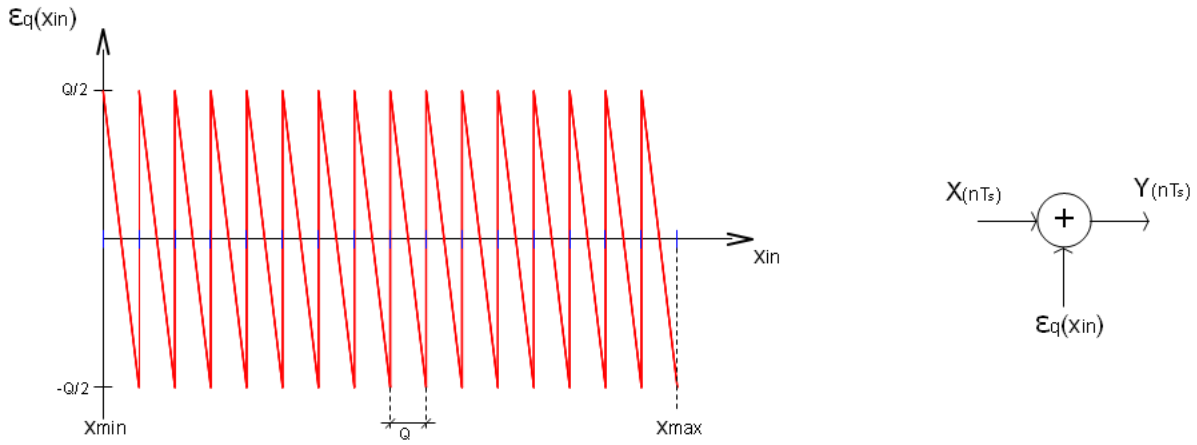
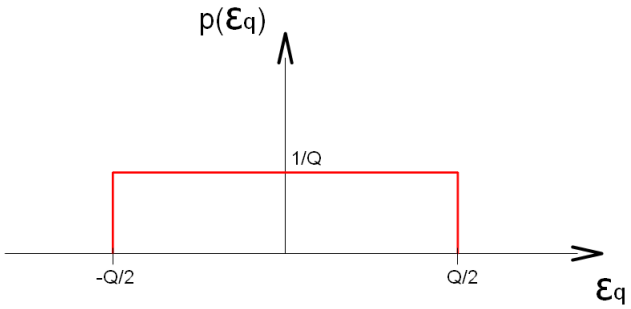


Figura 2.6: Ruido de cuantificación

$$\varepsilon_q(nT_s) = Q \cdot X(nT_s) - Q^2 \cdot (k + 1/2) \quad , \quad X(nT_s) \in [Q \cdot k, Q \cdot (k+1)] \tag{2.7}$$

Tratando la señal de entrada como una variable aleatoria, es posible calcular su función densidad de probabilidad y a partir de ella obtener la potencia del ruido de cuantificación. Para un Q pequeño, se puede suponer que la densidad de probabilidad de la señal de entrada (y por consiguiente de la señal de ruido) es uniforme dentro del intervalo de cuantificación, con lo que se tiene:



$$p(\varepsilon_q) = \frac{1}{Q} \quad , \quad \varepsilon_q \in [{}^{-Q}\!/_{2}, {}^{+Q}\!/_{2}] \qquad \textbf{(2.8)}$$

Figura 2.7: Densidad de probabilidad del ruido de cuantificación

Y de esta forma, la potencia del ruido de cuantificación queda determinada por:

$$N_{rms} = \sqrt{\int_{-\infty}^{+\infty} \varepsilon_q^2 \cdot p(\varepsilon_q) \cdot d\varepsilon_q} = \sqrt{\int_{-Q/2}^{+Q/2} \frac{\varepsilon_q^2}{Q} \cdot d\varepsilon_q} = \sqrt{\frac{Q^2}{12}} = \frac{Q}{\sqrt{12}} \qquad \textbf{(2.9)}$$

con esta expresión se demuestra que el ruido de cuantificación es directamente proporcional al tamaño de los intervalos de cuantificación (Q) o lo que es lo mismo, inversamente proporcional al número de intervalos de cuantificación (M). De esta manera, en el caso ideal de poder trabajar con un número infinito de intervalos de cuantificación ($Q = 0$) el ruido de cuantificación sería nulo, y por lo tanto, el proceso de cuantificación no implicaría una pérdida de información.

La tercera y última fase del proceso de conversión analógico-digital es la **codificación**. La codificación consiste en la asignación de un código a cada muestra de la señal, dependiendo de en qué nivel de cuantificación se encuentre, es decir, a qué intervalo de cuantificación pertenezca. Mayoritariamente se emplean para la codificación el binario natural, BCD o código Gray.

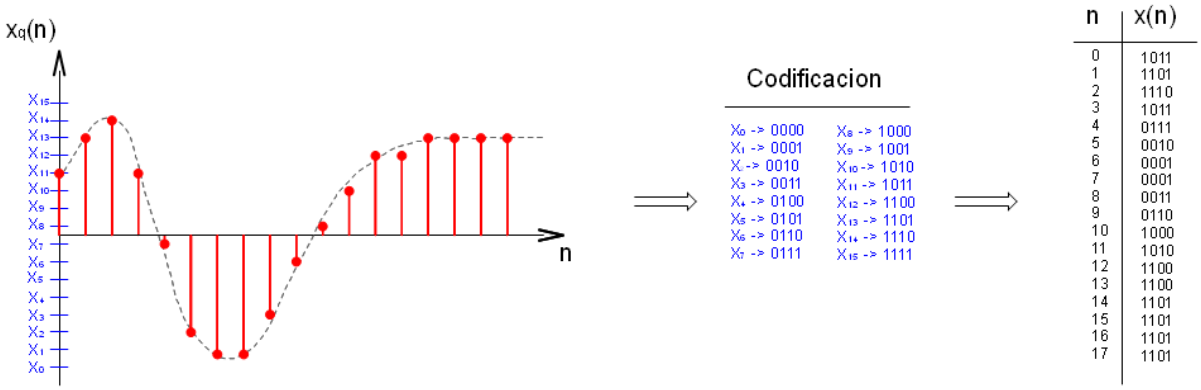


Figura 2.8: Codificación de una señal

Después de superar las tres fases del proceso de conversión: muestreo, cuantificación y codificación, se obtiene como resultado la señal digital (tiempo discreto, amplitud discreta) que representa de forma unívoca a la señal analógica original.

2.2 PARÁMETROS DE CALIDAD DE UN ADC

2.2.1 Función de transferencia ideal

La gran mayoría de los convertidores (ADC y DAC) presentan idealmente la siguiente característica de entrada-salida:

$$V = \frac{V_{fs}}{2^N} \cdot (b_0 \cdot 2^0 + b_1 \cdot 2^1 + ... + b_{N-1} \cdot 2^{N-1}) \qquad \textbf{(2.10)}$$

donde el término escalar V hace referencia a la señal en el dominio analógico, y el vector b define la señal en el dominio digital. V_{fs} es el valor máximo que puede tomar la variable analógica V, lo que se conoce como fondo de escala. N es el número de bits del convertidor.

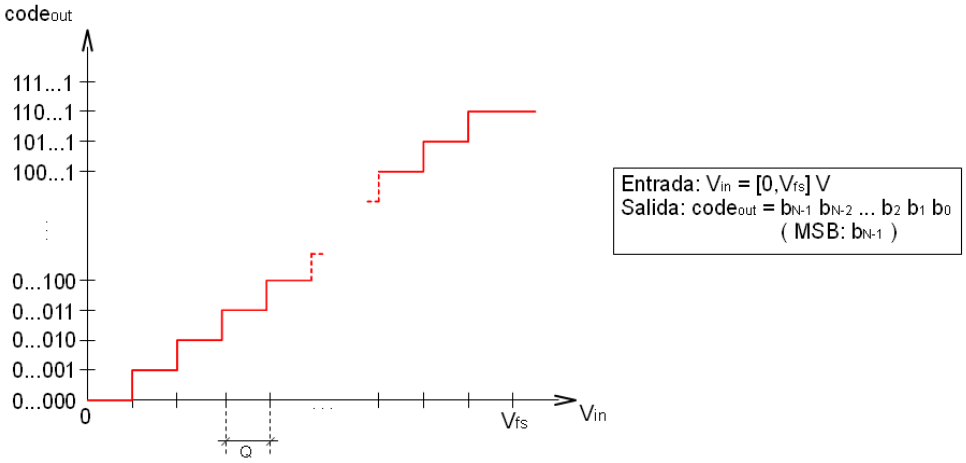


Figura 2.9: Función de transferencia ideal del ADC

La variable V puede representar cualquier magnitud analógica que defina a una señal: tensión, corriente, presión… (normalmente nos referiremos a V como a una tensión) mientras que el vector b describe la palabra binaria de N bits que representa el intervalo de cuantificación al cual se asocia cada muestra de la tensión de entrada

El número de bits (N) de un convertidor determina su resolución. Con N bits se pueden representar hasta 2^N palabras binarias, con lo que se tienen hasta $M = 2^N$ intervalos de cuantificación. A mayor número de bits, mayor número de intervalos de cuantificación y por

consiguiente, menor tamaño para cada uno de ellos (V_{fs}/M) y menor error de cuantificación cometido.

Habitualmente se dice que un convertidor es de N bits de resolución, pero a menudo, también se define el término resolución como: la mínima variación de la tensión analógica a la entrada que provoca un cambio de un LSB en el código de salida. En este caso, se habla de la resolución en términos de la tensión del LSB (V_{LSB}):

$$V_{LSB} = \frac{V_{fs}}{M} = \frac{V_{fs}}{2^N} \quad \textbf{(2.11)}$$

2.2.2 Parámetros de calidad estáticos

Al hablar de parámetros estáticos, se hace referencia a todos aquellos parámetros de calidad de un ADC que se miden bajo condiciones de continua (DC), es decir, con el convertidor trabajando con tensiones continuas a su entrada.

Todos los parámetros de calidad estáticos se miden a partir de la función de transferencia del ADC, comparándola con la del ADC ideal y cuantificando sus diferencias.

A) Error de cuantificación

El error de cuantificación es innato a la propia naturaleza del convertidor, y por lo tanto, es siempre el mismo independientemente del tipo de convertidor, de la resolución… Se define como la diferencia entre la entrada y la salida del ADC.

$$Error_{cuantificación} = Entrada - Salida \quad \textbf{(2.12)}$$

La amplitud del ruido de cuantificación varia siempre entre 0 LSB y 1 LSB, y su carácter aleatorio se traduce en la aparición de un ruido blanco en el espectro de la señal de salida. La potencia del ruido de cuantificación se ha calculado anteriormente, y vale:

$$N_{rms} = \frac{Q}{\sqrt{12}} \quad \textbf{(2.13)}$$

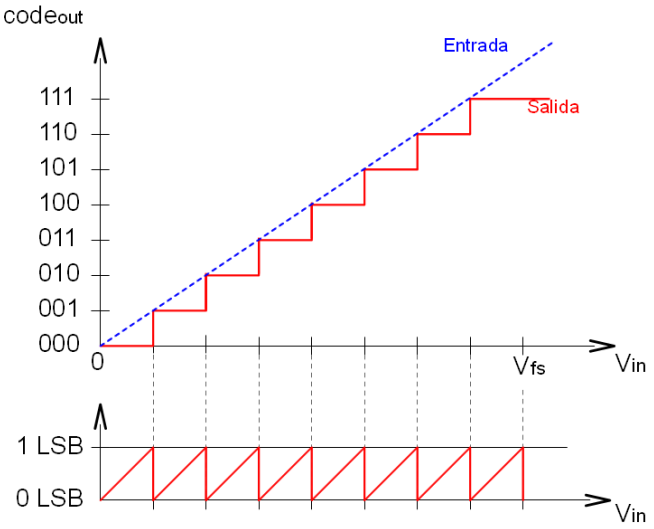


Figura 2.10: Error de cuantificación

B) Error de *offset*

El error de *offset* es aquel que provoca un desplazamiento de toda la función de transferencia del convertidor. Aparece cuando la señal de entrada analógica 0V no genera el código de salida 00…0.

Se define como la desviación de la salida real frente a la ideal para el valor de salida más pequeño. Se puede cuantificar en valor absoluto (voltios), en LSB's, en % ó en ppm del fondo de escala.

$$Error_{offset} = Transicion\_real_{0-1} - Transicion\_ideal_{0-1} \qquad \textbf{(2.14)}$$

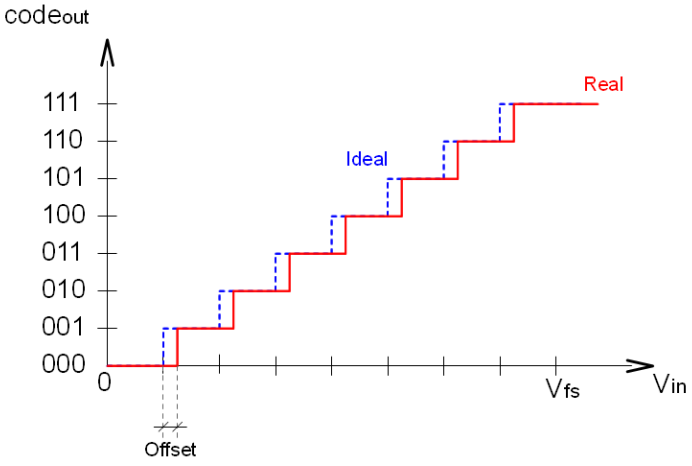


Figura 2.11: Error de offset

Los errores de *offset* en un convertidor no son preocupantes, pues se pueden compensar fácilmente restando de la salida el error de *offset* medido.

C) Error de ganancia o factor de escala

El error de ganancia o factor de escala es aquel que provoca una variación en la pendiente de la función de transferencia del convertidor.

Se define como la diferencia entre las desviaciones de la salida real frente a la ideal para el valor de salida más grande y para el valor de salida más pequeño. Normalmente se cuantifica en % del fondo de escala o en LSB's.

$$Error_{gain} = (Transicion\_real_{(N-2)->(N-1)} - Transicion\_ideal_{(N-2)->(N-1)}) + \\ -(Transicion\_real_{0->1} - Transicion\_ideal_{0->1}) \quad \textbf{(2.15)}$$

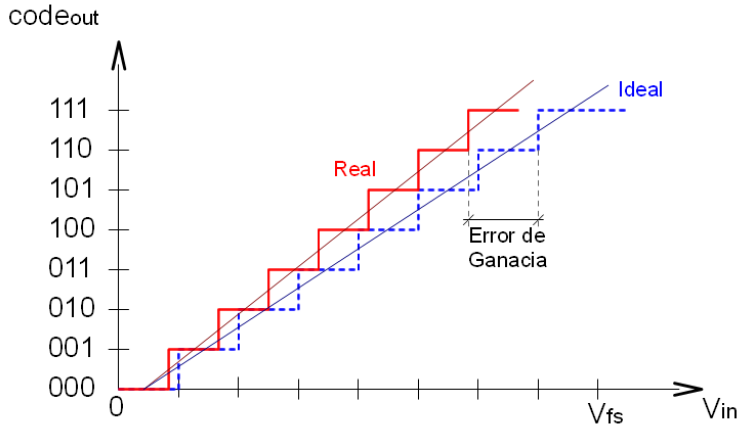


Figura 2.12: Error de ganancia

Al igual que ocurre con el error de *offset*, el error de ganancia también es fácilmente compensable.

D) Error de linealidad

Como indica su nombre, los errores de linealidad son los causantes de las no-linealidades en la función de transferencia del convertidor. Se definen dos parámetros para medir la linealidad de un convertidor:

Error de linealidad diferencial (DNL): se define como la desviación de la salida real frente a la ideal para cada uno de los valores de salida del convertidor. Normalmente se definen los valores máximo y mínimo de DNL.

$$DNL_{code} = Ancho_{code} - Ancho_{ideal} = Ancho_{code} - 1 \qquad \textbf{(2.16)}$$

$$\begin{aligned} DNL_{code} < 0 \quad &\rightarrow \quad Ancho_{code} < 1LSB \\ DNL_{code} = 0 \quad &\rightarrow \quad Ancho_{code} = 1LSB \\ DNL_{code} > 0 \quad &\rightarrow \quad Ancho_{code} > 1LSB \end{aligned}$$

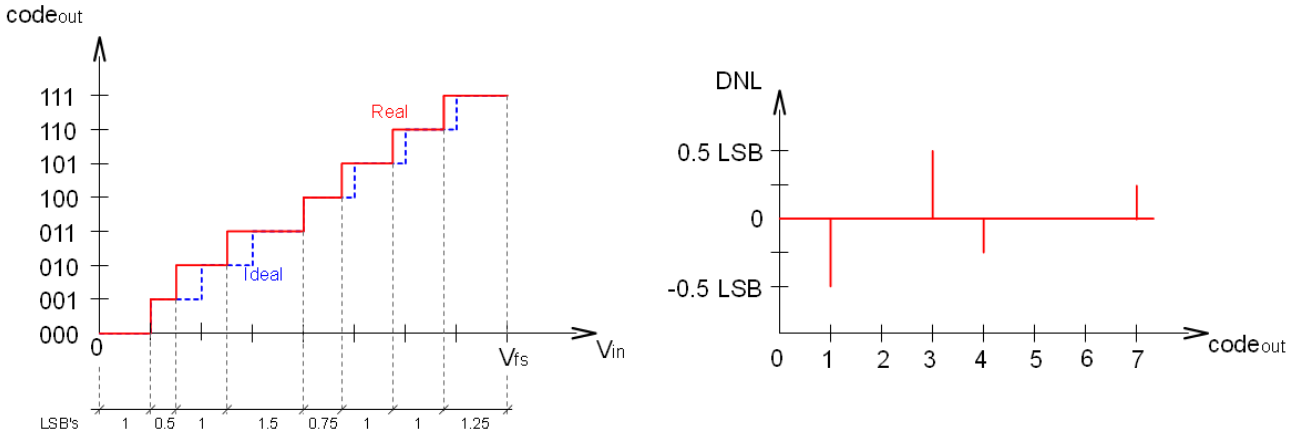


Figura 2.13: Error de linealidad (DNL)

Missing codes: se dice que un convertidor presenta *missing codes* (códigos perdidos) cuando tiene un $DNL_{code} = -1LSB$ en alguno de sus códigos. Esto significa que, para todo el rango de tensiones analógicas de entrada, el ADC nunca generará ese código a su salida.

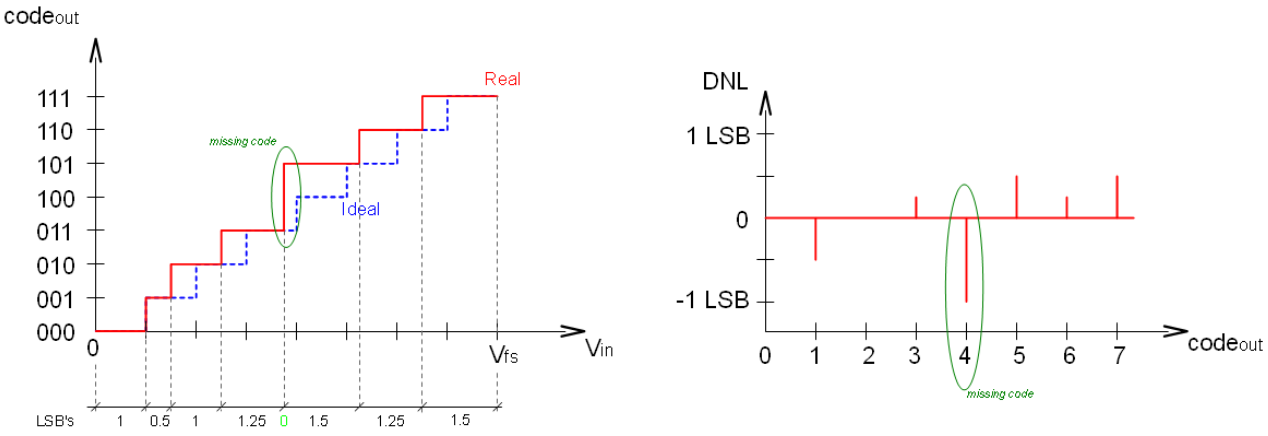


Figura 2.14: Error de linealidad (missing codes)

Error de linealidad integral (INL): se define como la desviación de la salida real del convertidor frente a la salida ideal. El error de linealidad integral (INL) se calcula como la integral discreta del error de linealidad diferencial (DNL).

$$INL_{code} = INL_{code-1} + DNL_{code}$$
$$INL_0 = 0 \qquad \textbf{(2.17)}$$

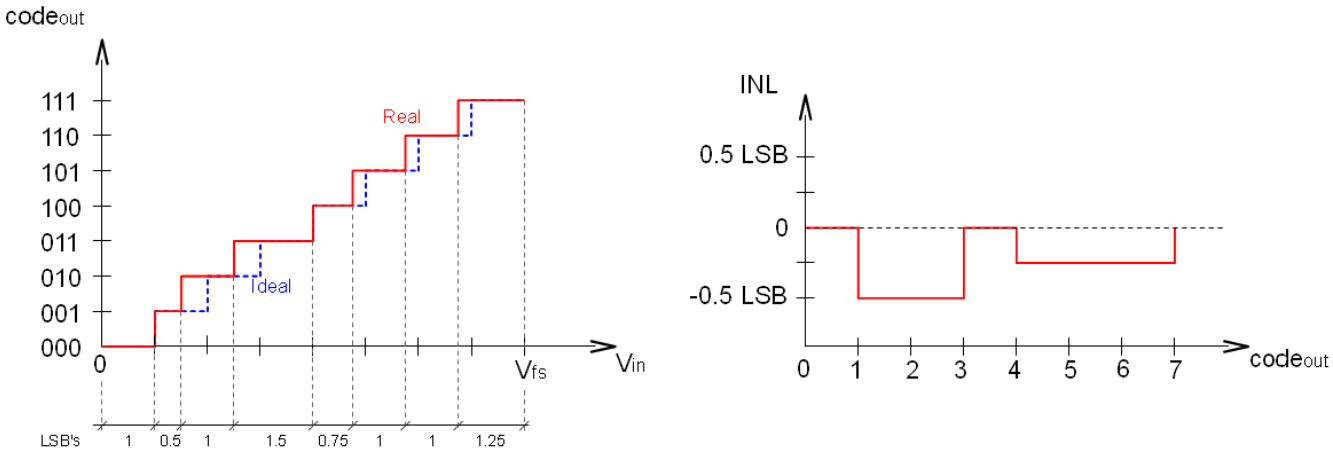


Figura 2.15: Error de linealidad (INL)

Los errores de linealidad DNL e INL se cuantifican en LSB's. Conseguir un ADC con una buena característica de linealidad es fundamental, pues los problemas de linealidad son difícilmente compensables, a diferencia de lo que ocurre con los errores de *offset* o de ganancia.

2.2.3 Parámetros de calidad dinámicos

Los parámetros de calidad dinámicos estudian el comportamiento del ADC en condiciones de alterna (AC), es decir, con una señal variante con el tiempo a la entrada.

Existen varias herramientas para el estudio de este tipo de parámetros, pero la más habitual es el uso de la FFT, en la siguiente figura se puede ver la configuración típica para realizar este tipo de test.

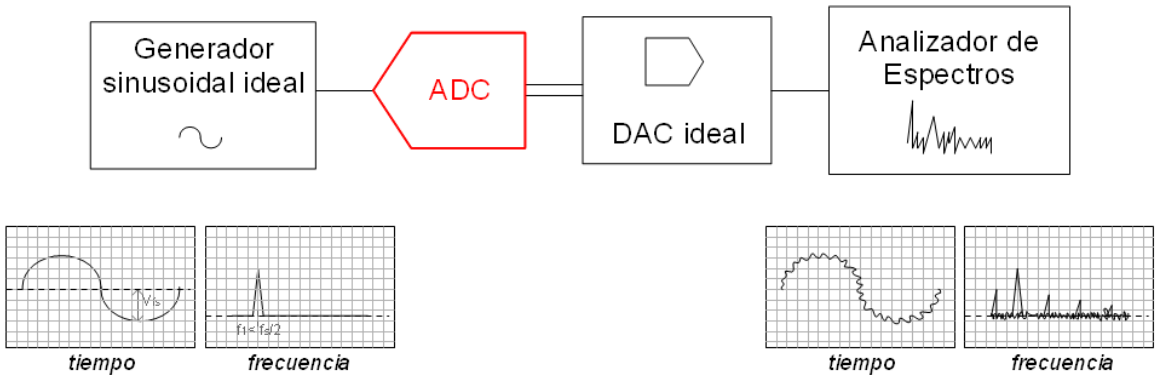


Figura 2.16: Medida de parámetros dinámicos

A partir del análisis de la FFT, se observan las siguientes características del espectro de la señal de salida:

- Componente de continua (DC): aparece debido al error de *offset* del propio convertidor, y al mantenimiento del valor de salida del convertidor durante el tiempo transcurrido entre dos muestras consecutivas.

- Ruido de fondo: es el generado por el proceso de cuantificación (ruido de cuantificación). Solo disminuye si se dispone de un número mayor de bits al cuantificar.

- Armónicos: aparecen por pequeñas imperfecciones en la forma de la señal de salida (a la salida del convertidor no se tiene una senoidal pura). Estas imperfecciones vienen derivadas de los problemas de linealidad del ADC.

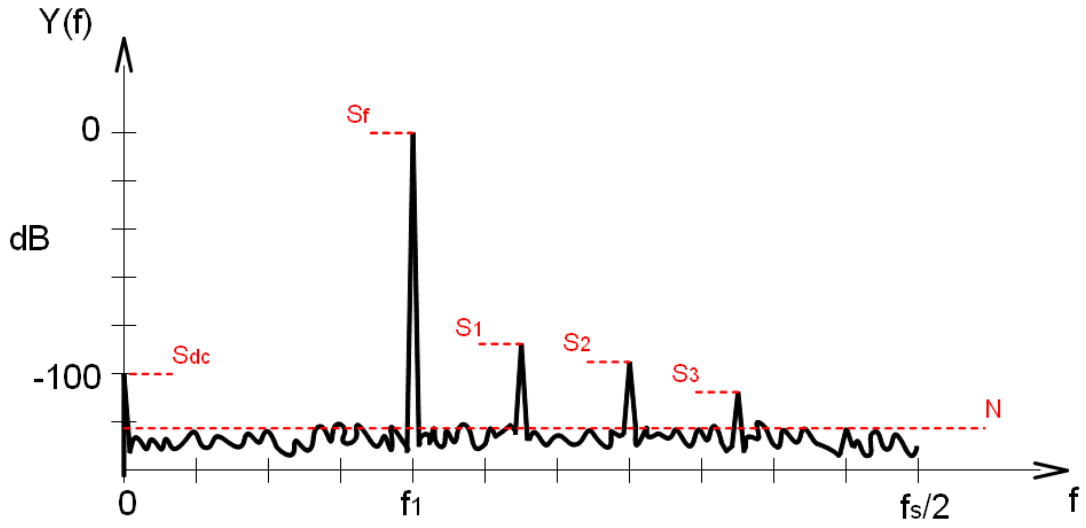


Figura 2.17: Espectro de la señal de salida de un ADC real

A) Relación señal-a-ruido (SNR)

La relación Señal-a-Ruido (SNR) de un convertidor se define como el cociente entre la potencia de la señal y la potencia del ruido de fondo.

$$SNR = \frac{S_f}{N} \tag{2.18}$$

Como se ha explicado anteriormente, el ruido de fondo está vinculado directamente al ruido de cuantificación (ruido blanco) y su potencia eficaz, calculada en el punto anterior, vale:

$$N = \frac{Q}{\sqrt{12}} \tag{2.19}$$

por otro lado, la potencia de la señal se calcula como la potencia eficaz de una senoidal a fondo de escala, y su valor es de:

$$S_f = \sqrt{\frac{1}{T}\cdot\int_0^T \left(\frac{V_{fs}}{2}\right)^2 \cdot \sin^2\ 2\pi\cdot f\cdot t} = \sqrt{\frac{1}{T}\cdot\int_0^T \frac{V_{fs}^2}{4}\cdot \sin^2\ 2\pi\cdot f\cdot t} = \sqrt{\frac{V_{fs}^2}{8}} = \sqrt{\frac{Q\cdot 2^{N\ 2}}{8}} = \frac{Q\cdot 2^N}{\sqrt{8}} \tag{2.20}$$

con lo que al final se llega a la conclusión de que, la SNR depende exclusivamente del tamaño del intervalo de cuantificación (Q), o lo que es lo mismo, del número de bits utilizados para cuantificar. De esta forma, la SNR de un convertidor queda totalmente definida mediante la siguiente expresión:

$$SNR = 6.02 \cdot N + 1.78 \quad dB \tag{2.21}$$

La expresión definida anteriormente determina la SNR máxima alcanzable por un convertidor de N bits. En un caso real, la SNR obtenida será siempre menor, pues existen otros factores, además del ruido de cuantificación, que influyen sobre el ruido de fondo, aumentándolo y haciendo que disminuya la SNR. Éste es el caso del ruido $K \cdot T / C$ en los convertidores capacitivos.

B) Relación señal-a-ruido + distorsión (SINAD)

La SINAD se define de forma muy similar a la SNR, se calcula como el cociente entre la potencia de la señal y la potencia del ruido de fondo más la distorsión.

$$SINAD = \frac{S_f}{N + S_1 + S_2 + ...} \tag{2.22}$$

En este parámetro se incluyen todas las fuentes de imperfección medidas en la FFT, con lo que constituye una de las medidas más significativas a la hora de determinar las características dinámicas de un convertidor.

C) Distorsión armónica total (THD)

En la distorsión armónica total (THD) se mide el ratio entre la componente de frecuencia fundamental y los armónicos. Normalmente se consideran para el cálculo los cuatro armónicos más significativos de la FFT.

$$THD = \frac{S_1 + S_2 + ...}{S_f} \tag{2.23}$$

Como se ha explicado anteriormente, los armónicos son el resultado de las no-linealidades del

convertidor, por lo que este parámetro es el que se emplea para cuantificar las no-linealidades del ADC.

D) Rango dinámico libre de espúreos (SFDN)

Se define el rango dinámico libre de espúreos (SFDN) como el ratio entre la componente fundamental y el pico espúreo de mayor amplitud. Habitualmente, uno de los armónicos del espectro es el que establece el espúreo de mayor amplitud.

$$SFDN = \frac{S_f}{S_i|_{\max}} \tag{2.24}$$

CAPÍTULO 3

Fundamentos del SAR ADC y del DAC Capacitivo

3.1 EL CONVERTIDOR DE APROXIMACIONES SUCESIVAS (SAR ADC)

El convertidor analógico-digital denominado convertidor de aproximaciones sucesivas (SAR ADC), es aquel que implementa una arquitectura compuesta por tres bloques: un convertidor digital-analógico (DAC), un comparador y una lógica de control (SAR). Y que basa su funcionamiento en un proceso iterativo, mediante el cual, va probando diferentes códigos digitales hasta encontrar el que se corresponde con la señal analógica de entrada.

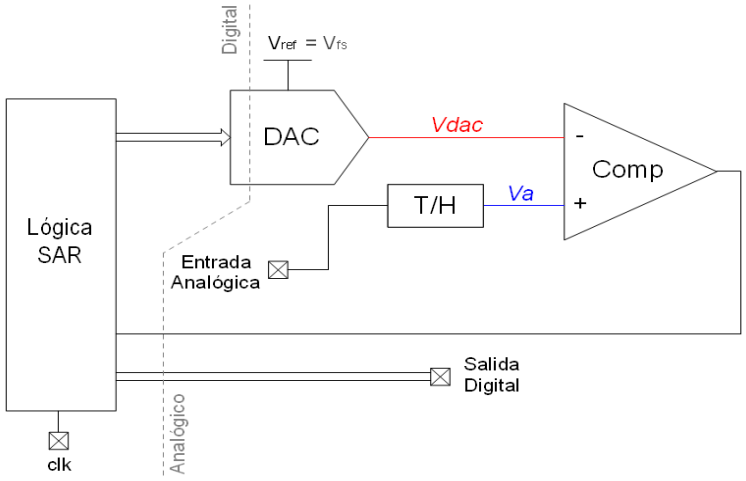


Figura 3.1: Arquitectura de un SAR ADC single-ended

La idea es la siguiente: el sistema de muestreo T&H toma una muestra de la señal analógica de entrada y mantiene su valor (V_a) en la entrada positiva del comparador. Por otro lado, en la entrada negativa del comparador, se tiene la señal analógica (V_{dac}) proveniente del DAC, la cual se corresponde con el valor analógico que representa el código digital generado por la lógica SAR. De este modo, el comparador determina si la señal analógica de entrada (V_a) es de valor superior o inferior a la señal generada por el DAC (V_{dac}), y a partir de esta información, la lógica SAR generará otro código digital de búsqueda que se aproxime más al valor de la señal de entrada y lo aplicará de nuevo al DAC. Este proceso se repite, de forma iterativa, hasta encontrar el código digital de N bits, cuya señal analógica asociada (V_{dac}), se aproxime lo máximo posible a la señal analógica de entrada (V_a).

El código digital resultante tras la búsqueda, se corresponderá al valor analógico de entrada convertido.

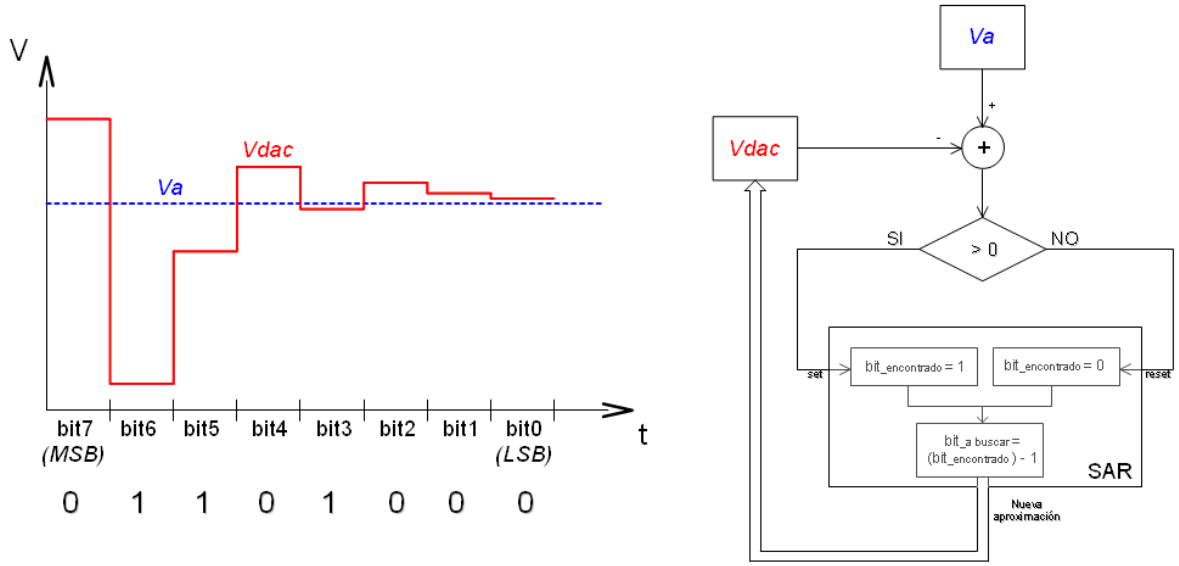


Figura 3.2: Señales a la entrada del comparador. Diagrama de flujo del funcionamiento del SAR ADC

La lógica SAR, es la que se encarga de establecer y aplicar la estrategia de búsqueda para encontrar el código digital de N bits que mejor representa la señal analógica de entrada. Como su nombre indica, la lógica SAR implementa un algoritmo de búsqueda basado en aproximaciones sucesivas, este algoritmo funciona de la siguiente forma: se compara en primer lugar el valor analógico de entrada con la mitad del valor de fondo de escala ($V_{fs}/2$), en caso de que la señal de entrada sea mayor que ese valor, se vuelve a comparar esta vez con $3 \cdot V_{fs}/4$, si por el contrario

fuese menor, se comparara con $V_{fs}/4$, y así sucesivamente, dividiendo cada vez el margen dinámico en trozos de la mitad de tamaño, hasta el máximo de resolución posible.

Con las aproximaciones sucesivas, la variación de la señal analógica (V_{dac}) en cada paso, es la mitad que en el paso anterior, lo que se corresponde con el cambio de un bit en el código de búsqueda, comenzando por el bit más significativo. Es decir, si se considera un rango dinámico que varía entre 0 y V_{fs} voltios, en la primera iteración, el MSB de la palabra binaria permitirá distinguir entre las señales de entrada que se encuentren por arriba o por debajo de la mitad del fondo de escala ($V_{fs}/2$), obteniéndose de esta forma el valor correspondiente para este bit. Fijado ya el valor del bit MSB, se puede restringir la búsqueda para el siguiente bit menos significativo, que podrá determinar si la señal de entrada es superior o inferior a la mitad de un rango dinámico de tamaño mitad al anterior, con lo que el umbral de comparación será de $3 \cdot V_{fs}/4$ si el MSB vale uno ó de $V_{fs}/4$ si el MSB se ha fijado a cero. Este proceso continuará iterativamente hasta llegar a determinar el bit LSB de la palabra binaria que se está buscando.

Al concluir la búsqueda, se habrá obtenido un código binario de N bits que se corresponderá con el valor de la señal analógica de entrada, con un error de $\pm Q/2$, correspondiente al error de cuantificación del convertidor.

A continuación, se expone un ejemplo de búsqueda SAR para un convertidor de 8 bits. En la figura se puede observar cómo va variando la salida del DAC (V_{dac}) en cada iteración, y cuáles son los valores correspondientes de los bits que se van encontrando.

Iteración	Código de búsqueda	Tensión de salida del DAC (*Vdac*)	Resultado comparación	Bit encontrado
1	1000_0000	5.0000000 V	0	Bit7 = 0
2	0100_0000	2.5000000 V	1	Bit6 = 1
3	0110_0000	3.7500000 V	1	Bit5 = 1
4	0111_0000	4.3750000 V	0	Bit4 = 0
5	0110_1000	4.0625000 V	1	Bit3 = 1
6	0110_1100	4.2187500 V	0	Bit2 = 0
7	0110_1010	4.1406250 V	0	Bit1 = 0
8	0110_1001	4.1015625 V	0	Bit0 = 0
Tensión de referencia: *Vfs* = 10V . Tensión analógica a convertir: *Va* = 4.1V				

Tabla 3.1: Ejemplo de búsqueda SAR para un convertidor de 8 bits

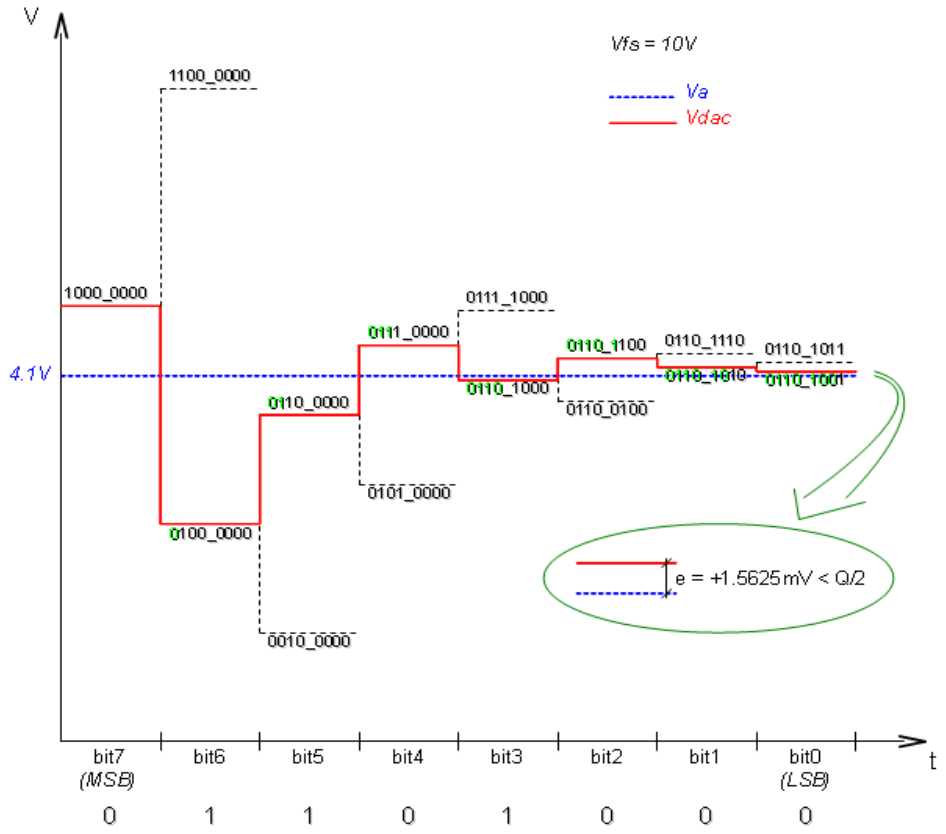


Figura 3.3: Ejemplo de búsqueda SAR para un convertidor de 8 bits

Para un SAR ADC con N bits de resolución, se requiere de N+1 ciclos de reloj para realizar cada conversión, con independencia de cuál sea la señal de entrada a convertir, con lo que el *throughput* del convertidor vendrá fijado únicamente por la frecuencia de reloj y el número de bits utilizados. Por otro lado, la resolución requerida para el comparador deberá ser, de al menos, N+1 bits y su velocidad lo suficientemente elevada para poder hacer una comparación por cada ciclo de reloj. Y en cuanto a la referencia del DAC, es claro que para N bits de resolución, la referencia (V_{ref}) necesitará, como mínimo, una precisión de N+1 bits.

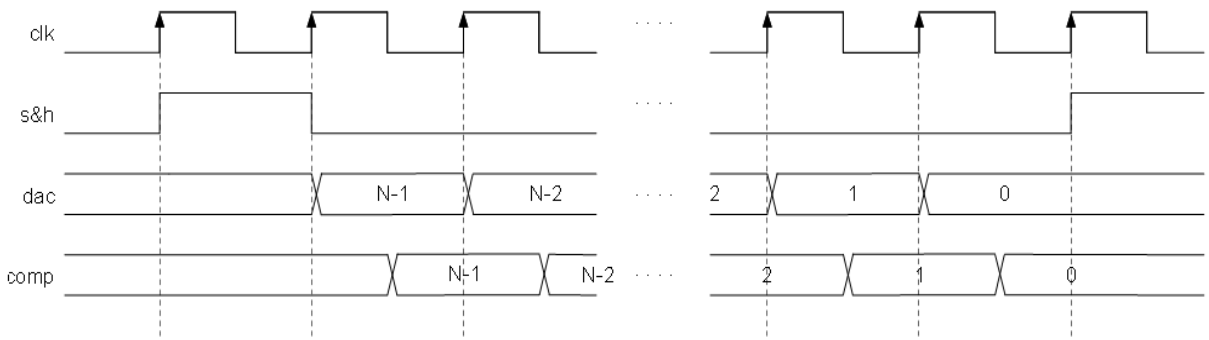


Figura 3.4: Cronograma de funcionamiento de un SAR ADC

Las principales ventajas de este tipo de convertidores son: su sencillez, la velocidad de conversión constante y la facilidad con la que se puede incrementar el número de bits (resolución) y la velocidad. El punto más desfavorable lo constituye la necesidad de tener que emplear bloques críticos para su diseño, como son el DAC y el comparador.

3.2 EL DAC CAPACITIVO

Los convertidores digital-analógico capacitivos, también conocidos como *charge redistribution DAC's*, están formados básicamente por un *array* de capacidades binarias y un conjunto de *switches*. Esta arquitectura de DAC es muy popular en la implementación de ADC's de aproximaciones sucesivas, pues presenta características que la hacen muy interesante para su uso en este tipo de ADC's, como son: la eficiencia energética, la rapidez y el sistema de T&H incorporado.

Una posible implementación de DAC capacitivo se muestra en la siguiente figura:

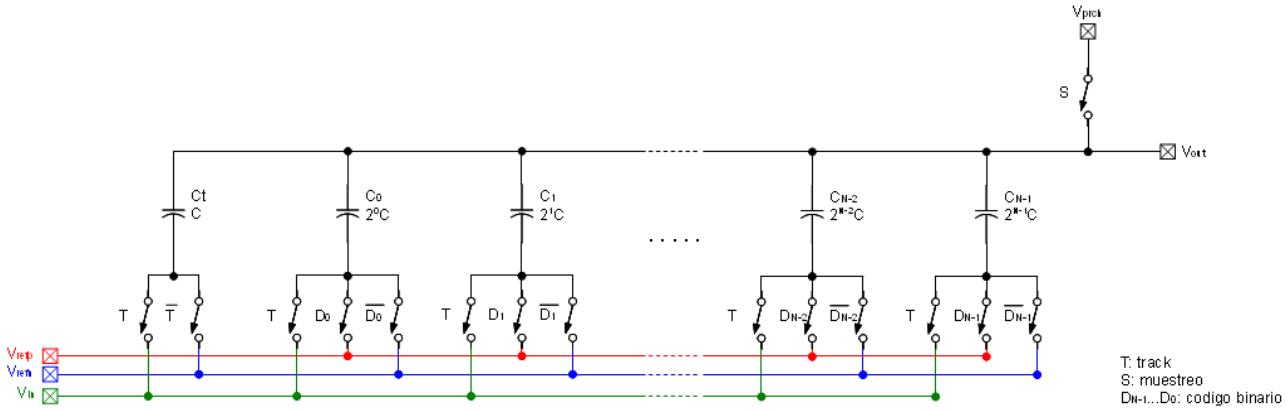


Figura 3.5: DAC capacitivo de N bits

Como se puede apreciar, la estructura de este tipo de DAC es muy sencilla. Las capacidades binarias se interconectan con sus *top-plates* a un mismo nodo común, que constituye la salida del DAC, y con sus *bottom-plates* a uno de los *switches*, mediante los cuales se selecciona una de las tensiones de referencia (V_{refp} ,V_{refn}) o la tensión analógica de entrada (V_{in}). Por otro lado, se utiliza un *switch* conectado al nodo común de salida, con la finalidad de poder precargar los *top-plates* de las capacidades a una tensión fija (V_{prch}).

Dejemos de lado por un momento, el *switch* para la precarga del nodo común, y los *switches* de conmutación con la tensión de entrada (V_{in}).En esta situación, se tiene un *array* de capacidades binarias conectadas en paralelo, y de capacidad total $2^N \cdot C$ (el valor C, define la unidad de capacidad y puede tomar cualquier valor).

Supongamos que inicialmente todas las capacidades están descargadas, si ahora se hacen conmutar los *switches* de los *bottom-plates* de cada capacidad a V_{refp} ó V_{refn} en función del código de una

palabra binaria de N bits: $D_{N-1}D_{N-2}...D_0$ (donde D_{N-1} es el bit MSB), la tensión que se obtendrá a la salida (V_{out}) será la tensión generada por el divisor capacitivo formado entre: la suma de todas las capacidades conectadas a V_{refp} y el resto de capacidades (conectadas a V_{refn}), es decir:

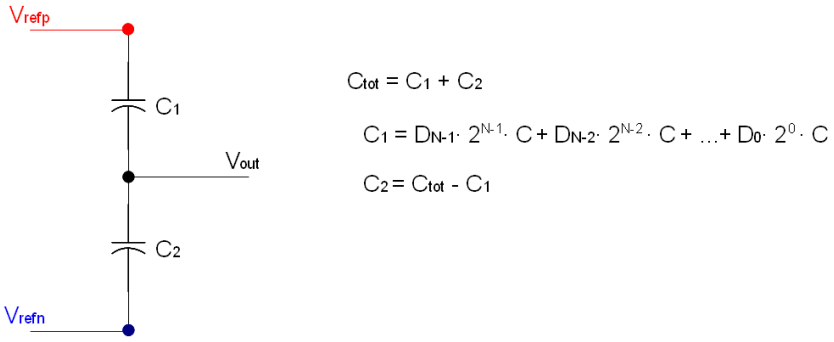


Figura 3.6: Circuito equivalente de un DAC capacitivo de N bits

$$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{C_1}{C_{tot}} = \frac{V_{refp} - V_{refn}}{2^N} \cdot \left(D_{N-1} \cdot 2^{N-1} + D_{N-2} \cdot 2^{N-2} + ... + D_0 \cdot 2^0 \right) \quad \textbf{(3.1)}$$

o lo que es lo mismo:

$$V_{out} = (V_{refp} - V_{refn}) \cdot \sum_{k=0}^{N-1} D_k \cdot 2^{k-N} \quad \textbf{(3.2)}$$

De esta manera, se ha obtenido una tensión analógica V_{out}, que depende directamente del código digital aplicado. El sistema funciona como un convertidor digital-analógico.

En el siguiente ejemplo con tres bits, se ilustra con más detalle el funcionamiento de esta arquitectura:

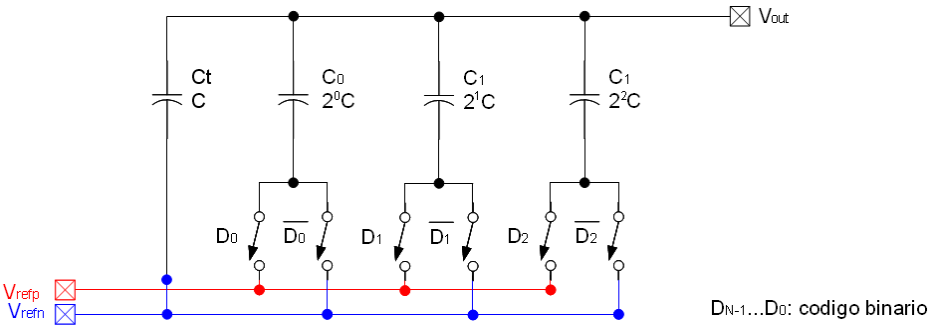


Figura 3.7: DAC capacitivo de 3 bits

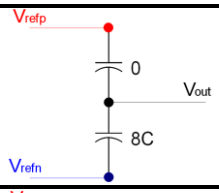
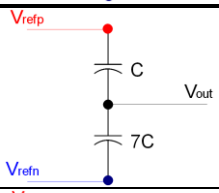
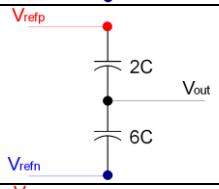
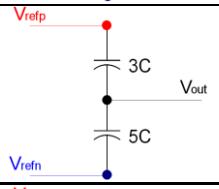
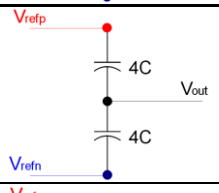
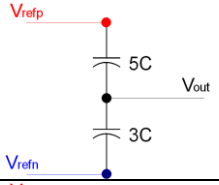
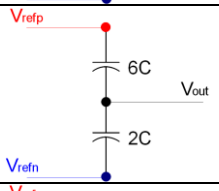
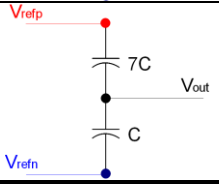
Código de entrada ($D_{N-1},\ldots, D_0$)	Circuito equivalente	Tensión de salida V_{out} (V)
000		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{0}{8} = 0$
001		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{1}{8}$
010		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{2}{8}$
011		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{3}{8}$
100		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{4}{8}$
101		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{5}{8}$
110		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{6}{8}$
111		$V_{out} = (V_{refp} - V_{refn}) \cdot \frac{7}{8}$

Tabla 3.2: Ejemplo de DAC capacitivo de 3 bits

Si a la estructura analizada anteriormente, se le añade el *switch* de precarga y los *switches* de conmutación de la tensión de entrada (V_{in}), es posible implementar con ella un sistema de T&H. Lo cual constituye una da las ventajas principales de la arquitectura *charge redistribution*.

La posibilidad de poder integrar un T&H en un DAC capacitivo se basa en el principio de redistribución de cargas, según el cual, la carga total almacenada en las capacidades del DAC permanecerá constante en todo momento, independientemente de cuál sea la tensión aplicada sobre el *bottom-plate* de cada capacidad.

Si inicialmente, en lugar de tener las capacidades del DAC descargadas, se precargan con una carga equivalente a la tensión de entrada V_{in} en un instante dado, posteriormente, al aplicar el código binario sobre las capacidades, esta carga inicial se redistribuirá entre todas ellas de tal manera que, ahora, la tensión en el nodo de salida (V_{out}) ya no solo dependerá del código binario aplicado, sino también, del valor de la V_{in} en ese instante.

Para poder precargar las capacidades en función del valor de la V_{in} en un instante determinado, es decir, para poder muestrear el valor de la tensión de entrada sobre el nodo de salida (V_{out}), se requiere de las siguientes fases:

1- Fase de seguimiento (TRACK): inicialmente, el *switch* de precarga del nodo común se activa, forzando la tensión de los *top-plates* de las capacidades a un valor fijo, en este caso por sencillez se tomara $V_{prch} = 0V$ pero podría valer cualquier otro valor. Al mismo tiempo, se activan los *switches* de conmutación de la V_{in}, con lo que los *bottom-plates* de las capacidades están variando con la tensión V_{in}. En esta situación, la carga total acumulada en el DAC será:

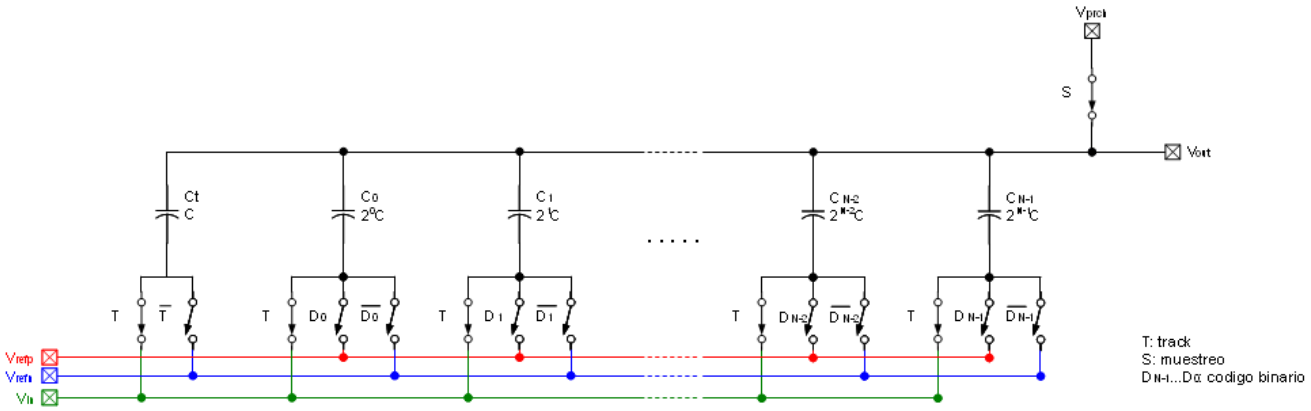


Figura 3.8: DAC capacitivo de N bits en modo track

$$Q_{total} = C_{total} \cdot V_{total} = 2^N \cdot C \cdot (V_{prch} - V_{in}(t)) = 2^N \cdot C \cdot (0 - V_{in}(t)) \quad \textbf{(3.3)}$$

$$V_{out} = V_{prch} = 0V \quad \textbf{(3.4)}$$

Como se observa en la expresión anterior, la carga total va variando en función de $V_{in}(t)$, es decir, sigue a la tensión de entrada. Estamos en modo de seguimiento.

2- Muestreo de la tensión de entrada: en el instante (t_s) en el que se abre el *switch* de precarga y se fijan a V_{refn} los *bottom-plates* de todas las capacidades, se produce el muestreo de la V_{in}. Ahora, la carga total del DAC ya no variará siguiendo a la $V_{in}(t)$, sino que tendrá un valor fijo e igual a la carga equivalente inyectada por la $V_{in}(t)$ en el instante t_s.

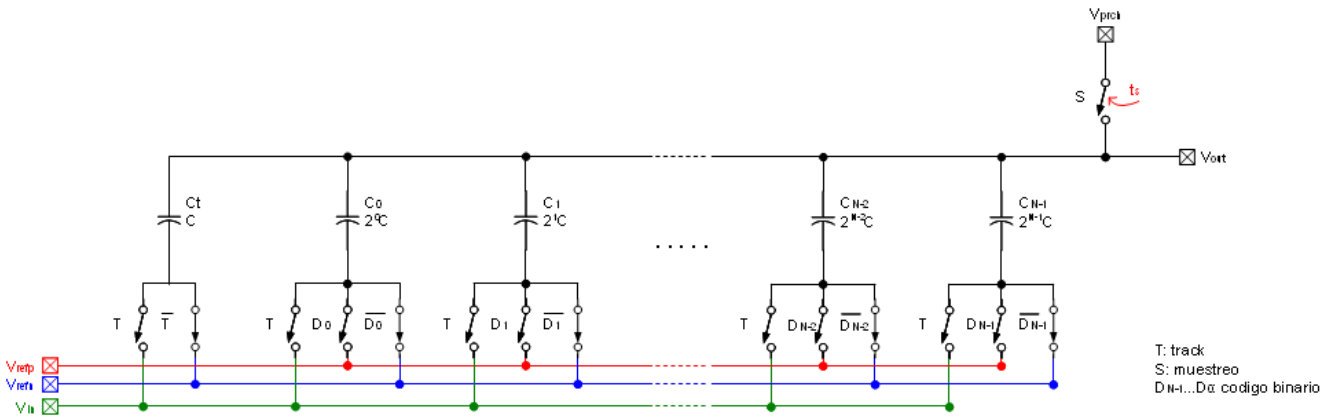


Figura 3.9: DAC capacitivo de N bits en modo muestreo

carga en el instante justo antes de muestrear:

$$Q_{total}(t_s^-) = C_{total} \cdot V_{total}(t_s^-) = 2^N \cdot C \cdot (0 - V_{in}(t_s)) \quad \textbf{(3.5)}$$

carga en el instante justo después de muestrear:

$$Q_{total}(t_s^+) = C_{total} \cdot V_{total}(t_s^+) = 2^N \cdot C \cdot (V_{out} - V_{refn}) \quad \textbf{(3.6)}$$

Según el principio de redistribución de cargas:

$$Q_{total}(t_s^-) = Q_{total}(t_s^+) \quad \rightarrow \quad V_{out} = -V_{in}(t_s) + V_{refn} \quad \textbf{(3.7)}$$

3- Fase de mantenimiento (HOLD): una vez muestreada la tensión de entrada, el valor $-V_{in}(t_s)$ se mantendrá de forma permanente en el nodo de salida, estamos en el modo de mantenimiento. A partir de ahora, si se aplica sobre el DAC cualquier código digital, éste modificará la tensión del nodo de salida (V_{out}) sumando al valor de $-V_{in}(t_s)$ muestreado la tensión correspondiente al código aplicado:

$$V_{out} = -V_{in}(t_s) + V_{dac} \tag{3.8}$$

$$V_{dac} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \left(D_{N-1} \cdot 2^{N-1} + D_{N-2} \cdot 2^{N-2} + ... + D_0 \cdot 2^0 \right) \tag{3.9}$$

La arquitectura de DAC capacitivo aquí presentada es muy popular, pues presenta características interesantes que la hacen adecuada para muchas aplicaciones. En cuanto a su consumo, es mínimo, pues al tratarse de una arquitectura capacitiva, el consumo viene determinado exclusivamente por la carga y descarga dinámica de sus capacidades. Esto también influye en su velocidad de funcionamiento, que es muy elevada. Por otro lado, incorpora un sistema de T&H que la hace muy atractiva, al poder evitarse la utilización de otro bloque especifico para estas funciones. Además de todo esto, se trata de una arquitectura sencilla, lo que posibilita que su modificación o la ampliación a resoluciones mayores no constituyan un verdadero problema.

En cuanto a sus características negativas habría que destacar su sensibilidad a los parásitos, pues una capacidad parásita sobre el nodo común afectaría directamente a la salida del DAC, quedando esta atenuada. Por lo que respecta al tamaño, tampoco es muy óptima, pues por cada bit adicional que se añada se dobla el área requerida.

CAPÍTULO 4

Planificación y flujo de diseño

Como se indica en el capítulo introductorio de este documento, uno de los objetivos de este proyecto, además del estudio y la implementación de este SAR ADC con calibración, es el aprendizaje y entrenamiento en el uso de las herramientas básicas de diseño microelectrónico. Teniendo en cuenta todo ello, se ha seguido una planificación acorde a los conocimientos iniciales de los que se partía y los objetivos que se querían alcanzar.

Como resultado, la planificación seguida para realizar este proyecto ha quedado dividida en tres fases o bloques principales.

En el primero de ellos, denominado *Topology Understanding*, se pretende hacer un estudio teórico de los convertidores ADC de tipo SAR, estudiar su topología básica partiendo de una estructura *single-ended* sencilla de 8 bits y posteriormente, por extensión de la anterior, llegar hasta la topología *fully-differential* de 12 bits que finalmente se implementará.

Además, se inicia el estudio teórico de los DAC's capacitivos, el bloque principal de cualquier SAR ADC, se parte de una arquitectura de 8 bits sencilla y de un modelo matemático de DAC implementado con MATLAB (véase anexo D) para entender su funcionamiento y las distintas posibilidades que ofrece, y posteriormente, tras obtener la estructura final a implementar, se elabora el *schematic* correspondiente en *Cadence* y se hacen las primeras simulaciones.

Al mismo tiempo, durante esta fase del proyecto se dedica tiempo a familiarizarse con las herramientas de diseño y el uso del simulador.

Todo el estudio y las conclusiones alcanzadas durante esta fase, se detallan en los capítulos 2, 3 y 6 de esta memoria.

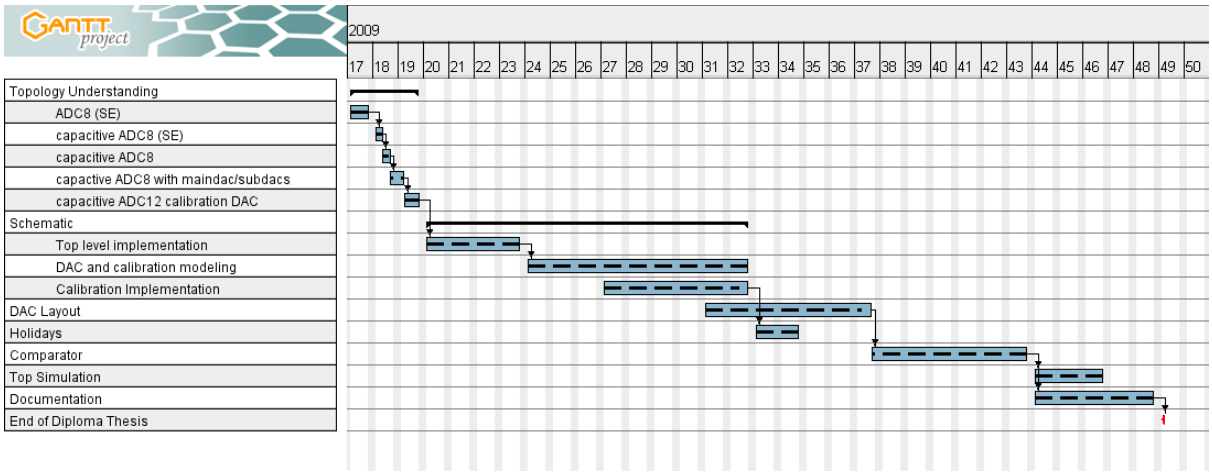


Figura 4.1: Planificación del PFC

La segunda de las fases (*Schematic* y *DAC layout*), empieza con la preparación de las primeras simulaciones *toplevel* del SAR ADC junto con la arquitectura de DAC diseñada, con la finalidad de corroborar el buen funcionamiento de la topología. Para ello, se elaboran los primeros modelos *Verilog*/*Verilog-AMS* del comparador y de la lógica de control.

En esta fase, el mayor esfuerzo se centra en la implementación del algoritmo de calibración, se hace un estudio teórico del mismo y un modelo en MATLAB para entender perfectamente su comportamiento. Finalmente tras haberlo simulado tanto de forma matemática en MATLAB, como mediante modelos en *Cadence*, se pasa a sintetizar la lógica digital que lo implementará. A partir de la descripción *Verilog* y de las herramientas de síntesis digital requeridas, se implementa el denominado bloque de lógica de control, el cual aglutina el sistema de calibración junto al algoritmo de búsqueda SAR.

Esta segunda etapa concluye con la elaboración del *layout* del DAC, para la cual se requiere conocer previamente ciertas características importantes de este tipo de *layouts* y familiarizarse con las herramientas de: edición de *layout*, LVS y extracción.

El trabajo realizado durante esta segunda fase del proyecto queda reflejado en los capítulos 7, 8 y 9.

La tercera y última fase (*Comparator* y *Top Simulation*), se centra únicamente en el estudio teórico

y el diseño de un comparador adecuado para los requerimientos exigidos por este ADC.

Como última tarea del proyecto se llevan a cabo diversas simulaciones *toplevel*, haciendo trabajar todos los bloques en conjunto: lógica de control, DAC y comparador, y verificando su correcto funcionamiento.

Todos los detalles a cerca del comparador diseñado se encuentran descritos en el capítulo número diez de esta memoria.

Como se puede apreciar en la planificación descrita anteriormente, el flujo de diseño a seguir a la hora de abordar este proyecto es de tipo *top-down*, es decir, va de alto a bajo nivel. Se parte del diseño de la topología del ADC, una vez comprobado que funciona correctamente y responde a las especificaciones esperadas, se va bajando poco a poco dentro de la jerarquía y se van implementando cada uno de los bloques que la componen. Finalmente, con las simulaciones de *toplevel*, se va comprobando el funcionamiento de cada uno de los bloques dentro de la jerarquía y el funcionamiento global de todo el sistema.

Para representar de forma más clara el flujo de diseño seguido y el orden en el que se van implementando cada uno de los bloques, se muestra el siguiente gráfico.

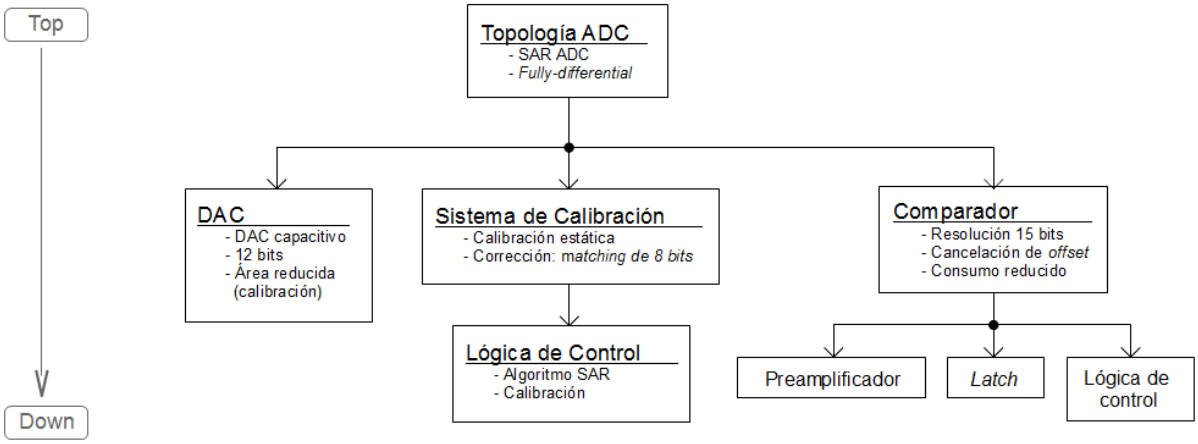


Figura 4.2: Flujo de diseño

Para concluir con este capítulo, a continuación se presenta una tabla con la relación de herramientas que se han utilizado en la realización de este proyecto.

Herramientas de Diseño Analógico	Cadence Design Framework II, vers. C5.10.41.500.6.131. Kit de diseño: HITKIT v3.70 de *Austriamicrosystems* Utilizando las herramientas de simulación Spectre (simulación analógica) y Ncsim (simulación mixta)
Herramientas de Diseño Digital	Synopsys 2007.12rc (sintetizador digital)
Herramienta auxiliares	Matlab, vers. R2007a Microsoft Excel 2003

Tabla 4.1: Relación de herramientas empleadas

CAPÍTULO 5

Especificaciones

5.1 ESPECIFICACIONES

Antes de comenzar con el diseño de cualquier circuito, es imprescindible en primer lugar, establecer de forma clara y unívoca los requerimientos exigidos por la aplicación a la que va destinado, y definir las condiciones en las que va a tener que trabajar. Para este proyecto en particular, las especificaciones que se van a definir no están sujetas a los requerimientos de ninguna aplicación en concreto, pues se trata de un SAR ADC de propósito general, pero sí se va a exigir que cumpla con unas características muy determinadas: bajo consumo y área reducida, las cuales lo van hacer interesante respecto a otros SAR ADC de la misma clase.

Teniendo en cuenta lo anterior y recordando que, el objetivo de este Proyecto Fin de Carrera no trata el cubrir totalmente todas las etapas de diseño hasta llegar a la fabricación, las especificaciones establecidas van a ser de carácter general, sin ser estrictas en algunos aspectos como: condiciones de funcionamiento, rango dinámico, impedancia de entrada… y centrándose primordialmente en las características más relevantes de un convertidor: resolución, *throughput*, área y consumo.

Dicho todo esto, en la siguiente tabla se resumen las especificaciones definidas para este proyecto:

Tipo de convertidor	SAR (convertidor de aproximaciones sucesivas)
Topología	*Fully-differential*
Resolución	12 bits
Throughput	10kSPS
Consumo	~30µA
Área	~0.4mm^2 (Austriamicrosystems C35 process)

Tabla 5.1: Especificaciones del ADC

5.2 PLANIFICACIÓN DEL DISEÑO

Conocidas las especificaciones del diseño, es momento de plantearse en que partes del circuito se va a focalizar el esfuerzo para lograr cada uno de los requerimientos establecidos. En este proyecto, la mayor dificultad radica en cumplir las especificaciones de área y consumo, con lo que va a ser necesario identificar aquellas partes del diseño que van a tener una influencia mayor sobre estos parámetros.

Como se ha visto en el capítulo número tres, un convertidor SAR ADC está compuesto por tres bloques básicos: la parte digital de control, el DAC y un comparador. En este caso, se desea implementar una topología *fully-differential* con lo que se necesitarán dos DAC´s en lugar de uno, como ocurría para la topología *single-ended*.

En cuanto al consumo, se puede afirmar que el comparador va a ser el bloque más limitante, pues la lógica digital tiene un consumo despreciable, y el DAC presenta un consumo teórico cero, al implementarse mediante un *array capacitivo*. Teniendo en cuenta este análisis, el esfuerzo de diseño para cubrir los requerimientos de consumo se va a centrar en el comparador. Diseñar un comparador con un consumo de corriente de unos 30µA y la rapidez suficiente para que el sistema pueda trabajar a 10kSPS va a ser el objetivo marcado.

En cuanto a los requerimientos de área, el bloque que más puede preocupar es el DAC. Para un DAC capacitivo de 12 bits, el número de capacidades requeridas es bastante elevado (4096 capacidades unitarias) además, el DAC es el elemento crítico que va a determinar las características de linealidad del ADC, por lo que se hace imprescindible implementar un *layout* muy cuidado, que garantice un buen *matching* entre capacidades y unos parásitos mínimos, con la finalidad de obtener una buena linealidad. Todo esto hace que el área requerida para este DAC se incremente significativamente, estimándola en torno a unos $0.25mm^2$ (sin tener en cuenta el área necesaria para el rutado). Si se dispone de un presupuesto de $0.4mm^2$ para la totalidad del diseño y los dos DAC's solos ya consumen más de esta área, ésta solución no va a ser factible para alcanzar los requerimientos de área impuestos.

5.2.1 Necesidad de calibrar

Como solución al problema de área excesiva en el DAC, se propone la implementación de un sistema de calibración.

El principal inconveniente que se tiene con los DAC's capacitivos de cierta resolución (a partir de 12/13 bits), es la dificultad de conseguir un *layout* con un *matching* de las capacidades lo suficientemente bueno como para que la linealidad del DAC no se vea comprometida. De hecho, para DAC's con resoluciones mayores de 15 bits, se hace imprescindible el empleo de algún tipo de técnica de calibración para corregir los errores de *matching*, pues es imposible mantener una buen *matching*, y por consiguiente una buena linealidad, con estas resoluciones tan grandes.

Estos sistemas de calibración, basan su funcionamiento en medir las errores de *matching* de las capacidades del DAC, y posteriormente, tras aplicar un cierto algoritmo durante la conversión, compensar los errores de linealidad generados por el efecto de este mal *matching*. La principal ventaja de estos sistemas es que permiten implementar DAC's de área reducida a la vez que se trabaja con altas resoluciones.

En este proyecto, donde los requerimientos de área son tan estrictos, no es mala idea implementar uno de estos sistemas de calibración, pues con ellos no se hace imprescindible conseguir un *layout* con requerimientos de *matching* tan elevados, lo cual propicia que el área pueda ser reducida considerablemente. Se estima que con el uso de un sistema de calibración apropiado, se puede trabajar con un DAC de área casi tres veces menor a la original, con lo que en este caso se reduciría hasta los 0.09mm^2.

Más adelante, en el capítulo siete, se describe con profundidad el sistema de calibración que ha sido implementado en este proyecto.

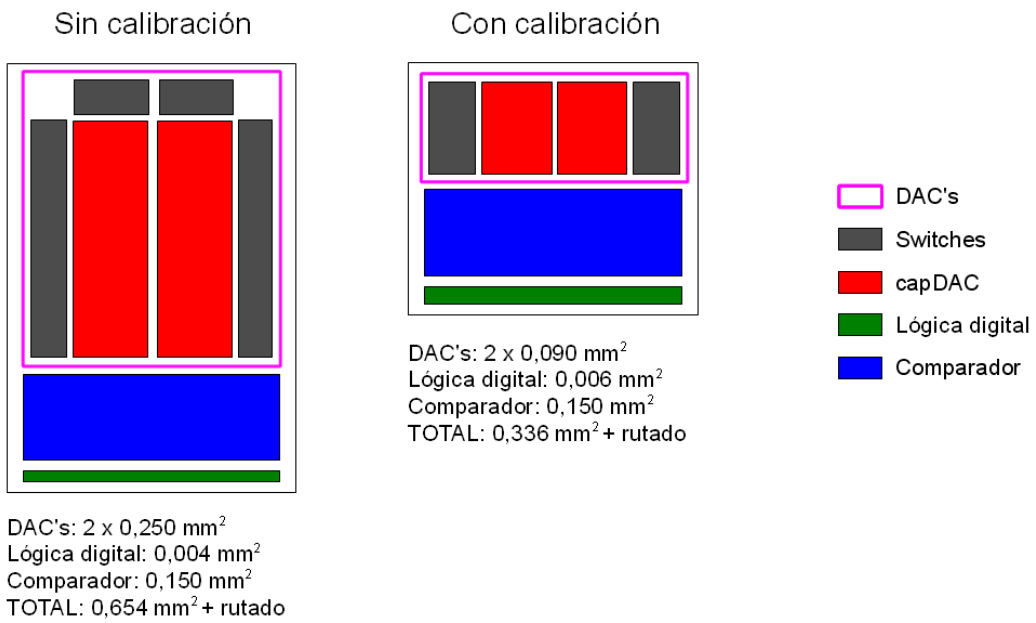


Figura 5.1: Estimación del área. Con calibración / Sin calibración

BLOQUE II

DISEÑO DEL SISTEMA

CAPÍTULO 6

Topología del SAR ADC

6.1 TOPOLOGÍA *FULLY-DIFFERENTIAL*

Según se indica en las especificaciones, el convertidor a implementar debe ser de tipo *fully-differential*. Esta topología es comúnmente utilizada en numerosos diseños, pues frente a la topología *single-ended* presenta características que la hacen muy atractiva, como son:

- La inmunidad frente al ruido de modo común.

- La cancelación de algunos armónicos generados por las no linealidades del circuito, lo que mejora la distorsión armónica total (THD) del sistema.

- La expansión del rango dinámico de entrada al doble. Para un convertidor SAR ADC, como en el caso de este proyecto, esta característica es favorable en cuanto al diseño, pues se traduce en una tensión del LSB (V_{LSB}) de valor doble, con lo que la precisión requerida por el comparador no será tan elevada.

Estas ventajas no son gratuitas, pues en las topologías *fully-differential* la complejidad y el tamaño del circuito aumentan considerablemente con respecto a las *single-ended*, lo que repercute directamente en un área mayor y un consumo más elevado.

En el tercer capítulo de este documento se introdujo el principio de funcionamiento de los convertidores de aproximaciones sucesivas (SAR), basándose para ello, en una topología de

convertidor *single-ended*. Para este proyecto se requiere de una topología *fully-differential*, que en base funciona de igual manera que una *single-ended*, pero con la diferencia de que ahora se va a trabajar con señales diferenciales a la entrada, lo cual va a requerir de una estructura con dos ramas: rama positiva y rama negativa.

Por extensión de la *single-ended*, se ha llegado a la siguiente topología para el SAR ADC *fully-differential* a implementar en este proyecto:

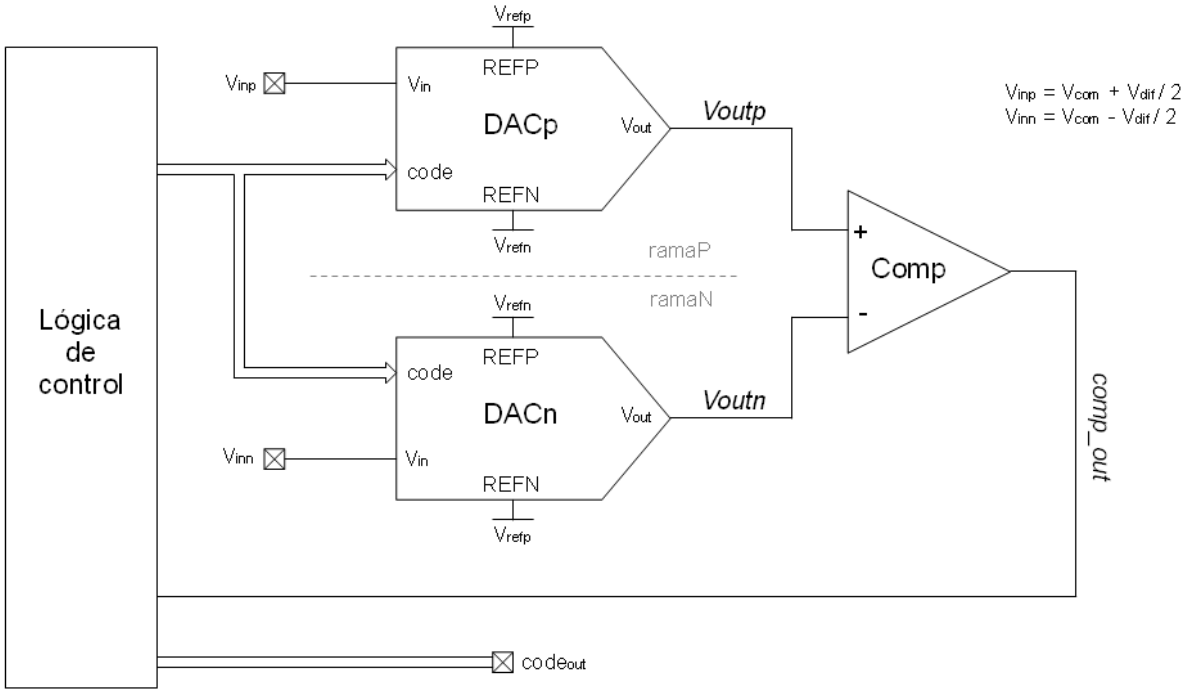


Figura 6.1: Topología SAR ADC fully-differential

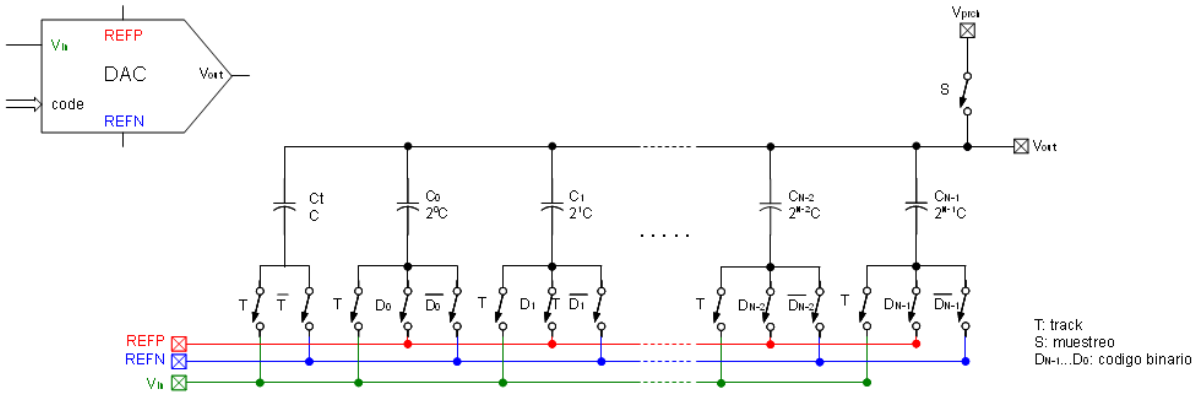
En una topología SAR *fully-differential* se requiere de dos DAC's (uno para la rama positiva y otro para la negativa), un comparador y un bloque digital de control. Al trabajar con DAC's capacitivos, no es necesario ningún bloque de T&H, pues como se vio en el capítulo tres, esta arquitectura de DAC permite el muestreo de la señal de entrada.

El principio de funcionamiento SAR se ejecuta de forma complementaria en las dos ramas: por un lado, en la rama positiva, el DACp muestrea la tensión de entrada positiva (V_{inp}) y convierte los códigos de búsqueda (positivos), y por otro, en la rama negativa, el DACn muestrea la tensión de entrada negativa (V_{inn}) y convierte los mismos códigos de búsqueda anteriores, pero negados (negativos). Para conseguir esto, como se observa en la figura, los DAC's conectan sus referencias

de forma simétrica, es decir, el DACp tiene: $REFP = V_{refp}$ y $REFN = V_{refn}$,y el DACn: $REFP = V_{refn}$ y $REFN = V_{refp}$.

Teniendo en cuenta lo anterior, las tensiones generadas por ambos DAC's serán de valores opuestos y simétricas con respecto al modo común de la tensión de entrada (V_{com}):

RECORDATORIO: DAC capacitivo



$$V_{out} = V_{prch} - V_{in} + V_{dac}$$

$$V_{dac} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot (D_{N-1} \cdot 2^{N-1} + ... + D_0 \cdot 2^0)$$

Tensiones de entrada:

$$V_{com} = \frac{(V_{refp} - V_{refn})}{2} \quad \textbf{(6.1)}$$

$$V_{inp} = V_{com} + \frac{V_{dif}}{2} \quad \textbf{(6.2)}$$

$$V_{inn} = V_{com} - \frac{V_{dif}}{2} \quad \textbf{(6.3)}$$

Tensiones aplicadas por los DAC's:

$$V_{dacp} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot (D_{N-1} \cdot 2^{N-1} + ... + D_0 \cdot 2^0) \quad \textbf{(6.4)}$$

$$V_{dacn} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot (\overline{D}_{N-1} \cdot 2^{N-1} + ... + \overline{D}_0 \cdot 2^0) \quad \textbf{(6.5)}$$

Tensiones de salida:

$$V_{outp} = V_{prch} - V_{inp} + V_{dacp} = V_{prch} - (V_{com} + \frac{V_{dif}}{2}) + V_{dacp} \quad \textbf{(6.6)}$$

$$V_{outn} = V_{prch} - V_{inn} + V_{dacn} = V_{prch} - (V_{com} - \frac{V_{dif}}{2}) + V_{dacn} \quad \textbf{(6.7)}$$

Si se toma como tensión de precarga de los DAC's (V_{prch}) la tensión de modo común, se obtiene lo siguiente:

$$V_{prch} = V_{com} \quad \rightarrow \quad \begin{cases} V_{outp} = V_{com} - (V_{com} + \frac{V_{dif}}{2}) + V_{dacp} = -\frac{V_{dif}}{2} + V_{dacp} & \textbf{(6.8)} \\ V_{outn} = V_{com} - (V_{com} - \frac{V_{dif}}{2}) + V_{dacn} = +\frac{V_{dif}}{2} + V_{dacn} & \textbf{(6.9)} \end{cases}$$

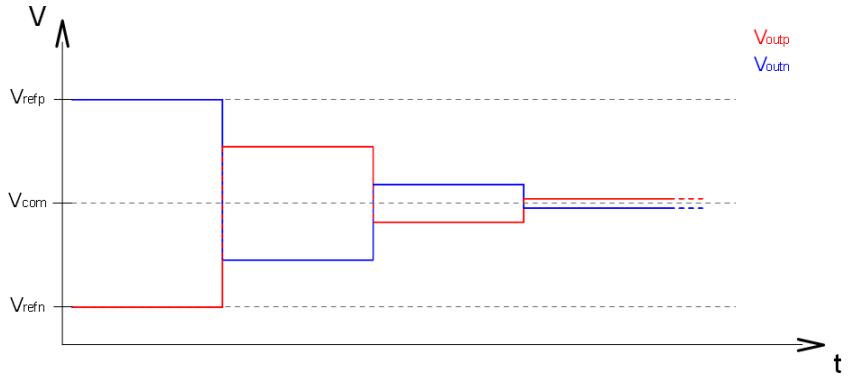
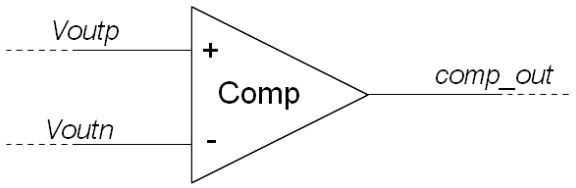


Figura 6.2: Tensión de salida del DACp y DACn

Las salidas de ambos DAC's son comparadas, determinándose de esta forma si el código de búsqueda aplicado genera una tensión diferencial ($V_{dacp} - V_{dacn}$) mayor o menor a la tensión diferencial de entrada (V_{dif}) a convertir; con esta información, la lógica de control aplica el algoritmo SAR y genera un nuevo código de búsqueda. Repitiendo el ciclo durante N pasos, se encuentra el código binario de N bits que representa a la V_{dif} de entrada.



$$V_{outp} = -\frac{V_{dif}}{2} + V_{dacp}$$

$$V_{outn} = +\frac{V_{dif}}{2} + V_{dacn}$$

Figura 6.3: Detalle del comparador

$$V_{comp} = V_{outp} - V_{outn} = -\frac{V_{dif}}{2} + V_{dacp} - \frac{V_{dif}}{2} - V_{dacn} = (V_{dacp} - V_{dacn}) - V_{dif} \quad \textbf{(6.10)}$$

$$comp\_out = \begin{cases} 0 \ , & V_{comp} < 0 \\ 1 \ , & V_{comp} > 0 \end{cases} \quad \textbf{(6.11)}$$

La función de transferencia de este SAR ADC *fully-differential* quedaría de la siguiente forma:

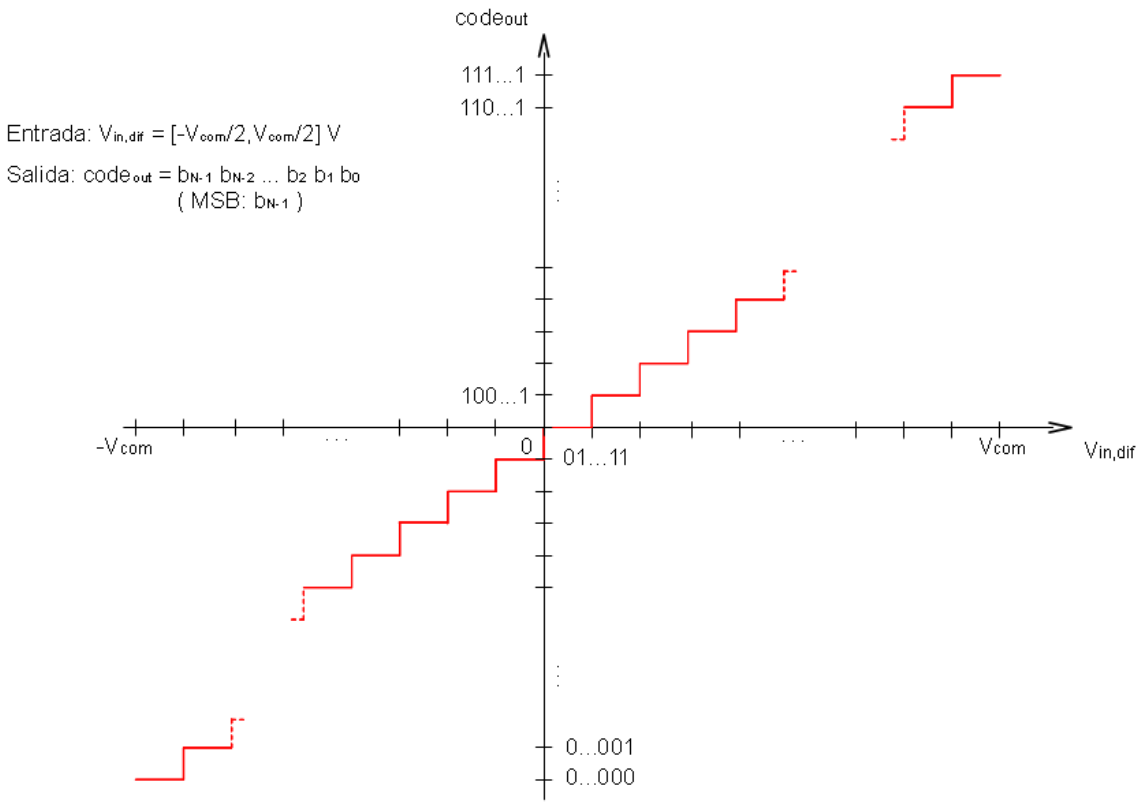


Figura 6.4: Función de transferencia del SAR ADC fully-differential

6.2 SISTEMA DE PRECARGA Y MUESTREO

Como se expuso en el capítulo número tres (al explicar el funcionamiento de los DAC's capacitivos) y se ha recordado anteriormente, para que los DAC's puedan realizar las funciones de muestreo requieren de una tensión de precarga (V_{prch}). Teóricamente, esta tensión de precarga puede tomar cualquier valor, pero en la práctica va a ser necesario fijarla a $V_{prch} = V_{com}$, para conseguir que las salidas de los DAC's varíen siempre dentro del rango dinámico de funcionamiento del comparador: $\left[V_{refp}, V_{refn}\right]$.

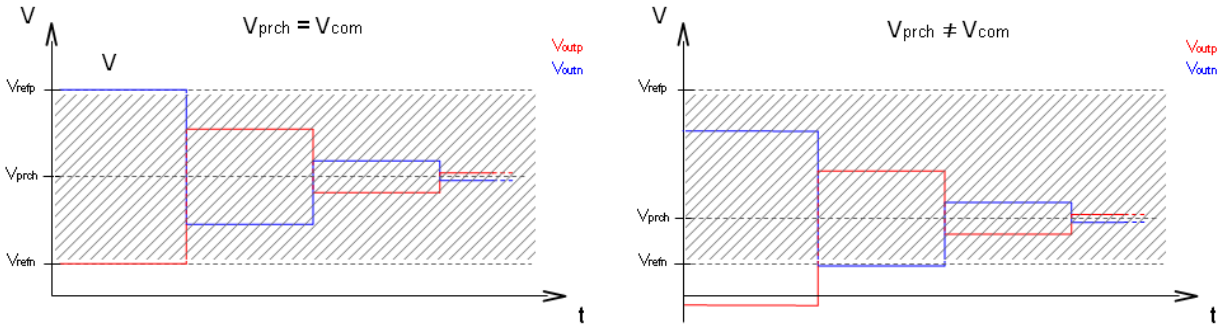


Figura 6.5: Tensión de salida de los DAC's con Vprch=Vcom y Vprch≠Vcom

Una posible solución para generar la V_{com} de precarga, es utilizar un *buffer* especialmente diseñado para este fin. Aunque parece la solución más lógica, conlleva asociados algunos problemas importantes, principalmente de área y consumo. La tensión de precarga requiere de una estabilidad elevada, para que el muestreo y la posterior conversión de la señal puedan llevarse a cabo de forma satisfactoria, ello hace necesario extremar las especificaciones para el *buffer*, y por consiguiente el área y el consumo requeridos para este circuito aumentan considerablemente.

Con el fin de no tener que implementar una referencia exclusiva para generar la V_{com}, y evitar de esta forma los problemas que ello conlleva, se ha optado por implementar un sistema de precarga y muestreo basado en *switches*, que permite generar una V_{com} muy estable a partir de las tensiones de referencia disponibles (V_{refp}, V_{refn}).

Como se puede observar en la figura, con el nuevo sistema se mantienen los *switches* de precarga de los DAC's, aunque ahora conectados a las referencias V_{refp} y V_{refn}, y se añade un nuevo *switch* que conecta entre si las salidas de ambos DAC's.

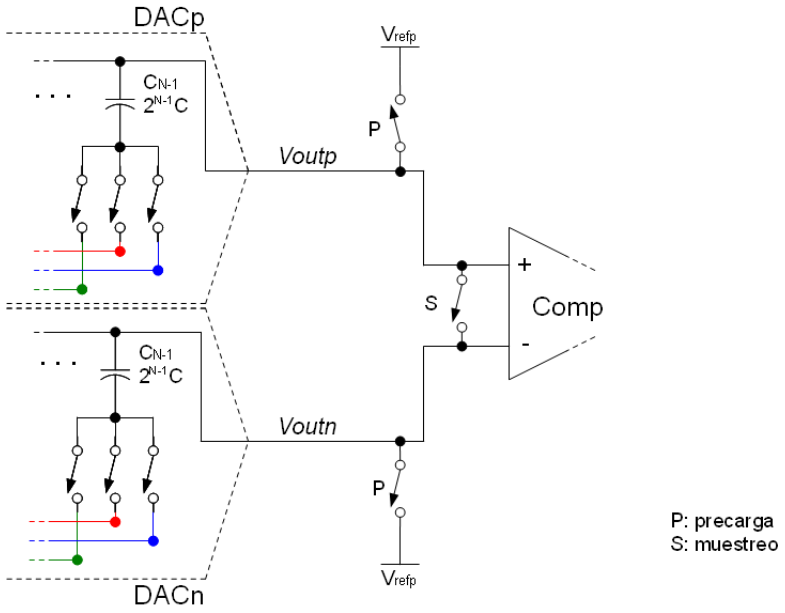


Figura 6.6: Detalle de los switches de precarga y muestreo

Para generar la V_{com} se utiliza la siguiente secuencia de apagado-encendido de los *switches*:

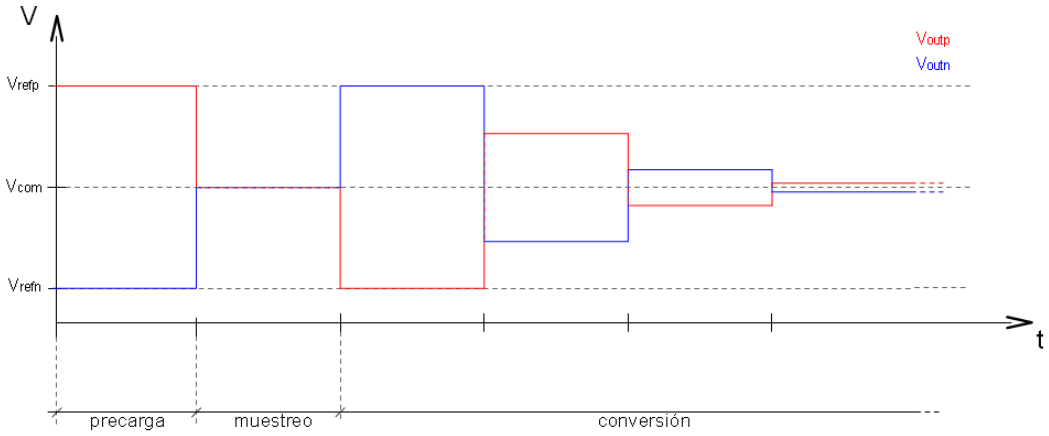


Figura 6.7: Secuencia del sistema de precarga y muestreo

1. Fase de precarga: los *switches* de precarga se encuentran activos, mientras que el de muestreo está abierto. En esta configuración, las tensiones en los nodos de salida de ambos DAC's se fijan a V_{refp} y V_{refn}, respectivamente.

2. Fase de muestreo: en esta fase se invierte el estado de los *switches*, quedando los *switches* de precarga abiertos y el *switch* de muestreo cerrado. Por consiguiente, las salidas de ambos DAC's

quedan cortocircuitadas y aparece en ellas la tensión V_{com}. Recordar que: $V_{com} = (V_{refp} - V_{refn})/2$

3. Fase de conversión: durante esta fase, tanto los *switches* de precarga como el de muestreo se mantienen abiertos, dejando que los DAC's trabajen normalmente mientras se ejecuta el algoritmo de búsqueda SAR.

6.3 DESCRIPCIÓN FUNCIONAL. *TIMING*

La topología de SAR ADC aquí presentada consta de los tres bloques principales en los que se basa cualquier convertidor SAR: lógica de control, DAC y comparador. Partiendo de ellos, se ha diseñado una topología particular de SAR ADC que se ajusta, de la mejor forma posible, a los requerimientos establecidos para este proyecto: arquitectura SAR *fully-differential* de 12 bits, área reducida y bajo consumo.

Los DAC's constituyen el corazón de la topología. Son de tipo capacitivo, lo que permite un consumo muy ajustado, además incluyen un sistema de precarga y muestreo, que hace posible el prescindir del típico circuito de T&H, que aumentaría el área del diseño. Por otro lado, para poder reducir aún más el área de este bloque, se va a implementar un sistema de calibración, mediante el cual es factible trabajar con un *array* de capacidades de dimensiones muy reducidas sin preocuparse por los problemas de *matching*, pues la calibración se encarga de corregirlos. Todos los detalles del sistema de calibración y la estructura final de DAC se describen en capítulos posteriores.

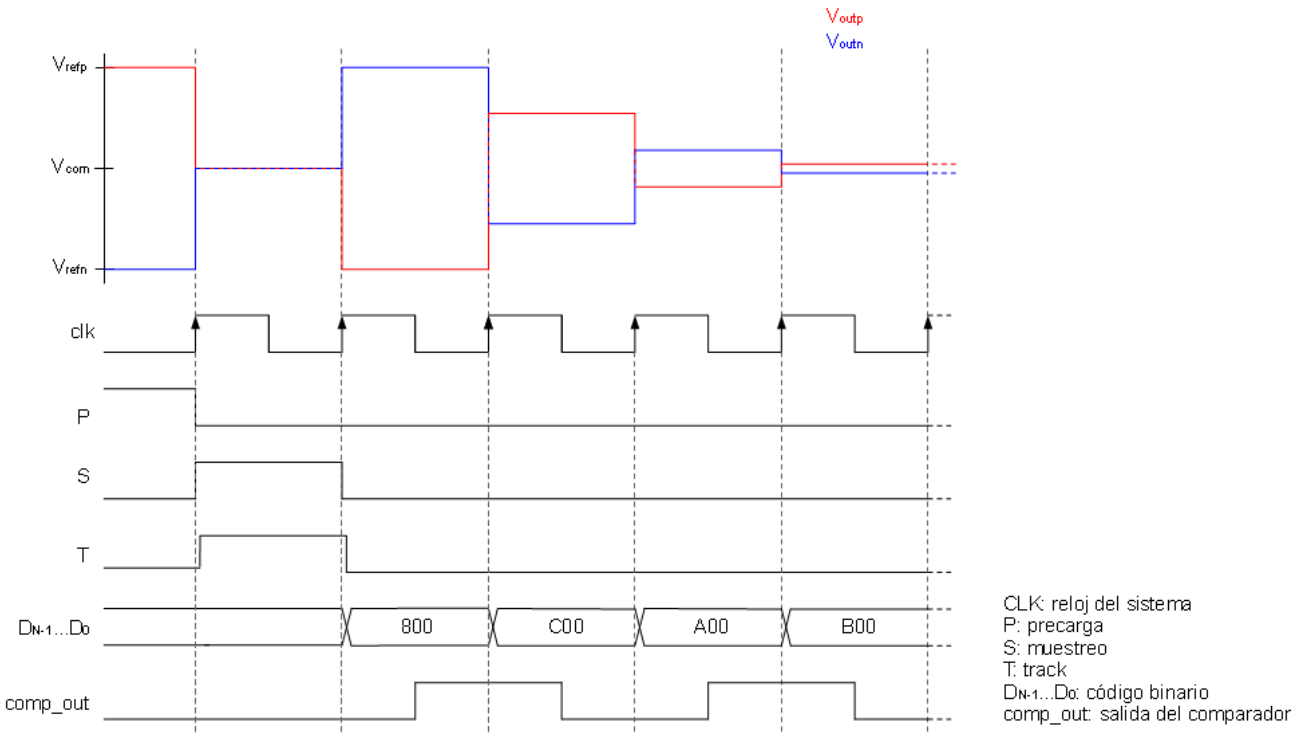


Figura 6.8: Cronograma de funcionamiento del SAR ADC fully-differential

En cuanto a la lógica de control, simplemente destacar que se basa en una máquina de estados, mediante la cual se va a ejecutar el algoritmo de búsqueda SAR junto con todas las operaciones

necesarias para la calibración del DAC. En siguientes capítulos se detallará su funcionamiento y las características de su implementación.

Y finalmente, el comparador, el cual requiere de una elevada precisión para comparar las señales provenientes de los DAC's, por lo que va a ser necesario implementar una topología de comparador que incorpore cancelación de *offse*t. Además, en cuanto al consumo, el comparador es el bloque más crítico, así que se necesita optimizar el compromiso velocidad-consumo para alcanzar un *througput* de 10kSPS y obtener un consumo reducido. En el capítulo décimo se describen todos los detalles acerca de este bloque.

Para finalizar con la descripción de la topología del SAR ADC, en la figura 6.8 se presenta un cronograma en donde se aprecia el *timing* de todas las señales que intervienen durante el proceso de conversión, y se resume la secuencia de funcionamiento del convertidor.

CAPÍTULO 7

Sistema de Calibración

7.1 INTRODUCCIÓN

Las técnicas de conversión analógico-digital de aproximaciones sucesivas requieren del uso de componentes con un *matching* elevado, para conseguir altas prestaciones en cuanto a resolución y linealidad. Concretamente, el bloque crítico que determina la resolución y las características de linealidad de un SAR ADC es el DAC. Se puede afirmar que la resolución y la linealidad que se obtienen para el DAC se corresponden directamente con las del propio ADC, por lo que el mayor esfuerzo, en este sentido, va ha concentrarse en el diseño del DAC.

Para este proyecto, se ha optado por implementar un DAC de 12 bits de tipo capacitivo, atendiendo a las ventajas que supone el uso de este tipo de DAC's en una topología SAR ADC, las cuales fueron expuestas de forma detallada en el capitulo cinco de este documento.

Los DAC's de tipo capacitivo se basan en *arrays* de capacidades binarias, es decir, *arrays* formados por capacidades con valores múltiplos de potencias de dos: $2^0 \cdot C$, $2^1 \cdot C$, $2^2 \cdot C$ … La función de transferencia de estos DAC's viene determinada directamente por los ratios entre las capacidades del *array*. De esta forma, si por cualquier motivo el ratio entre capacidades es modificado, la función de transferencia quedará alterada, apareciendo los denominados errores de linealidad.

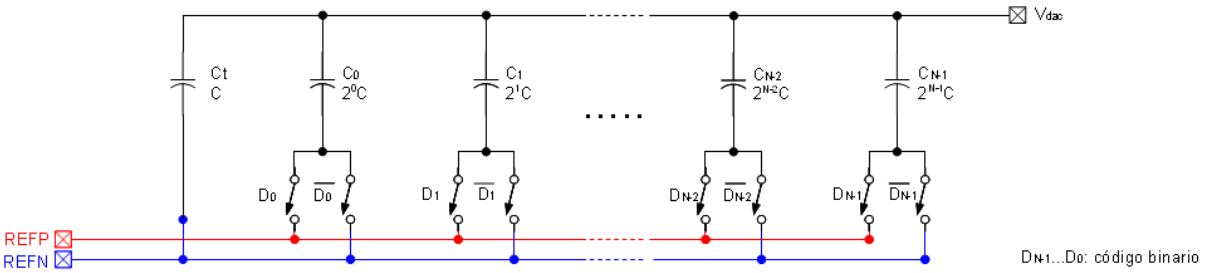
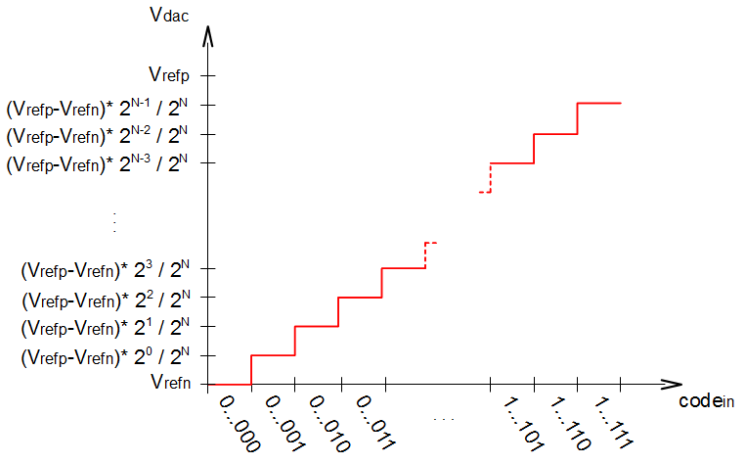


Figura 7.1: DAC capacitivo de N bits



Función de transferencia del DAC ideal (con un *matching* entre capacidades perfecto):

$$V_{dac,ideal} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \sum_{k=0}^{N-1} D_k \cdot 2^k$$

Figura 7.2: Fdt de un DAC ideal de N bits

Por defectos del proceso de fabricación, los ratios entre las capacidades del *array* no van a ser los perfectos, apareciendo de esta forma lo que se conoce como errores de *matching*. Esto errores se traducen en una variación del valor de las capacidades, de tal manera que:

$$C'_k = 2^k \cdot C \cdot (1+\varepsilon_k) \qquad ,k = N-1,...,0 \tag{7.1}$$

$$C'_t = 2^0 \cdot C \cdot (1+\varepsilon_t) \tag{7.2}$$

donde los ε_i representan el error del ratio entre capacidades.

Desarrollando de nuevo la función de transferencia del DAC, considerando ahora los nuevos valores de las capacidades, se obtiene una *fdt* sustancialmente modificada con respecto a la ideal, apareciendo en ella errores de ganancia y linealidad:

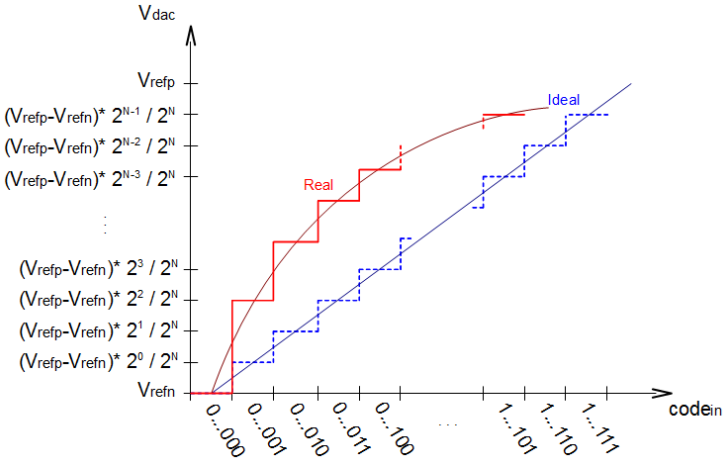


Figura 7.3: Fdt de un DAC real de N bits

Función de transferencia del DAC real (con errores de *matching* entre capacidades):

$$V_{dac,real} = \frac{(V_{refp} - V_{refn})}{2^N + \sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k + 2^0 \cdot \varepsilon_t} \cdot \sum_{k=0}^{N-1} D_k \cdot 2^k \cdot (1+\varepsilon_k)$$

De la expresión anterior se deduce que, el error de ganancia del DAC es generado por la suma de los errores de *matching* de todas las capacidades (incluida la capacidad C_t), lo que se corresponde al término: $\sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k + 2^0 \cdot \varepsilon_t$ de la función de transferencia. Definamos ese término como:

$$\varepsilon_{gain} = \sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k + 2^0 \cdot \varepsilon_t \qquad \textbf{(7.3)}$$

Por otro lado, los errores de linealidad vienen determinados por la contribución de los errores de *matching* de cada una de las capacidades sobre la tensión de salida (V_{dac}). Estas contribuciones (tensión de error) son de la forma:

$$V_{error,k} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k \qquad \textbf{(7.4)}$$

con lo que, la tensión de error total cometida para un cierto código $D_{N-1}...D_0$ será:

$$V_{error} = \sum_{k=0}^{N-1} D_k \cdot V_{error,k} \qquad \textbf{(7.5)}$$

Teniendo en cuenta estas consideraciones, la función de transferencia del DAC real puede ser reescrita en función de la *fdt* del DAC ideal ($V_{dac,ideal}$), del termino de error de ganancia (ε_{gain}) y de la tensión de error (V_{error}).

$$V_{dac,real} = \left(\frac{2^N}{2^N + \varepsilon_{gain}} \right) \cdot V_{dac,ideal} + V_{error} \quad \textbf{(7.6)}$$

Se concluye de esta manera que, los DAC's capacitivos son extremadamente sensibles al *matching* entre sus capacidades, y en consecuencia a los defectos del proceso de fabricación. Esto constituye uno de los principales inconvenientes de trabajar con este tipo de DAC's.

El valor de los errores de *matching* (ε_k) de cada capacidad, depende directamente del *matching* entre capacidades alcanzado con el *layout* del DAC. Habitualmente, para conseguir un *layout* con unas buenas características de *matching*, y que las ε_k de cada capacidad sean lo suficientemente pequeñas como para que no afecten de forma apreciable a la función de transferencia del DAC, se requiere de un estilo de *layout* muy cuidado, lo cual deriva en un área elevada. Por otro lado, a medida que la resolución del DAC aumenta, se hace más difícil el conseguir los requerimientos de *matching* necesarios para que el DAC consiga una buena linealidad. Todo esto conlleva a que, cuando se precise de una resolución elevada (a partir de 12/13 bits) o de un área del DAC muy reducida, sea inevitable la utilización de alguna técnica de compensación o calibración para corregir los problemas del DAC derivados de un *matching* insuficiente, pues el *matching* requerido para el *layout* va a ser muy difícil de alcanzar.

7.2 PROCESO DE AUTOCALIBRACIÓN

Para el diseño del sistema de calibración se asume que todas las capacidades del DAC presentan errores de *matching*, y que la suma de los errores de *matching* de todas ellas (incluida la capacidad C_t) es cero. Atendiendo a esta consideración, el término de error de ganancia (ε_{gain}) definido anteriormente se anula, con lo que la función de transferencia del DAC solo va a verse afectada por errores de linealidad.

$$\sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k + 2^0 \cdot \varepsilon_t = 0 \quad \rightarrow \quad \boxed{\varepsilon_{gain} = 0} \tag{7.7}$$

La técnica de calibración que aquí se presenta, se basa en el uso de una capacidad de calibración (C_{cal}) junto a un segundo DAC (DAC de calibración ó calDAC), a partir de los cuales se pretende inyectar o extraer carga del DAC principal (DAC de conversión ó convDAC) con la finalidad de corregir los errores de linealidad generados por el *matching* insuficiente de sus capacidades.

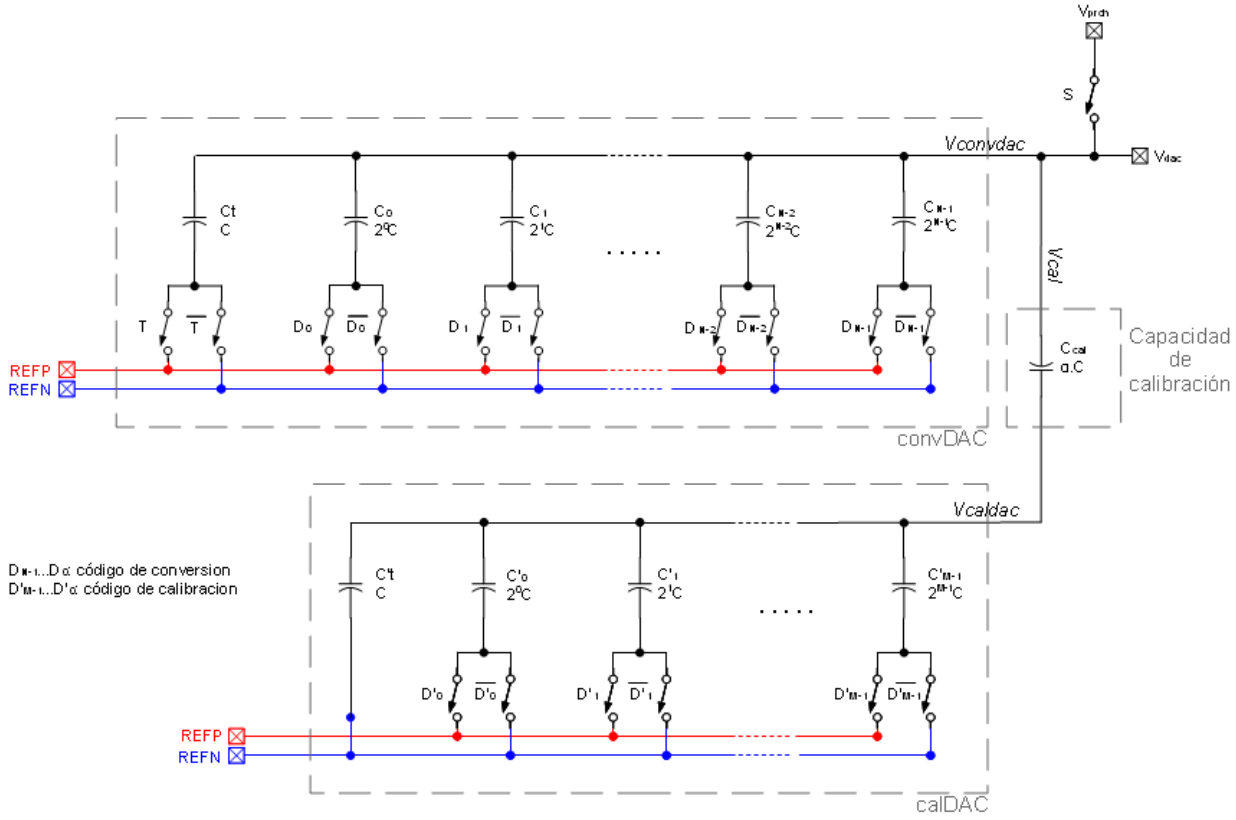


Figura 7.4: Arquitectura DAC de conversión + DAC de calibración

El principio de funcionamiento de esta estructura es sencillo. Gracias a la capacidad de calibración C_{cal}, es posible compensar la tensión de error (V_{error}) que el convDAC añade al nodo de salida (V_{dac}) debido el problema de *matching* de sus capacidades. Para ello, simplemente basta con conectar la C_{cal} a una tensión V_{caldac} adecuada para generar en el nodo de salida del DAC (V_{dac}) una tensión de calibración (V_{cal}) de valor igual y signo contrario a la tensión de error del convDAC y de esta forma neutralizar su efecto y corregir el problema de linealidad.

Véase el funcionamiento a partir de las ecuaciones del DAC:

$$V_{dac} = V_{convdac} + V_{cal} \tag{7.8}$$

$$V_{convdac} = \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot \left(\sum_{k=0}^{N-1} D_k \cdot 2^k + \sum_{k=0}^{N-1} D_k \cdot 2^k \cdot \varepsilon_k \right) = V_{dac,ideal} + V_{error} \tag{7.9}$$

$$V_{cal} = \frac{C_{cal}}{2^N \cdot C + C_{cal}} \cdot V_{caldac} = \frac{\alpha}{2^N + \alpha} \cdot V_{caldac} \tag{7.10}$$

$$V_{caldac} = \frac{(V_{refp} - V_{refn})}{2^M + \alpha} \cdot \sum_{k=0}^{M-1} D'_k \cdot 2^k \tag{7.11}$$

$\rightarrow$ Si para el código de conversión aplicado sobre el convDAC: $D_{N-1}...D_0$ se tiene una tensión de error V_{error}. Si aplico sobre el calDAC un código de calibración: $D'_{M-1}...D'_0$ tal que genere una $V_{cal} = -V_{error}$, entonces:

$$V_{dac} = V_{convdac} + V_{cal} = (V_{dac,ideal} + V_{error}) - V_{error} = V_{dac,ideal} \tag{7.12}$$

se ha conseguido compensar el error de linealidad.

Notar, que al añadir la nueva capacidad de calibración C_{cal}, se ve modificada la capacidad total del convDAC, lo cual se traduce en una atenuación adicional sobre la (V_{dac}), es decir, un cambio en la pendiente de la *fdt* del DAC o lo que es lo mismo, un error de ganancia. Pero este error no va a suponer ningún problema, pues la atenuación es siempre la misma, tanto para la señal que se muestrea como para el código convertido, con lo que al final su efecto se compensa y no influye sobre el resultado.

Para poder aplicar esta estrategia de corrección, se requiere de un sistema de calibración que sea capaz de: por una parte, medir los errores de *matching* de cada capacidad ($V_{error,k}$) y a partir de ellos calcular los códigos de calibración ($D'_{M-1}...D'_0$) requeridos para su compensación, y por otra, combinar los códigos de calibración hallados, para poder corregir la tensión de error (V_{error}) generada para cada código de conversión ($D_{N-1}...D_0$) que se aplica al convDAC. Estas tareas, definen las dos fases del funcionamiento del sistema de calibración: fase de calibración y fase de conversión.

7.2.1 Fase de calibración

La fase de calibración se ejecuta una sola vez, y en ella se buscan los códigos de calibración requeridos para compensar los errores de *matching* de cada una de las capacidades del convDAC ($V_{error,k}$). El proceso se repite N veces, una vez por capacidad a calibrar, empezando por la búsqueda del código de calibración para la capacidad de mayor peso (C_{N-1}) y terminando por la de menor peso (C_0).

La búsqueda de los valores de calibración se basa en un proceso de conversión analógico-digital SAR aplicado sobre el calDAC, en donde, la tensión a convertir se corresponde con la tensión residual ($V_{res,k}$) que aparece en la salida del DAC (V_{dac}) al realizar una cierta operación de muestreo. Al concluir la búsqueda SAR, se obtiene el código digital del calDAC que representa a la $V_{res,k}$. Esta tensión residual se encuentra directamente relacionada con la tensión de error ($V_{error,k}$) con lo que una vez obtenido el código digital que la representa, es posible obtener, a partir de éste y de los códigos de calibración de las capacidades de peso superior (calculados con anterioridad), el código de calibración buscado.

Veamos a continuación, cual es el procedimiento a seguir para encontrar el código de calibración correspondiente a la capacidad de mayor peso C_{N-1}, y el de otra capacidad cualquiera C_j , $j \in [N-2,...,0]$.

✓ *Código de calibración para la capacidad* C_{N-1}*:*

PASO 1: Muestreo

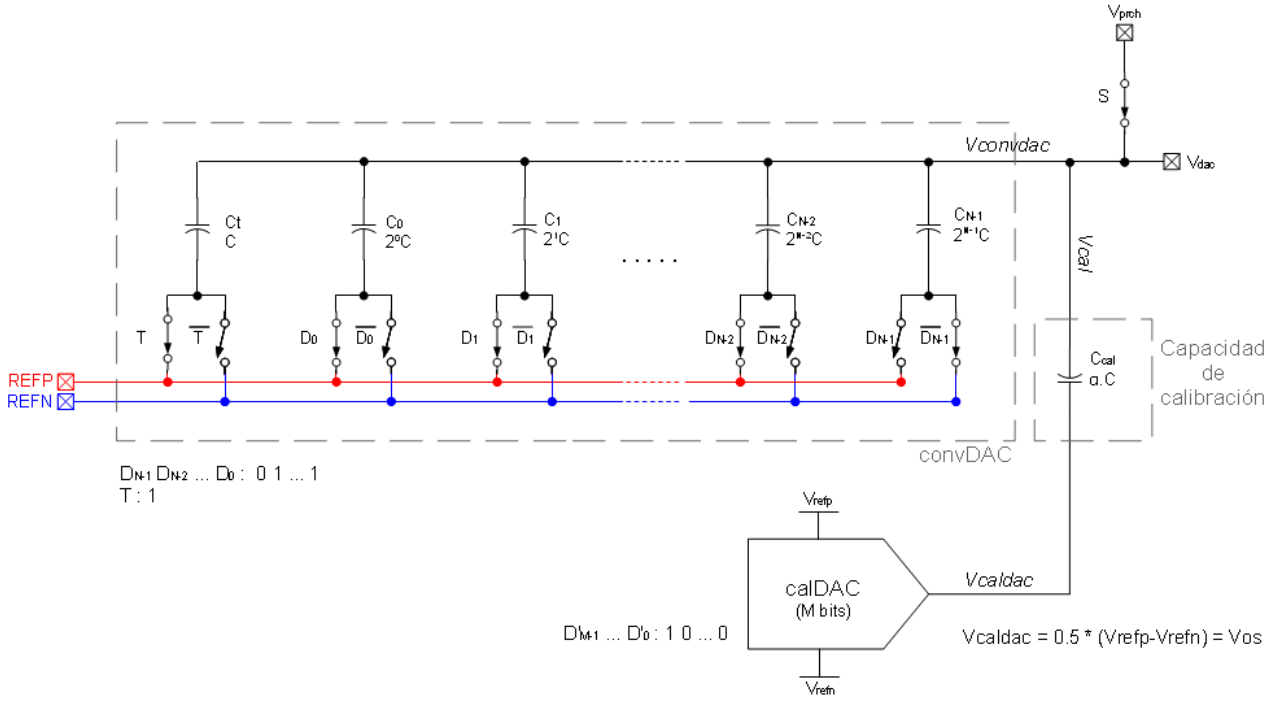


Figura 7.5: Configuración del DAC para el muestreo de $V_{res,N-1}$

Carga total en el DAC durante el muestreo:

$$Q_M = (V_{prch} - V_{refp}) \cdot \left[C_t + \sum_{k=0}^{N-2} C_k \right] + (V_{prch} - V_{refn}) \cdot C_{N-1} + (V_{prch} - V_{OS}) \cdot C_{cal} \qquad \textbf{(7.13)}$$

PASO 2: Búsqueda SAR

Carga total en el DAC durante la búsqueda SAR:

$$Q_B = (V_{dac} - V_{refn}) \cdot \left[C_t + \sum_{k=0}^{N-2} C_k \right] + (V_{dac} - V_{refp}) \cdot C_{N-1} + (V_{dac} - V_{res,N-1}) \cdot C_{cal} \qquad \textbf{(7.14)}$$

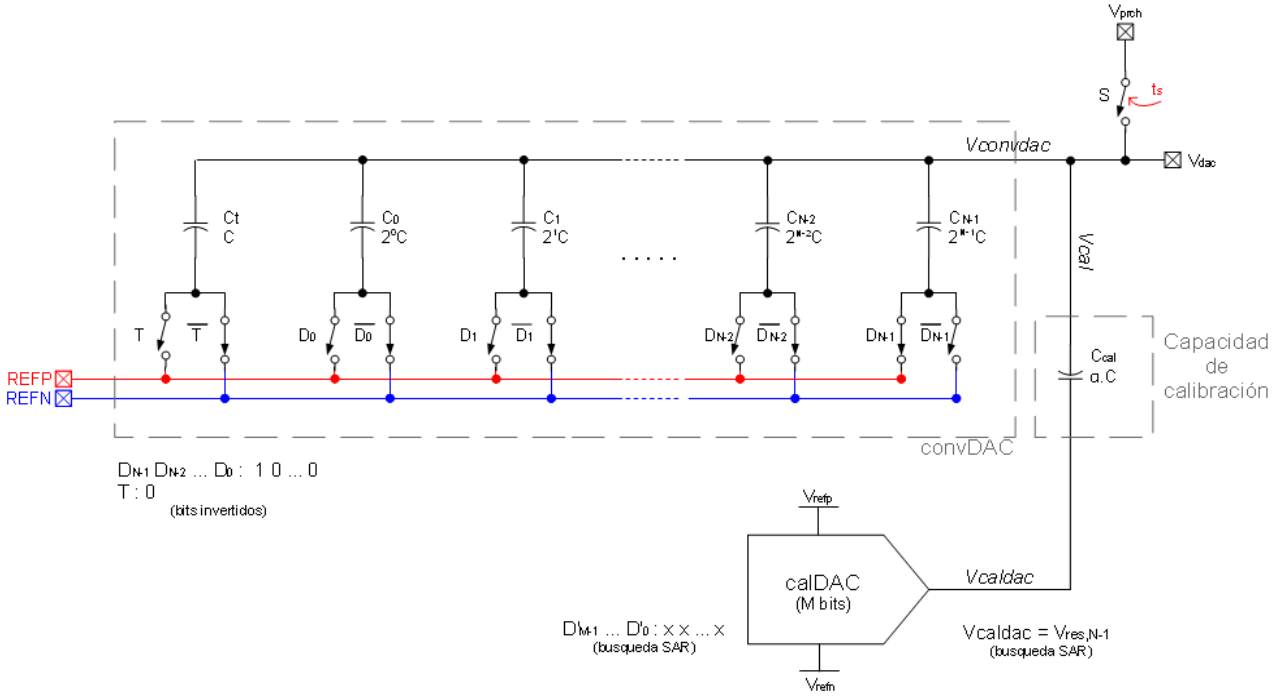


Figura 7.6: Configuración del DAC para la búsqueda de $V_{res,N-1}$

Según el principio de redistribución de cargas, la carga total almacenada en el DAC durante el muestreo y durante la búsqueda SAR deben ser iguales, con lo que:

$$Q_M = Q_B \tag{7.15}$$

$$V_{dac}\cdot\left[C_t+\sum_{k=0}^{N-2}C_k+C_{N-1}+C_{cal}\right]=V_{prch}\cdot\left[C_t+\sum_{k=0}^{N-2}C_k+C_{N-1}+C_{cal}\right]+V_{refp}\cdot\left[-C_t-\sum_{k=0}^{N-2}C_k+C_{N-1}\right]+$$
$$-V_{refn}\cdot\left[-C_t-\sum_{k=0}^{N-2}C_k+C_{N-1}\right]+C_{cal}\cdot\ V_{res,N-1}-V_{OS} \tag{7.16}$$

donde:

$$\rightarrow\left[C_t+\sum_{k=0}^{N-2}C_k+C_{N-1}+C_{cal}\right]=2^N\cdot C+C\cdot\varepsilon_t+\sum_{k=0}^{N-1}C_k\cdot\varepsilon_k+\alpha\cdot C=2^N\cdot C+0+\alpha\cdot C=$$
$$=2^N\cdot C+\alpha\cdot C$$

$$\rightarrow\left[-C_t-\sum_{k=0}^{N-2}C_k+C_{N-1}\right]=-C_t-\sum_{k=0}^{N-2}2^k\cdot C+2^{N-1}\cdot C-C_t\cdot\varepsilon_t-\sum_{k=0}^{N-2}2^k\cdot C\cdot\varepsilon_k+2^{N-1}\cdot C\cdot\varepsilon_{N-1}=$$
$$=-2^{N-1}\cdot C+2^{N-1}\cdot C+2^{N-1}\cdot C\cdot\varepsilon_{N-1}+2^{N-1}\cdot C\cdot\varepsilon_{N-1}=$$
$$=2\cdot 2^{N-1}\cdot C\cdot\varepsilon_{N-1}=2^N\cdot C\cdot\varepsilon_{N-1}$$

despejando la V_{dac} de la expresión anterior y simplificando, se tiene que:

$$V_{dac} = \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^N \cdot \varepsilon_{N-1} + V_{prch} + \frac{\alpha}{2^N + \alpha} \cdot (V_{res,N-1} - V_{OS}) \quad \textbf{(7.17)}$$

Al finalizar el proceso SAR, la V_{dac} será igual a la V_{prch}, con lo que el valor de la tensión residual $V_{res,N-1}$ encontrada por el calDAC valdrá:

$$V_{dac} = V_{prch} = \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^N \cdot \varepsilon_{N-1} + V_{prch} + \frac{\alpha}{2^N + \alpha} \cdot (V_{res,N-1} - V_{OS}) \quad \textbf{(7.18)}$$

$$\downarrow$$

$$\frac{\alpha}{2^N + \alpha} \cdot (V_{res,N-1} - V_{OS}) = -\frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^N \cdot \varepsilon_{N-1} \quad \textbf{(7.19)}$$

$$\downarrow$$

$$(V_{res,N-1} - V_{OS}) = -\frac{2^N + \alpha}{\alpha} \cdot \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^N \cdot \varepsilon_{N-1} \quad \textbf{(7.20)}$$

Igualando la tensión residual encontrada, con la tensión de error generada por el error de *matching* del C_{N-1}, se obtiene:

$$\begin{cases} V_{error,N-1} = -\dfrac{2^N + \alpha}{\alpha} \cdot \dfrac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^{N-1} \cdot \varepsilon_{N-1} & \textbf{(7.21)} \\ (V_{res,N-1} - V_{OS}) = -\dfrac{2^N + \alpha}{\alpha} \cdot \dfrac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2^N \cdot \varepsilon_{N-1} & \textbf{(7.20)} \end{cases}$$

$$\downarrow$$

$$V_{error,N-1} = -\frac{(V_{res,N-1} - V_{OS})}{2} \quad \textbf{(7.22)}$$

*NOTA: la V_{OS} de la ecuación representa la tensión de *offset* fijada en el calDAC durante el muestreo, con la finalidad de que el valor de la $V_{res,N-1}$ pueda tomar valores positivos y negativos y poder compensar de esta forma el exceso o defecto de cargas en el convDAC generado por las ε_{N-1} positivas y negativas.

Por lo tanto, la tensión de calibración ($V_{cal,N-1}$) que se requiere aplicar sobre la salida (V_{dac}) para compensar el error de C_{N-1}, será:

$$V_{cal,N-1} = -V_{error,N-1} = \frac{(V_{res,N-1} - V_{OS})}{2} \qquad \textbf{(7.23)}$$

donde: $(V_{res,N-1} - V_{OS})$ se corresponde con la tensión residual hallada por el calDAC, al aplicar sobre este el algoritmo de búsqueda SAR.

✓ *Código de calibración para la capacidad C_j:*

PASO 1: Muestreo

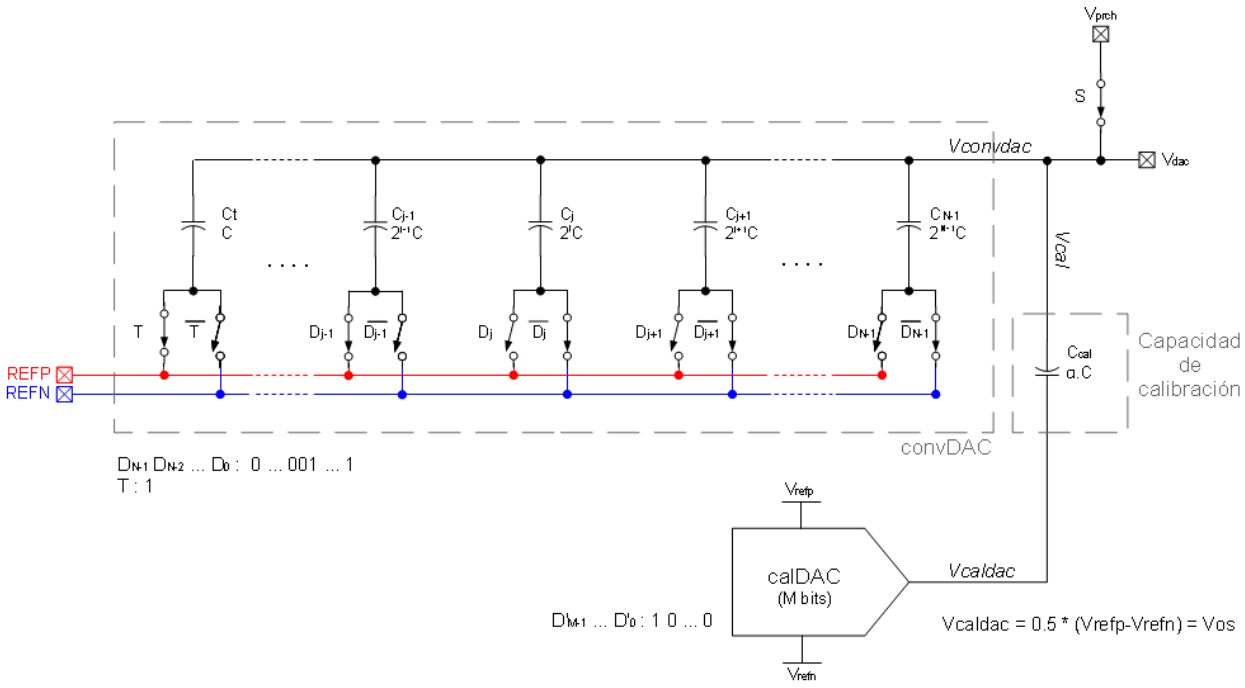


Figura 7.7: Configuración del DAC para el muestreo de $V_{res,j}$

Carga total en el DAC durante el muestreo:

$$Q_M = (V_{prch} - V_{refp}) \cdot \left[C_t + \sum_{k=0}^{j-1} C_k \right] + (V_{prch} - V_{refn}) \cdot \sum_{k=j}^{N-1} C_k + (V_{prch} - V_{OS}) \cdot C_{cal} \qquad \textbf{(7.24)}$$

PASO 2: Búsqueda SAR

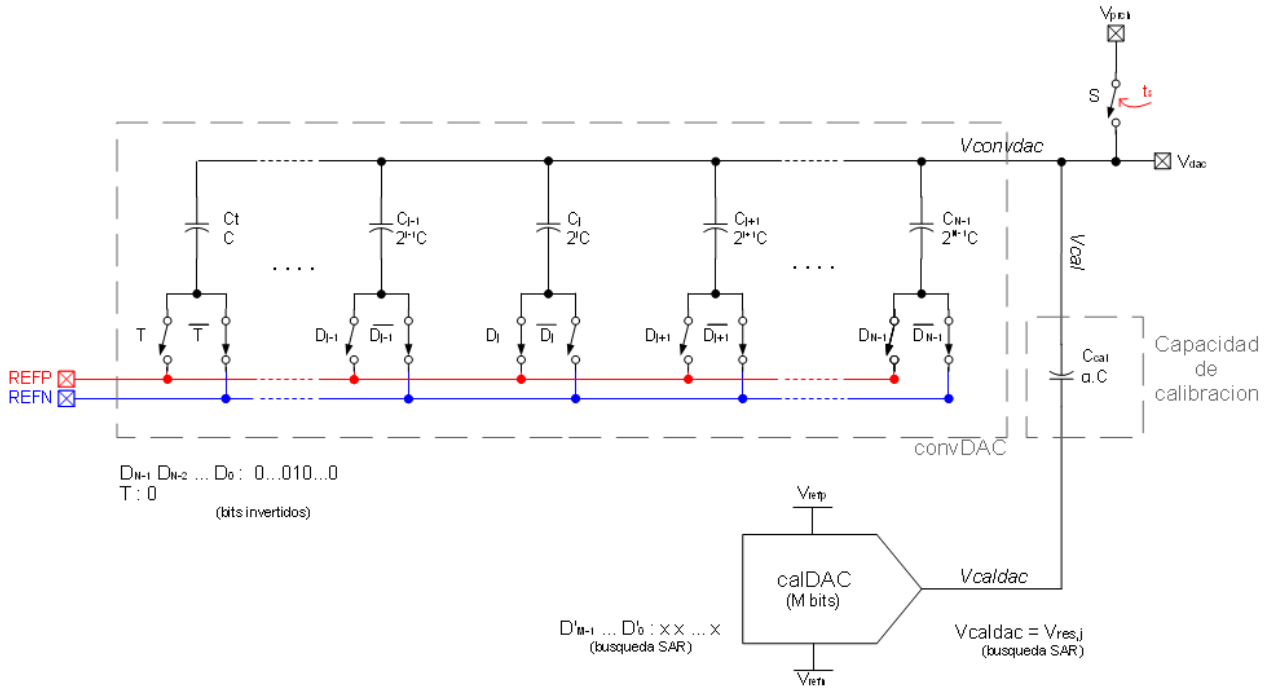


Figura 7.8: Configuración del DAC para la búsqueda de $V_{res,j}$

Carga total en el DAC durante la búsqueda SAR:

$$Q_B = (V_{dac} - V_{refn}) \cdot \left[C_t + \sum_{k=0}^{j-1} C_k + \sum_{k=j+1}^{N-1} C_k \right] + (V_{dac} - V_{refp}) \cdot C_j + (V_{dac} - V_{res,j}) \cdot C_{cal} \quad \textbf{(7.25)}$$

Según el principio de redistribución de cargas, la carga total almacenada en el DAC durante el muestreo y durante la búsqueda SAR deben ser iguales, con lo que:

$$Q_M = Q_B \quad \textbf{(7.15)}$$

$$\begin{aligned} V_{dac} \cdot \left[C_t + \sum_{k=0}^{j-1} C_k + C_j + \sum_{k=j+1}^{N-1} C_k + C_{cal} \right] = V_{prch} \cdot \left[C_t + \sum_{k=0}^{j-1} C_k + C_j + \sum_{k=j+1}^{N-1} C_k + C_{cal} \right] + \\ + V_{refp} \cdot \left[-C_t - \sum_{k=0}^{j-1} C_k + C_j \right] + C_{cal} \cdot V_{res,j} - V_{OS} + \\ - V_{refn} \cdot \left[-C_t - \sum_{k=0}^{j-1} C_k + C_j + \sum_{k=j+1}^{N-1} C_k - \sum_{k=j+1}^{N-1} C_k \right] \end{aligned}$$

(7.26)

donde:

$$\rightarrow \left[C_t + \sum_{k=0}^{j-1} C_k + C_j + \sum_{k=j+1}^{N-1} C_k + C_{cal} \right] = 2^N \cdot C + C \cdot \varepsilon_t + \sum_{k=0}^{N-1} C_k \cdot \varepsilon_k + \alpha \cdot C = 2^N \cdot C + 0 + \alpha \cdot C = 2^N \cdot C + \alpha \cdot C$$

$$\rightarrow \left[-C_t - \sum_{k=0}^{j-1} C_k + C_j \right] = -C_t - \sum_{k=0}^{j-1} 2^k \cdot C + 2^j \cdot C - C_t \cdot \varepsilon_t - \sum_{k=0}^{j-1} 2^k \cdot C \cdot \varepsilon_k + 2^j \cdot C \cdot \varepsilon_j =$$

$$= -2^j \cdot C + 2^j \cdot C + 2 \cdot 2^j \cdot C \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} 2^k \cdot C \cdot \varepsilon_k =$$

$$= 2 \cdot 2^j \cdot C \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} 2^k \cdot C \cdot \varepsilon_k$$

despejando la V_{dac} de la expresión anterior y simplificando, se tiene que:

$$V_{dac} = \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2 \cdot 2^j \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} 2^k \cdot \varepsilon_k + V_{prch} + \frac{\alpha}{2^N + \alpha} \cdot (V_{res,j} - V_{OS}) \quad \textbf{(7.27)}$$

Al finalizar el proceso SAR, la V_{dac} será igual a la V_{prch}, con lo que el valor de la tensión residual $V_{res,j}$ encontrada por el calDAC valdrá:

$$V_{dac} = V_{prch} = \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2 \cdot 2^j \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} 2^k \cdot \varepsilon_k + V_{prch} + \frac{\alpha}{2^N + \alpha} \cdot (V_{res,j} - V_{OS}) \quad \textbf{(7.28)}$$

$$\downarrow$$

$$(V_{res,j} - V_{OS}) = -\frac{2^N + \alpha}{\alpha} \cdot \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot \left[2 \cdot 2^j \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} 2^k \cdot \varepsilon_k \right] \quad \textbf{(7.29)}$$

$$\downarrow$$

$$(V_{res,j} - V_{OS}) = -\frac{2^N + \alpha}{\alpha} \cdot \frac{(V_{refp} - V_{refn})}{2^N + \alpha} \cdot 2 \cdot 2^j \cdot \varepsilon_j + \sum_{k=j+1}^{N-1} V_{error,k} \quad \textbf{(7.30)}$$

Igualando la tensión residual encontrada, con la tensión de error generada por el error de *matching* del C_j, se obtiene:

$$\begin{cases} V_{error,j} = -\dfrac{2^N+\alpha}{\alpha}\cdot\dfrac{(V_{refp}-V_{refn})}{2^N+\alpha}\cdot 2^j\cdot\varepsilon_j & \textbf{(7.31)} \\ (V_{res,j}-V_{OS}) = -\dfrac{2^N+\alpha}{\alpha}\cdot\dfrac{(V_{refp}-V_{refn})}{2^N+\alpha}\cdot 2\cdot 2^j\cdot\varepsilon_j + \sum_{k=j+1}^{N-1} V_{error,k} & \textbf{(7.30)} \end{cases}$$

$$\downarrow$$

$$V_{error,j} = -\frac{1}{2}\cdot\left[(V_{res,j}-V_{OS}) + \sum_{k=j+1}^{N.1} V_{error,k}\right] \qquad \textbf{(7.32)}$$

Por lo tanto, la tensión de calibración ($V_{cal,j}$) que se requiere aplicar sobre la salida (V_{dac}) para compensar el error de *matching* de C_j, será:

$$\boxed{V_{cal,j} = -V_{error,j} = \frac{1}{2}\cdot\left[(V_{res,j}-V_{OS}) - \sum_{k=j+1}^{N-1} V_{cal,k}\right]} \qquad \textbf{(7.33)}$$

donde: $(V_{res,j}-V_{OS})$ es la tensión medida por el calDAC y $V_{cal,k}$ representa las tensiones de calibración de las capacidades de peso superior, las cuales han sido halladas con anterioridad.

Para finalizar con la descripción de la fase de calibración, en la figura 7.9 se muestra un diagrama de flujo que sintetiza la secuencia de operaciones realizadas para encontrar los valores de calibración de cada una de las N capacidades del convDAC.

7.2.2 Fase de conversión

La fase de conversión define el estado de funcionamiento normal del convertidor, es decir, el estado en el que el DAC realiza tareas de conversión. Durante esta fase, el sistema de calibración se encarga de calcular el código de calibración del calDAC, necesario para compensar el error total (V_{error}) generado en la salida del DAC al aplicar un cierto código $D_{N-1}...D_0$ en su entrada.

Para ello, se parte de los códigos de calibración hallados durante la fase previa de calibración, es decir, los códigos que al aplicarlos sobre el calDAC permiten corregir la tensión de error ($V_{error,k}$) generada por cada capacidad individualmente. Según lo estudiado al principio de este capítulo, la tensión de error total (V_{error}) que aparece a la salida del DAC cuando sobre éste se aplica un cierto código de conversión $D_{N-1}...D_0$, está formada por la contribución de las tensiones de error

($V_{error,k}$) generadas por cada una de las capacidades del convDAC que se encuentran activas, es decir:

$$V_{error} = \sum_{k=0}^{N-1} D_k \cdot V_{error,k} \qquad , donde \quad D_k = 0,1 \tag{7.34}$$

Por lo tanto, siguiendo esta misma definición, el código de calibración a aplicar sobre el calDAC para compensar la V_{error}, será el resultado de sumar todos los códigos de calibración que compensan las $V_{error,k}$ de cada una de las capacidades activas, por lo que:

$$V_{cal} = \sum_{k=0}^{N-1} D_k \cdot V_{cal,k} \tag{7.35}$$

en forma de códigos del calDAC, seria:

$$código(V_{cal}) = \sum_{k=0}^{N-1} D_k \cdot código(V_{cal,k}) \tag{7.36}$$

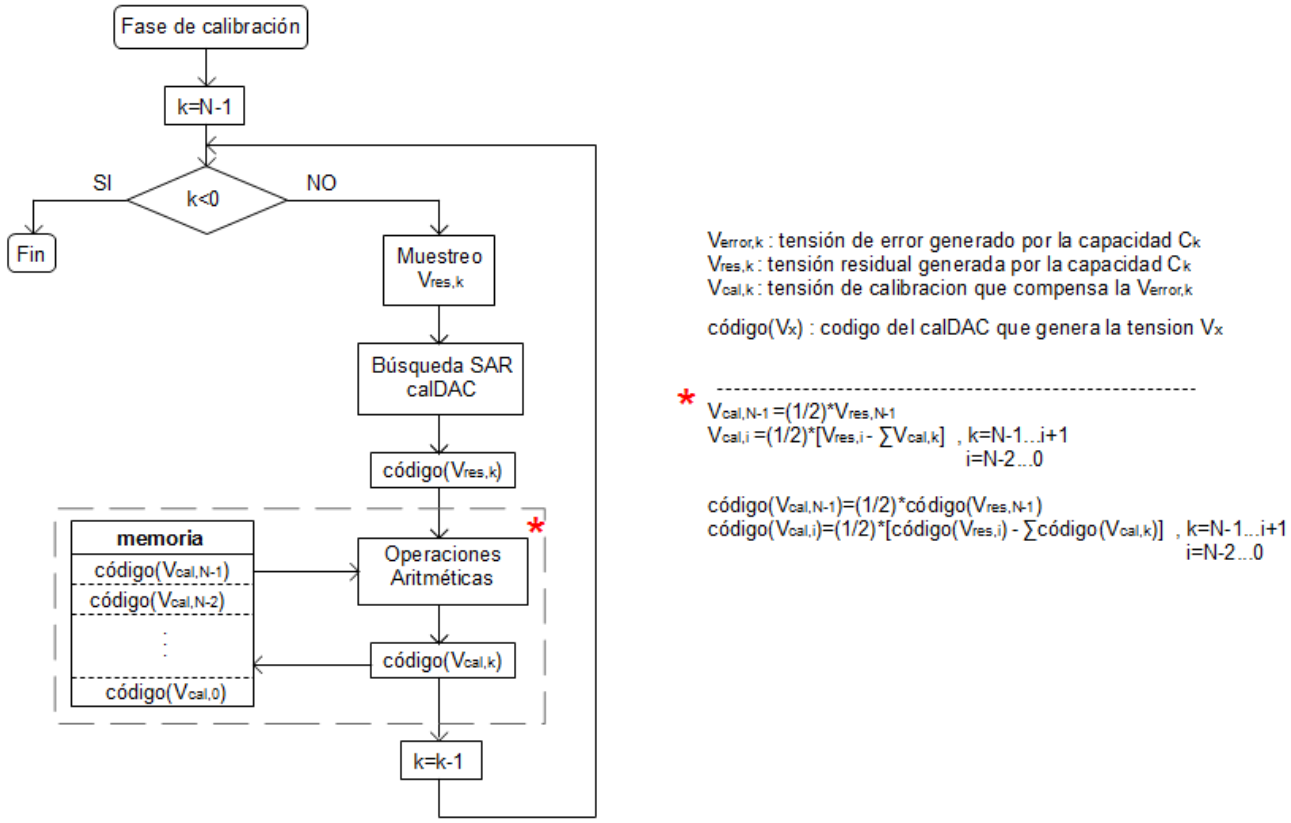


Figura 7.9: Diagrama de flujo de la fase de calibración

Véase el siguiente ejemplo que ilustra lo anterior. Consideremos un convDAC de 4 bits, para el cual, en un instante determinado el código digital a su entrada es: '1010'. Conociendo los códigos de calibración (*cal_bit3*, *cal_bit2*, *cal_bit1* y *cal_bit0*) que compensan las tensiones de error producidas por cada una de las capacidades, el código *cal_1010* a aplicar para compensar el error de linealidad cometido con el código 1010 se calculará como:

$$cal\_1010 = 1 \cdot cal\_bit3 + 0 \cdot cal\_bit2 + 1 \cdot cal\_bit1 + 0 \cdot cal\_bit0$$

En resumen, durante la fase de conversión el sistema de calibración se encargará simplemente de, leer el código digital a convertir en cada instante y sumar los códigos de calibración asociados a cada capacidad que se encuentre activa, para de esta forma, aplicando el código resultante sobre el calDAC, compensar el error de linealidad cometido. Conociendo que, los códigos a convertir se corresponderán con los códigos de búsqueda generados por el algoritmo SAR, se puede plantear una estrategia de cálculo, basada en un simple sumador y un acumulador, para obtener, de forma rápida y eficiente, el código de calibración necesario en cada iteración.

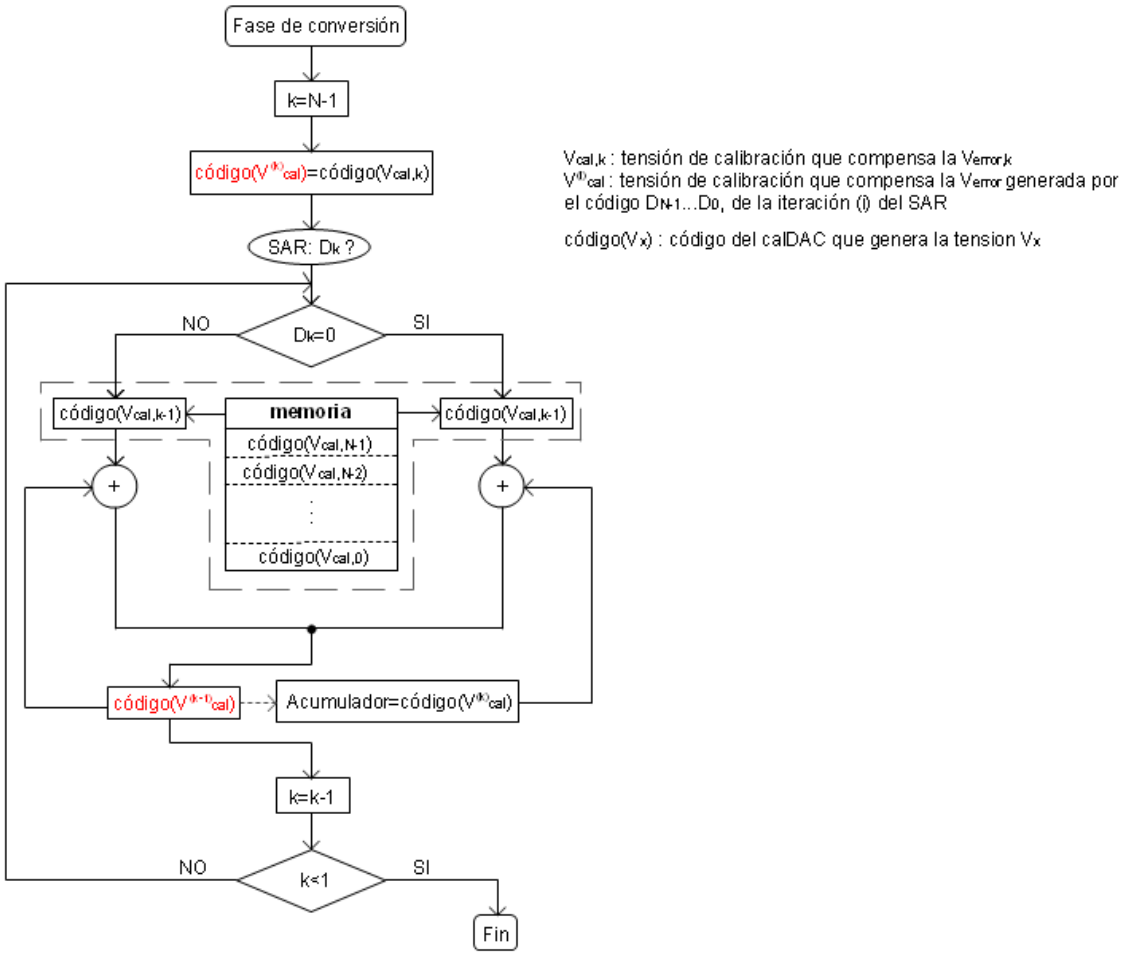


Figura 7.10: Diagrama de flujo de la fase de conversión

7.2.3 Calibración dinámica vs. Calibración estática

Dependiendo del momento y de la forma en la que la fase de calibración se lleva a cabo, se habla de una calibración de tipo estática o dinámica.

Se dice que la calibración es estática, cuando la fase de calibración se ejecuta una sola vez a lo largo de la vida del chip. La medida de los errores de *matching* del DAC y el posterior cálculo de los códigos de calibración se lleva a cabo durante la fase de testeo, en fabrica. A partir de ese momento, los códigos de calibración se almacenan en una memoria de sólo lectura y son utilizados para siempre en todas las conversiones que realice el circuito.

Este tipo de calibración tiene el inconveniente de no poder adaptarse a las posibles variaciones de *matching* que puede sufrir el circuito debido a: el deterioro por el paso del tiempo, la variabilidad de las condiciones de trabajo… lo cual puede provocar que, de forma muy puntual, los códigos de calibración que fueron medidos en fabrica no sean válidos y la calibración no sea satisfactoria. Por otro lado, la ventaja de una calibración estática se traduce en ahorro de área y consumo, pues al ejecutarse la fase de calibración en test, no es necesario incluir en el chip toda la lógica necesaria para realizar los cálculos de los códigos de calibración, basta con incluir una memoria de solo lectura que almacene los códigos una vez hayan sido calculados de forma externa.

En el polo opuesto a la calibración estática, se encuentra la calibración dinámica. Con la calibración dinámica se consigue que el circuito pueda recalibrarse en cualquier momento, de tal manera que siempre sea posible compensar los errores de *matching* bajo distintas condiciones de trabajo, distinto estado de deterioramiento… En este caso, no es posible prescindir de la lógica necesaria para el cálculo de los códigos de calibración, pues la fase de calibración va a ser ejecutada por el propio circuito, con lo que la complejidad del diseño aumenta sensiblemente y el área y consumo total se ven incrementados. La calibración dinámica normalmente funciona de la siguiente forma: durante el *power-up*, se ejecuta la fase de calibración, obteniéndose los códigos de calibración requeridos para las condiciones actuales en las que se encuentra el circuito, estos códigos son almacenados en memoria volátil y se utilizan durante la fase de conversión, hasta que se tiene un nuevo *power-up* y el proceso se vuelve a repetir.

En este proyecto se ha considerado el utilizar una calibración de tipo estática, de tal manera que toda la lógica para el cálculo de los códigos de calibración y la memoria donde se almacenan se han implementado en un bloque externo al del propio diseño, empleando para ello un modelo *Verilog*. La descripción *Verilog* utilizada se adjunta en el anexo C de este documento, con el nombre de *cal_memory.v*.

7.3 LÍMITES DE LA CALIBRACIÓN

En función de la magnitud de los errores de *matching* que vaya a presentar el convDAC, se requiere de un ajuste de la calibración determinado para poder compensarlos.

El sistema de calibración presentado permite su ajuste a través de dos parámetros: el valor de la capacidad de calibración (C_{cal}) y el número de bits del DAC de calibración (M). Variando estos dos parámetros es posible ajustar la precisión de la compensación y el rango de errores que es posible corregir.

Para el diseño de los parámetros del sistema de calibración hay que tener en cuenta varias cosas:

En primer lugar, hay que conocer los requerimientos de compensación del convDAC, es decir, saber cuánto de grandes serán los errores de linealidad que se van a cometer, y a partir de ellos definir el rango de valores de V_{cal} que el calDAC va a tener que ser capaz de aplicar. El rango de valores entre los que puede oscilar la V_{cal} viene fijado por la capacidad de calibración (C_{cal}):

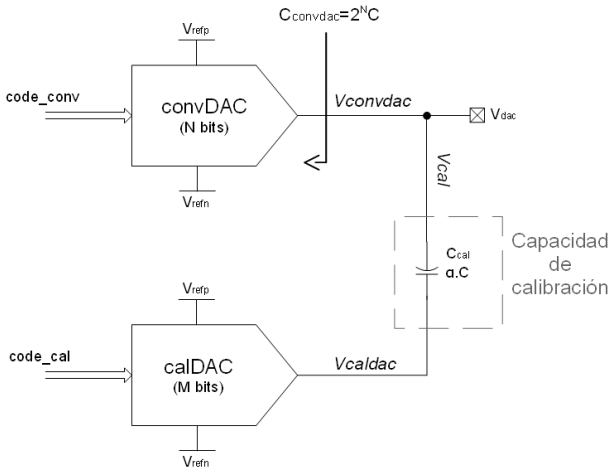


Figura 7.11: Diagrama de bloques del DAC con calibración

$$V_{cal} = V_{caldac} \cdot \left(\frac{C_{cal}}{C_{cal} + C_{convdac}} \right) \qquad \textbf{(7.37)}$$

$$V_{caldac} \in \left[V_{refn}, V_{refp} \right] = V_{OS} + \left[-\frac{(V_{refp} - V_{refn})}{2}, \frac{(V_{refp} - V_{refn})}{2} \right]$$

$$V_{cal} \in \frac{C_{cal}}{C_{cal} + C_{convdac}} \cdot \left\{ V_{OS} + \left[-\frac{(V_{refp} - V_{refn})}{2}, \frac{(V_{refp} - V_{refn})}{2} \right] \right\}$$

Suponiendo que se requiere de una $V_{cal}\,|_{MAX}$ para poder compensar los errores de linealidad máximos del convDAC, será necesario diseñar el sistema de calibración para cumplir que:

$$\left|V_{cal}\,|_{\max}\right| < \frac{(V_{refp} - V_{refn})}{2} \cdot \frac{C_{cal}}{C_{cal} + C_{convdac}} \qquad \textbf{(7.38)}$$

Por otro lado, también hay que tener en cuenta las propias imperfecciones del sistema de calibración, las cuales se van a traducir en errores durante el proceso de calibración. Es obvio, que el calDAC de M bits encargado de medir los errores de *matching* y de aplicar la V_{cal} de compensación, va a generar un error de cuantificación al medir las $V_{res,k}$ Además, como se ha visto en el apartado anterior, para obtener las $V_{cal,k}$ de calibración se requiere aplicar una serie de operaciones aritméticas sobre las $V_{res,k}$ medidas, con lo que en estas operaciones va a ser inevitable cometer errores de redondeo. Todas esas imperfecciones van a afectar al valor de calibración (V_{cal}) que finalmente se aplicará, haciendo que la compensación no sea exacta.

Si se hace un estudio del efecto de los errores de cuantificación y redondeo sobre las $V_{cal,k}$ aplicadas, se puede llegar a las siguientes conclusiones:

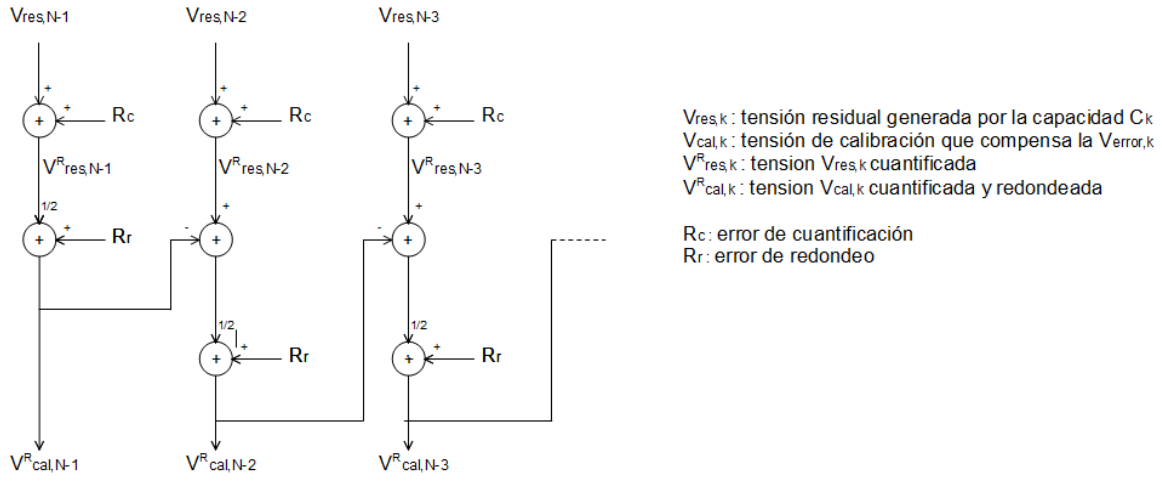


Figura 7.12: Diagrama de operaciones para el cálculo de los códigos de calibración

Para el cálculo de las tensiones de calibración ($V_{cal,k}$) se aplicaba la siguiente operación:

$$V_{cal,k} = \frac{1}{2} \cdot \left[V_{res,k} - \sum_{i=k+1}^{N-1} V_{cal,i} \right] \qquad \textbf{(7.39)}$$

Teniendo en cuenta que cada $V_{res,k}$ es afectada por la cuantificación del calDAC (R_c), y que al dividir por dos se comete un error de redondeo (R_r), se tiene que:

$$V^{R}{}_{cal,k} = R_r + \frac{1}{2}\cdot\left[(V_{res,k} + R_c) - \sum_{i=k+1}^{N-1} V^{R}{}_{cal,i}\right] \qquad \textbf{(7.40)}$$

donde R_c y R_r son variables aleatorias.

Restando las ecuaciones **(7.39)** y **(7.40)** se obtiene el error cometido al obtener las $V_{cal,k}$, debido a los efectos del redondeo y de la cuantificación:

$$\Delta V_{cal,k} = V^{R}{}_{cal,k} - V_{cal,k} = \sum_{i=k}^{N} R_r \cdot 2^{-(N-i)} + \sum_{i=k}^{N} R_c \cdot 2^{-(N+1-i)} \qquad \textbf{(7.41)}$$

$$\downarrow$$

$$\left|\Delta V_{cal,k}\ |_{\max}\right| < 2\cdot R_r\ |_{\max} + R_c\ |_{\max} \qquad \textbf{(7.42)}$$

Y considerando que:

- La variable aleatoria R_c está uniformemente distribuida entre $\pm LSB/2$ del calDAC (LSB_{caldac})

- La variable aleatoria R_r está uniformemente distribuida entre $\pm 2^{-(B+1)}\ LSB_{caldac}$ Considerando que las divisiones por dos se realizan con M+B bits de resolución: donde M es el número de bits del calDAC y B es el exceso de bits empleados.

queda finalmente que:

$$\left|\Delta V_{cal,k}\ |_{\max}\right| < \frac{1}{2} + 2\cdot 2^{-(B+1)} = \frac{1}{2}\cdot\ 1 + 2^{-B+1} \quad LSB_{caldac} \qquad \textbf{(7.43)}$$

Con lo que se concluye que el error máximo cometido en cada $V_{cal,k}$ calculada será siempre menor a 1 LSB_{caldac} excepto para el caso en que se tome $B = 0$.

Para que la calibración sea eficaz, es necesario que los errores $\left|\Delta V_{cal,k}\ |_{\max}\right|$ sean siempre menores a $\pm 0.5 LSB$ del convDAC ($LSB_{convdac}$). La siguiente expresión determina la relación entre el LSB_{caldac} y el $LSB_{convdac}$, que es función del valor de C_{cal} y del número de bits de del calDAC (M) y del convDAC (N):

$$\begin{cases} LSB_{convdac} = \dfrac{1}{2^N} \\ LSB_{caldac} = \left(\dfrac{C_{cal}}{C_{cal}+C_{convdac}}\right)\cdot\dfrac{1}{2^M} \end{cases} \longrightarrow \quad \begin{array}{l} LSB_{convdac} = K\cdot LSB_{caldac} \\ ,\ donde \quad K = \dfrac{2^N}{2^M}\cdot\left(\dfrac{C_{cal}}{C_{cal}+C_{convdac}}\right) \end{array} \qquad \textbf{(7.46)}$$

Por lo tanto, la condición a cumplir para que los errores de cuantificación y redondeo no interfieran en una buena calibración es:

$$\begin{cases} \left|\Delta V_{cal,k}\ |_{\max}\right| < 0.5 \quad LSB_{convdac} \\ \left|\Delta V_{cal,k}\ |_{\max}\right| < \dfrac{1}{2}\cdot\ 1+2^{-B+1} \quad LSB_{caldac} \quad = K\cdot\dfrac{1}{2}\cdot\ 1+2^{-B+1} \quad LSB_{convdac} \end{cases}$$

$$\downarrow$$

$$K\cdot\frac{1}{2}\cdot\ 1+2^{-B+1}\ < 0.5 \quad \rightarrow \quad \frac{2^N}{2^M}\cdot\left(\frac{C_{cal}}{C_{cal}+C_{convdac}}\right)\cdot\frac{1}{2}\cdot\ 1+2^{-B+1}\ < 0.5 \qquad \textbf{(7.47)}$$

En resumen, para el diseño del sistema de calibración se deben tener en cuenta las condiciones establecidas por **(7.38)** y **(7.47)** para obtener un rango de calibración adecuado, y al mismo tiempo, que los errores de redondeo y cuantificación no afecten al buen funcionamiento del sistema. Los pasos a seguir para el diseño son:

- ✓ Fijar C_{cal} para conseguir un rango de la V_{cal} adecuado.
- ✓ Fijar B para que el error de redondeo no afecte.
- ✓ Fijar M para tener suficiente resolución y que el error de cuantificación del calDAC no afecte.

CAPÍTULO 8

DAC capacitivo

8.1 TÉCNICA DE *SPLIT CAPACITOR ARRAY*

Los DAC's de tipo capacitivo son muy populares entre los diseñadores CMOS, pues son relativamente sencillos de implementar y ofrecen una buena exactitud. Sin embargo, esta arquitectura empieza a presentar inconvenientes cuando se requieren resoluciones elevadas; a medida que el número de bits del DAC aumenta, la capacidad necesaria para representar el bit MSB se multiplica por dos, lo que hace incrementar de forma considerable el área del diseño.

Por ejemplo, para un DAC capacitivo de 12 bits como el de este proyecto, las capacidades requeridas serían:

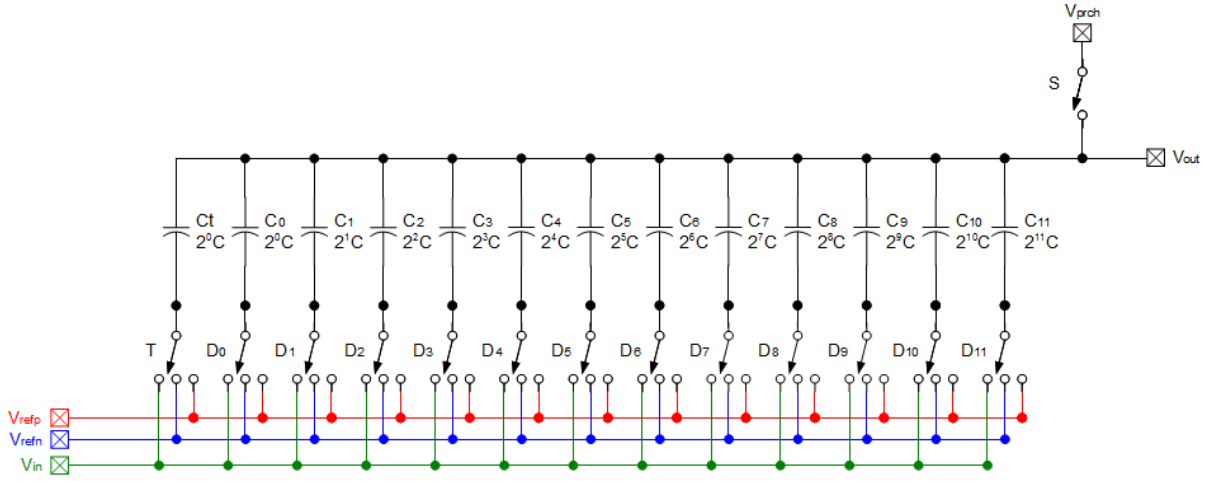


Figura 8.1: DAC capacitivo de 12 bits

si se toma como unidad de medida la capacidad unitaria (C), se puede afirmar que el número de capacidades unitarias para este DAC es de:

$$Núm\_caps = 2^0 + 2^0 + 2^1 + 2^2 + ... + 2^{10} + 2^{11} = 2^{12} = 4096 \quad \textbf{(8.1)}$$

teniendo en cuenta que la C mínima que se puede implementar con la tecnología de 0.35µm de *Austriamicrosystems* es de unos 70fF y ocupa alrededor de 131.1µm^2, el área necesaria para implementar sólo las 4096 capacidades es de 0.53mm^2, algo totalmente desorbitado para un DAC de 12 bits.

Una solución que permite disminuir de forma considerable el área de un DAC capacitivo con un número de bits elevado es, la técnica del *split capacitor array*. Esta técnica consiste en dividir el *array* capacitivo en varios *sub-arrays* de tamaño menor, uniéndolos entre sí a través de capacidades de acoplo (C_c). De esta forma se consigue reducir el número total de capacidades del DAC, y por consiguiente, el área que ocupan.

Si se aplica el *split capacitor array* sobre el DAC de 12 bits anterior, se observa que:

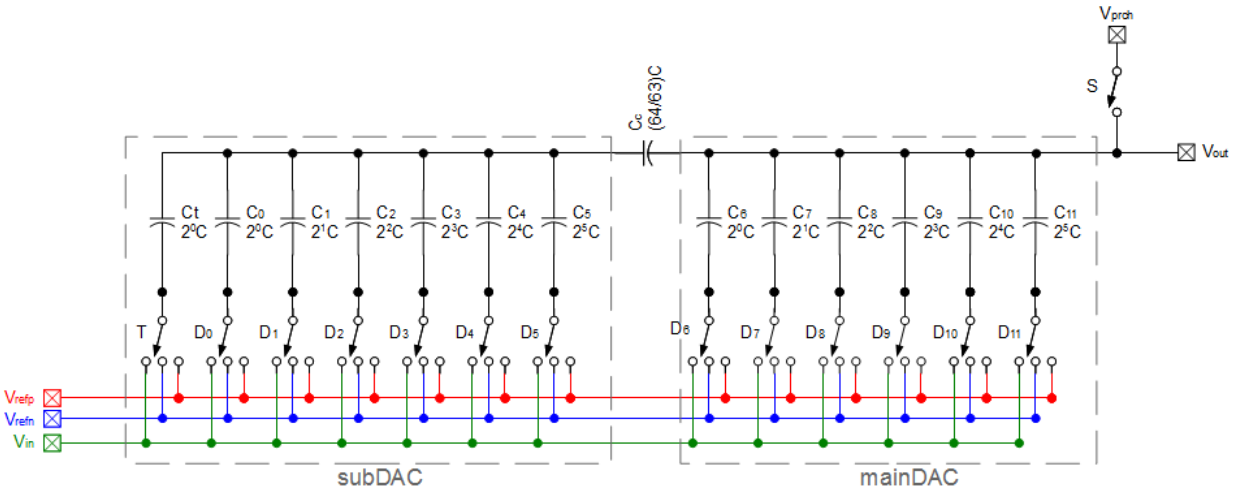


Figura 8.2: DAC capacitivo de 12 bits con split capacitor array 6-6

para esta configuración, el número total de capacidades requerido es:

$$\begin{aligned} Núm\_caps &= caps\_subarrays + caps\_acoplo = \\ &= \left[2^0 + 2\cdot(2^0 + 2^1 + ... + 2^5)\right] + C_c = 127 + C_c \end{aligned} \quad \textbf{(8.2)}$$

el cual contrasta con las 4096 capacidades que se utilizaban en la configuración original.

En el ejemplo anterior se ha optado por dividir el DAC original de 12 bits en dos *sub-arrays* de

6 bits, uniéndolos entre sí mediante la capacidad C_c, este tipo de estructura se define como 6-6 y no es la única que se puede emplear, también son posibles otras combinaciones como 8-4, 7-5,… O incluso utilizar más de dos *sub-arrays* junto con sus capacidades de acoplo correspondientes: 4-4-4, 5-5-2,…

La clave de este tipo de configuración reside en la atenuación introducida por las capacidades de acoplo. Mediante la capacidad C_c se consigue atenuar el efecto de las capacidades de los subDAC's, de tal manera que sus pesos comparados con los del mainDAC quedan reducidos, manteniéndose así el ratio binario entre todas las capacidades del DAC.

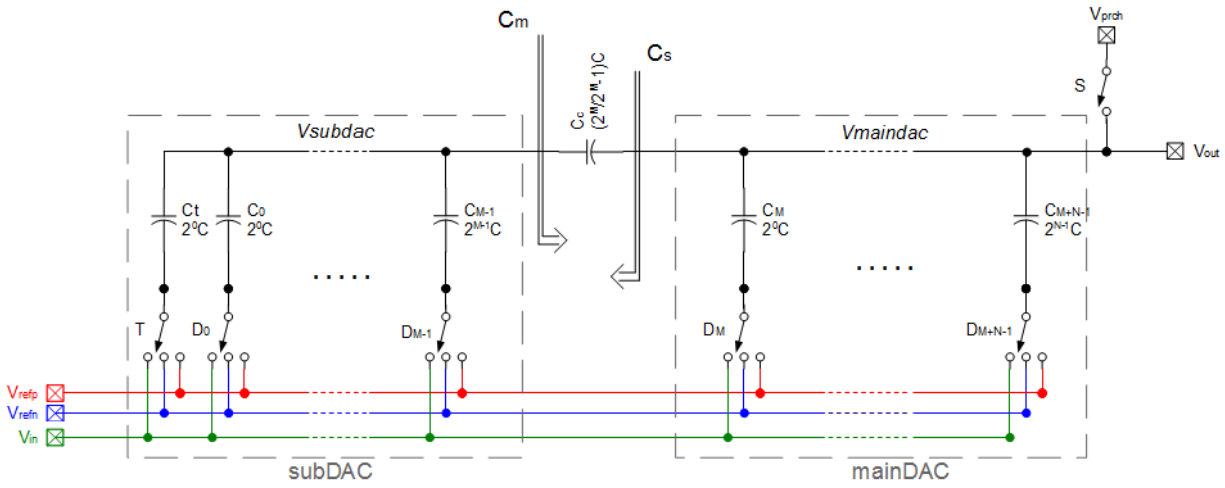


Figura 8.3: DAC capacitivo de (M+N) bits con split capacitor array M-N

N: *número de bits del mainDAC*

M: *número de bits del subDAC*

$M+N$: *número de bits del DAC*

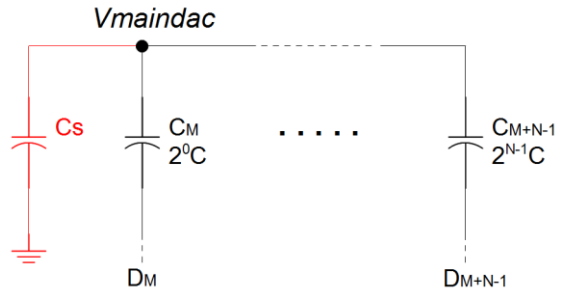
$$C_{maindac} = (2^N - 1)\cdot C \quad [\textit{capacidad total del mainDAC}])$$

$$C_{subdac} = 2^M \cdot C \quad [\textit{capacidad total del subDAC}]$$

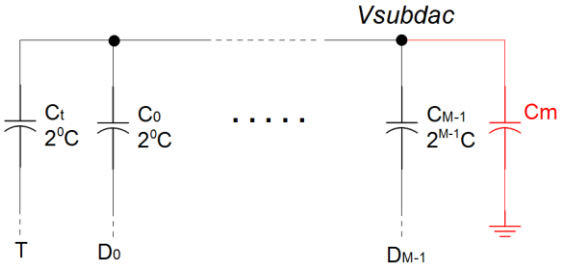
$$C_c = \left(\frac{C_{subdac}}{C_{subdac} - 1}\right)\cdot C = \left(\frac{2^M}{2^M - 1}\right)\cdot C \quad [\textit{capacidad de acoplo}]$$

$$C_m = \frac{C_c \cdot C_{maindac}}{C_c + C_{maindac}} = \frac{2^M \cdot (2^N - 1)}{2^M} + (2^M - 1)\cdot(2^N - 1)$$

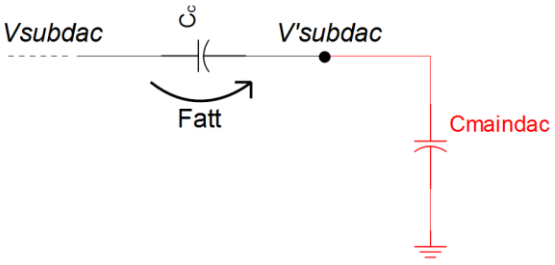
$$C_s = \frac{C_c \cdot C_{subdac}}{C_c + C_{subdac}} = \frac{2^M \cdot 2^M}{2^M + 2^M \cdot (2^M - 1)} = 1$$



$$V_{maindac} = \frac{(V_{refp} - V_{refn})}{C_{maindac} + C_s} \cdot \sum_{k=M}^{M+N-1} D_k \cdot 2^k \cdot C = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \sum_{k=M}^{M+N-1} D_k \cdot 2^k \quad \textbf{(8.3)}$$



$$V_{subdac} = \frac{(V_{refp} - V_{refn})}{C_{subdac} + C_m} \cdot \sum_{k=0}^{M-1} D_k \cdot 2^k \cdot C \quad \textbf{(8.4)}$$



$$F_{att} = \frac{V'_{subdac}}{V_{subdac}} = \frac{C_c}{C_c + C_{maindac}} = \frac{2^M}{2^M + (2^M - 1) \cdot (2^N - 1)} \quad \textbf{(8.5)}$$

$$V'_{subdac} = V_{subdac} \cdot F_{att} = \frac{(V_{refp} - V_{refn})}{C_{subdac} + C_m} \cdot F_{att} \cdot \sum_{k=0}^{M-1} D_k \cdot 2^k \cdot C = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \sum_{k=0}^{M-1} D_k \cdot \frac{2^k}{2^M} \quad \textbf{(8.6)}$$

$$V_{out} = V_{maindac} + V'_{subdac} = \frac{(V_{refp} - V_{refn})}{2^N} \cdot \sum_{k=-M}^{N-1} D_k \cdot 2^k \quad \textbf{(8.7)}$$

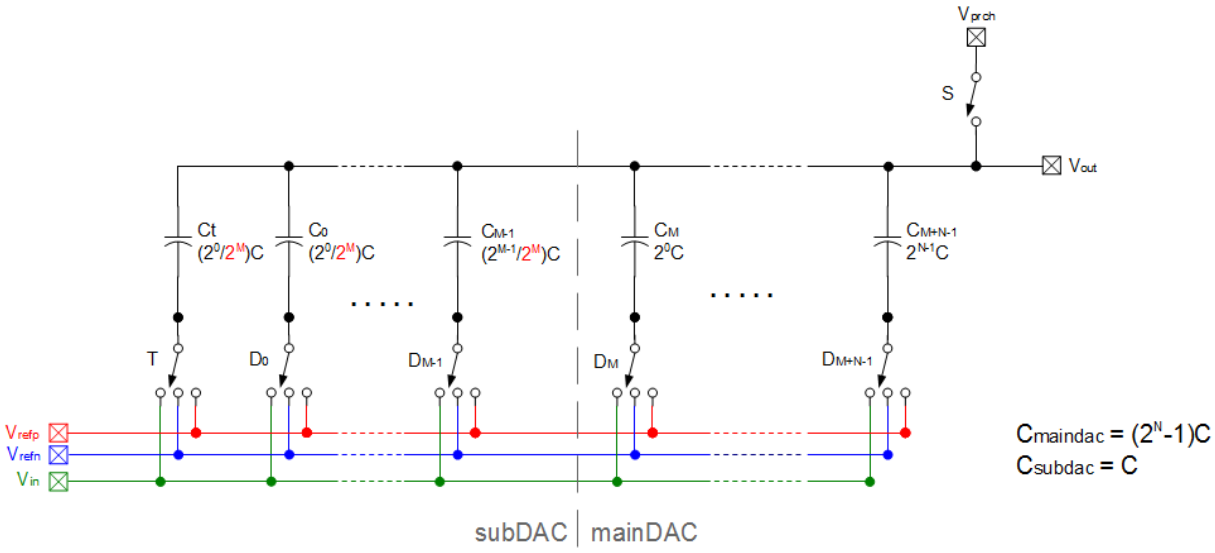


Figura 8.4: DAC capacitivo equivalente de (M+N) bits con split capacitor array M-N

Al aplicar esta técnica es imprescindible que las capacidades de acoplo entre cada *sub-array* estén correctamente dimensionadas, en caso contrario la linealidad se verá comprometida.

Si se tiene un valor de C_c inapropiado, la atenuación aplicada sobre los subDAC's no será la correcta con lo que, a pesar de tener unos subDAC's y un mainDAC totalmente lineales, el DAC en su conjunto presentará errores de linealidad. Estos errores de linealidad se van a manifestar en forma de picos de DNL cada 2^M códigos, es decir, en los códigos límite entre un *sub-array* y el siguiente.

Los problemas derivados de las capacidades de acoplo (C_c) van a ser muy difíciles de evitar: por una parte, el valor requerido para el C_c no va a ser múltiplo entero de C, con lo que se hace prácticamente imposible generarlo de forma precisa en el *layout*:

$$C_c = \left(\frac{capacidad\ \ total\ \ subDAC}{capacidad\ \ total\ \ subDAC\ \ -\ \ 1} \right) \cdot C \qquad \textbf{(8.8)}$$

por otro lado, hay que recordar que todas las capacidades van a presentar una capacidad parásita entre su *bottom-plate* y sustrato, lo cual en el caso del C_c va a ser crítico, pues al estar conectado en serie va a ser inevitable que su parásito afecte a alguno de los dos *sub-arrays*.

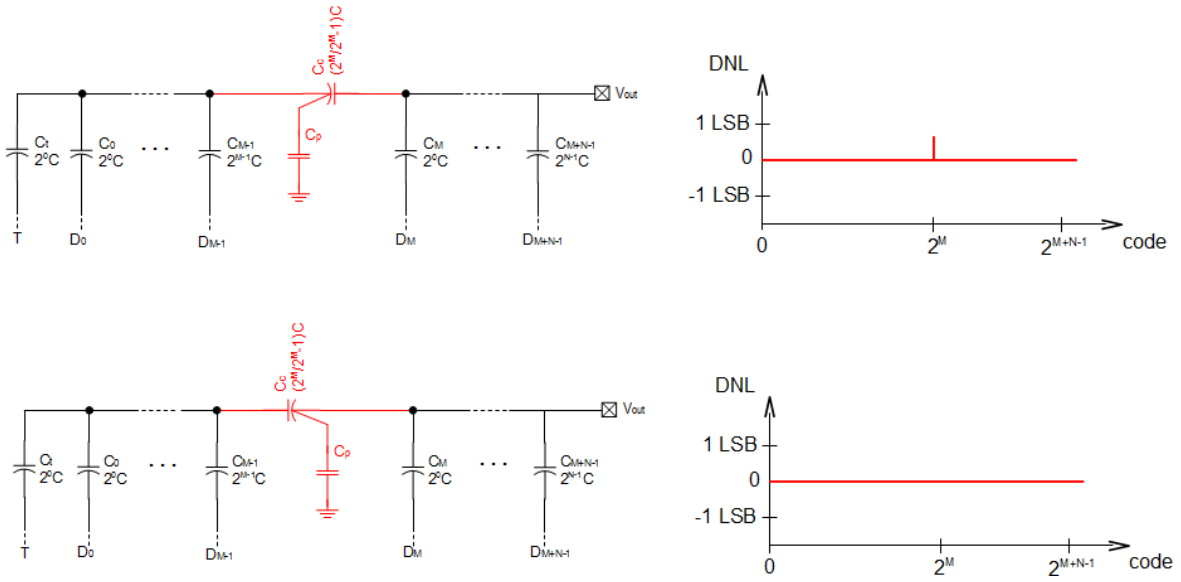


Figura 8.5: Efecto de la capacidad parásita a sustrato de la C_c

En el caso concreto de utilizar una estructura con un mainDAC y un solo subDAC, sí que es posible anular el efecto de la capacidad parasita del C_c. Si se conecta el C_c de tal manera que la capacidad parásita afecte al mainDAC el problema de linealidad se soluciona, pues una capacidad conectada en el nodo de salida del DAC genera una simple atenuación que se compensa durante el muestreo de la señal de entrada, con lo que la *fdt* del DAC no se ve afectada en estos casos.

8.2 ARQUITECTURA

Debido a los requerimientos de calibración de este DAC, los cuales fueron justificados en el capitulo anterior, la arquitectura básica a implementar se compone de dos partes: el DAC de conversión (convDAC) y el DAC de calibración (calDAC).

El convDAC, como su nombre indica, va a ser el encargado de realizar las conversiones digital-analógicas que se precisen en cada momento durante el proceso SAR. Según las especificaciones impuestas va a ser un DAC capacitivo de 12 bits.

Por su parte, el calDAC va a realizar labores de calibración, compensando para cada código de entrada del convDAC los errores de linealidad generados por los defectos de *matching* de sus capacidades. El DAC de calibración va a ser también de tipo capacitivo, con un número de bits M y una capacidad de atenuación C_{cal} fijadas según las necesidades de calibración del DAC principal.

8.2.1 DAC de conversión (convDAC)

Para poder cumplir con las especificaciones de área impuestas, se hace indispensable aplicar la técnica de *split capacitor array* sobre el DAC de conversión, pues para un DAC capacitivo de 12 bits se requieren 4096 capacidades unitarias, algo inviable para un diseño de área reducida, como es éste el caso.

Existen varias estructuras posibles para un *split capacitor array* de 12 bits, cada una de ellas tiene sus ventajas e inconvenientes, pero la mayor diferencia entre unas y otras viene determinada por el número de subDAC's que emplean y el número de bits de cada uno de ellos. A más subDAC's, el número de bits de cada uno de ellos será menor, lo cual va a repercutir directamente sobre el área, pues el número de capacidades unitarias requeridas se reduce. Pero por otro lado, para cada subDAC que se define es necesaria una capacidad de acoplo (C_c) asociada, y como se ha estudiado en el apartado anterior, estas capacidades provocan problemas de linealidad. Teniendo en cuenta lo expuesto, hay que analizar los pros y contras de cada estructura y llegar a un compromiso entre el área que ocupa y los problemas de linealidad que conlleva.

Para un DAC de 12 bits, las dos posibles estructuras con mayor flexibilidad a la hora de su implementación son: la 6-6 y la 4-4-4, pues presentan todos sus *sub-arrys* idénticos, con el mismo número de bits, lo cual va a traducirse en un *layout* más regular, facilitándose la optimización de su área, y en una mayor facilidad de ampliación si se requiriesen resoluciones mayores en un futuro.

Partiendo de estas dos estructuras, y no olvidando que se va a implementar un sistema de calibración, la estructura más adecuada es la 4-4-4, pues el número de capacidades unitarias que requiere es 2.6 veces menor que el de la 6-6, y al utilizar calibración los problemas adicionales de linealidad añadidos por el hecho de utilizar dos capacidades de acoplo no son tan críticos, pues van a ser compensados.

En la siguiente figura se muestra la estructura final que implementará el DAC de conversión:

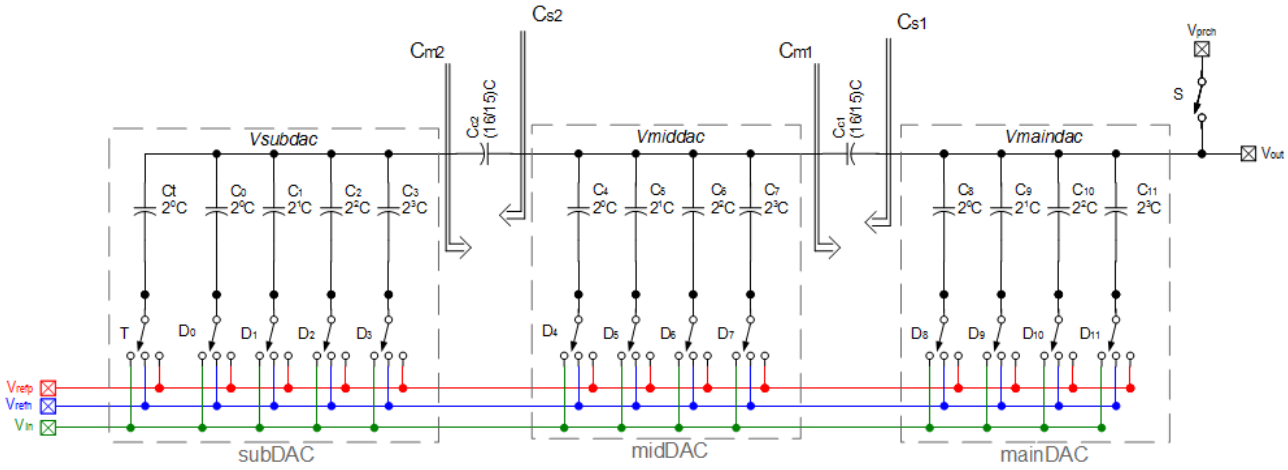


Figura 8.6: DAC capacitivo de 12 bits con split capacitor array 4-4-4

Las ecuaciones que definen su comportamiento son las siguientes:

N: *número de bits del mainDAC*

M: *número de bits del subDAC*

$M+N$: *número de bits del DAC*

$$C_{maindac} = (2^4 - 1) \cdot C = 15 \cdot C \quad [capacidad\ total\ del\ mainDAC])$$

$$C_{middac} = (2^4 - 1) \cdot C = 15 \cdot C \quad [capacidad\ total\ del\ midDAC]$$

$$C_{subdac} = 2^4 \cdot C = 16 \cdot C \quad [capacidad\ total\ del\ subDAC]$$

$$C_{c1} = \left(\frac{C_{middac} + C_{s2}}{C_{middac} + C_{s2} - C} \right) \cdot C = \left(\frac{16}{15} \right) \cdot C \quad [capacidad\ de\ acoplo1]$$

$$C_{c2} = \left(\frac{C_{subdac}}{C_{subdac} - C} \right) \cdot C = \left(\frac{16}{15} \right) \cdot C \quad [capacidad\ de\ acoplo2]$$

$$C_{m1} = \frac{C_{c1} \cdot C_{maindac}}{C_{c1} + C_{maindac}} = \frac{\left(\frac{16}{15}\right) \cdot 15}{\left(\frac{16}{15}\right) + 15} = 0.9958$$

$$C_{m2} = \frac{C_{c2} \cdot (C_{middac} + C_{m1})}{C_{c2} + (C_{middac} + C_{m1})} = \frac{\left(\frac{16}{15}\right) \cdot (15 + 0.9958)}{\left(\frac{16}{15}\right) + (15 + 0.9958)} = 0.9998$$

$$C_{s2} = \frac{C_{c2} \cdot C_{subdac}}{C_{c2} + C_{subdac}} = \frac{\left(\frac{16}{15}\right) \cdot 16}{\left(\frac{16}{15}\right) + 16} = 1$$

$$C_{s1} = \frac{C_{c1} \cdot (C_{middac} + C_{s2})}{C_{c1} + (C_{middac} + C_{s2})} = \frac{\left(\frac{16}{15}\right) \cdot (15 + 1)}{\left(\frac{16}{15}\right) + (15 + 1)} = 1$$

$$V_{maindac} = \frac{(V_{refp} - V_{refn})}{C_{maindac} + C_{s1}} \cdot \sum_{k=8}^{11} D_k \cdot 2^k \cdot C = \frac{(V_{refp} - V_{refn})}{16} \cdot \sum_{k=8}^{11} D_k \cdot 2^k \quad \textbf{(8.9)}$$

$$V_{middac} = \frac{(V_{refp} - V_{refn})}{C_{middac} + C_{s2} + C_{m1}} \cdot \sum_{k=4}^{7} D_k \cdot 2^k \cdot C = \frac{(V_{refp} - V_{refn})}{16.9958} \cdot \sum_{k=4}^{7} D_k \cdot 2^k \quad \textbf{(8.10)}$$

$$V_{subdac} = \frac{(V_{refp} - V_{refn})}{C_{subdac} + C_{m2}} \cdot \sum_{k=0}^{3} D_k \cdot 2^k \cdot C = \frac{(V_{refp} - V_{refn})}{16.9998} \cdot \sum_{k=0}^{3} D_k \cdot 2^k \quad \textbf{(8.11)}$$

$$F_{att1} = \frac{V'_{middac}}{V_{middac}} = \frac{C_{c1}}{C_{c1} + C_{maindac}} = \frac{\left(\frac{16}{15}\right)}{\left(\frac{16}{15}\right) + 15} = 0.0664 \quad \textbf{(8.12)}$$

$$F_{att2} = \frac{V'_{subdac}}{V_{subdac}} = \frac{C_{c2}}{C_{c2} + C_{middac} + C_{m1}} = \frac{\left(\frac{16}{15}\right)}{\left(\frac{16}{15}\right) + 15 + 0.9958} = 0.9999 \quad \textbf{(8.13)}$$

$$V_{out} = V_{maindac} + V_{middac} \cdot F_{att1} + V_{subdac} \cdot F_{att2} \quad \textbf{(8.14)}$$

Un aspecto a destacar de esta estructura es la posición en la que se conectan los condensadores de acoplo (C_{c1} y C_{c2}) con la finalidad de evitar, en la medida de los posible, los errores de linealidad derivados de las capacidades parasitas de *bottom-plate*. En el caso del C_{c1}, que une el mainDAC y el midDAC, es posible evitar el efecto negativo de la capacidad parásita si se conecta su *bottom-plate* al mainDAC, en este caso la capacidad parásita recaerá sobre el mainDAC, y como se vio anteriormente, una parásito en el nodo de salida del DAC no influye sobre la linealidad de éste. Para el C_{c2}, que conecta el midDAC y el subDAC, va a ser imposible evitar que su capacidad parásita ocasione errores de linealidad, partiendo de este hecho se situará el C_{c2} de forma que el error de linealidad cometido sea el menor posible, para ello se conectará el *bottom-plate* al subDAC, pues una capacidad parásita en el subDAC genera un error de linealidad menor que el que generaría la misma capacidad parásita en el midDAC.

8.2.2 DAC de calibración (calDAC)

Para abordar el diseño del DAC de calibración es necesario conocer de antemano el número de bits (M) y el valor de la capacidad de calibración (C_{cal}) requeridos para compensar adecuadamente los errores de linealidad del DAC de conversión.

Conociendo la estructura utilizada por el DAC de conversión (*split capacitor array* 4-4-4), es posible estimar los problemas de linealidad que va a presentar. Las no linealidades del convDAC van a venir determinadas básicamente por: el *matching* entre las capacidades, la capacidad parásita asociada a C_{c2} y el efecto de otros parásitos que puedan aparecer en el *layout*. De estas tres contribuciones, el *matching* entre capacidades es la que tiene mayor peso, por lo que para esta estimación se van a despreciar los efectos de las capacidades parásitas, y se va a considerar que la no linealidad del convDAC va estar determinada exclusivamente por el *matching* entre sus capacidades.

Para este proyecto, en el cual conseguir un DAC de dimensiones reducidas es el objetivo prioritario, se estima que el *matching* logrado con un *layout* de área muy reducida (valor de capacidad unitaria C mínimo y estilo de *layout* no excesivamente cuidado) va a estar en torno a unos 8 bits. Un *matching* más bien pobre, pero suficiente para un sistema con calibración.

Un *matching* de 8 bits significa que, en el caso hipotético de tener un DAC de 8 bits, el ratio entre las capacidades será lo suficientemente preciso como para no tener errores de linealidad. O lo que

es lo mismo, que el ratio entre capacidades va a tener una exactitud mínima del: $\pm\left(\frac{1}{2^8}\right)\%$.

DAC de 8bits:

caso peor de ratio entre capacidades:

$$\frac{(2^8-1)\cdot C}{2^8\cdot C}=0.9960 \quad \rightarrow \quad \frac{1}{2^8}\% \;\; de \;\; C \;\; \approx \;\; 0.5\% \;\; de \;\; C$$

es decir, que con un *matching* de 8 bits se estará cometiendo un error máximo de $\pm 0.5\%$ en cada una de las capacidades del convDAC, con lo que:

$$C_k = 2^k \cdot C \cdot (1+\varepsilon_k) \quad , donde \quad \varepsilon_k \in [-0.5\%, 0.5\%] \qquad \textbf{(8.15)}$$

$$k = N-1,...,0$$

A partir de la estimación de los errores de *matching* máximos para cada capacidad del convDAC (ε_k), es posible calcular los errores de linealidad asociados a ellos. La tensión de error máxima ($V_{error}|_{\max}$) generada por el convDAC en el caso peor, es decir, cuando el código a convertir sea el 11...1 (todas las capacidades activas) y los errores de *matching* (ε_k) sean todos máximos, se puede calcular como:

Peor caso:

$\varepsilon_k = \varepsilon_k|_{\max} = \pm 0.5\% \quad , k = N-1,...,0$

$D_k = 1 \quad , k = N-1,...,0$

$$V_{error,k} = \frac{(V_{refp}-V_{refn})}{2^N}\cdot 2^k \cdot \varepsilon_k \quad , k = N-1,...,0 \qquad \textbf{(8.16)}$$

$$V_{error} = \sum_{k=0}^{N-1} D_k \cdot V_{error,k} \quad \rightarrow \quad V_{error}|_{\max} = \frac{(V_{refp}-V_{refn})}{2^N}\cdot \sum_{k=0}^{N-1} 2^k \cdot \varepsilon_k|_{\max} =$$

$$= \frac{(V_{refp}-V_{refn})}{2^N}\cdot \varepsilon_k|_{\max}\cdot[2^N-1] \approx \qquad \textbf{(8.17)}$$

$$\approx (V_{refp}-V_{refn})\cdot \varepsilon_k|_{\max}$$

con lo que se obtiene:

$$V_{error}\,|_{\max} \approx \pm(V_{refp} - V_{refn}) \cdot 0.005 \qquad \textbf{(8.18)}$$

Ahora, obtenida la tensión de error máxima ($V_{error}\,|_{\max}$) que el calDAC va a tener que compensar, es posible aplicar las ecuaciones de diseño de la calibración (desarrolladas en el capítulo 7) y determinar las características requeridas para el DAC de calibración:

1. En primer lugar, se determina el valor de la capacidad de calibración (C_{cal}) necesaria para cubrir el rango de calibración requerido:

   Rango de tensiones de error (V_{error}) a calibrar:

$$V_{error} \in [-V_{error}\,|_{\max}, +V_{error}\,|_{\max}] = [-(V_{refp} - V_{refn}) \cdot 0.005, +(V_{refp} - V_{refn}) \cdot 0.005] \qquad \textbf{(8.19)}$$

   Según la ecuación **(7.38)** deducida en el capítulo siete, la relación entre C_{cal} y la $V_{cal}\,|_{\max}$ de calibración es:

$$\left|V_{cal}\,|_{\max}\right| < \frac{(V_{refp} - V_{refn})}{2} \cdot \frac{C_{cal}}{C_{cal} + C_{convdac}} \qquad \textbf{(7.38)}$$

   Sustituyendo valores y despejando se obtiene que:

$$\begin{cases} \left|V_{cal}\,|_{\max}\right| = (V_{refp} - V_{refn}) \cdot 0.005 \\ C_{cal} = \alpha \cdot C \\ C_{convdac} = 2^4 \cdot C \end{cases} \longrightarrow \quad 0.005 < \frac{1}{2} \cdot \left(\frac{\alpha}{\alpha + 2^4} \right) \qquad \textbf{(8.20)}$$

$$\boxed{\alpha > 0.1\hat{6}}$$

   Redondeando el valor de α, para que la C_{cal} se pueda obtener fácilmente a partir de combinaciones de capacidades unitarias, se tiene que:

$$\boxed{C_{cal} = \alpha \cdot C = \frac{1}{6} \cdot C} \qquad \textbf{(8.21)}$$

La cual se puede sintetizar a partir de 6 capacidades unitarias conectadas en serie.

2. A continuación se fija el número de bits exceso (B) que se emplearán para realizar las operaciones aritméticas de cálculo de los códigos de calibración.

Tomando B = 8, los errores de redondeo cometidos al realizar las operaciones aritméticas ya se hacen despreciables. Notar que el valor de B no va a ser crítico para este diseño, pues al implementar una calibración estática, la lógica para el cálculo de los códigos de calibración va a ser externa al chip.

3. Finalmente se determinará el numero de bits (M) necesario, para que los errores de cuantificación del calDAC no afecten al resultado final de la calibración.

Según la ecuación **(7.47)** deducida en el capítulo siete, la condición a cumplir para que los errores de cuantificación del calDAC y los errores de redondeo de las operaciones aritméticas no generen un error en la compensación superior a $0.5\ LSB's$, es:

$$\frac{2^N}{2^M}\cdot\left(\frac{C_{cal}}{C_{cal}+C_{convdac}}\right)\cdot\frac{1}{2}\cdot\ 1+2^{-B+1}\ <0.5 \qquad \textbf{(7.47)}$$

Sustituyendo valores y despejando el valor de M se obtiene que:

$$\begin{cases} N=12 \\ C_{cal}=\alpha\cdot C=\dfrac{1}{6}\cdot C \\ C_{convdac}=2^4\cdot C \\ B=8 \end{cases} \longrightarrow \quad M>\log_2\left\{2\cdot 2^N\cdot\frac{C_{cal}}{C_{cal}+C_{subdac}}\cdot\frac{1}{2}\cdot(1+2^{-B+1})\right\} \qquad \textbf{(8.22)}$$

$$\downarrow$$

$$\boxed{M>5.4bits\approx 6bits}$$

Con lo que se concluye que el calDAC deberá tener al menos 6 bits de resolución.

Para la implementación del calDAC se ha optado por utilizar una réplica del subDAC y midDAC del DAC de conversión, de esta forma se obtiene una estructura sencilla y pequeña, que cumple las características requeridas por la calibración y además regulariza el conjunto convDAC+calDAC, haciendo que todos los *sub-arrays* de la arquitectura sean de 4 bits. Como resultado se obtiene un calDAC de 8 bits y no de 6 bits, que sería el mínimo necesario, con lo que

se deja así un pequeño margen.

En la siguiente figura se muestra la estructura final del conjunto convDAC+calDAC. Se puede observar que el calDAC es una réplica exacta del subDAC y midDAC del DAC de conversión, con la peculiaridad de que su capacidad de acoplo C'_{c2} se conecta de forma diferente, esto es así para conseguir que la capacidad parásita no afecte a la linealidad del calDAC.

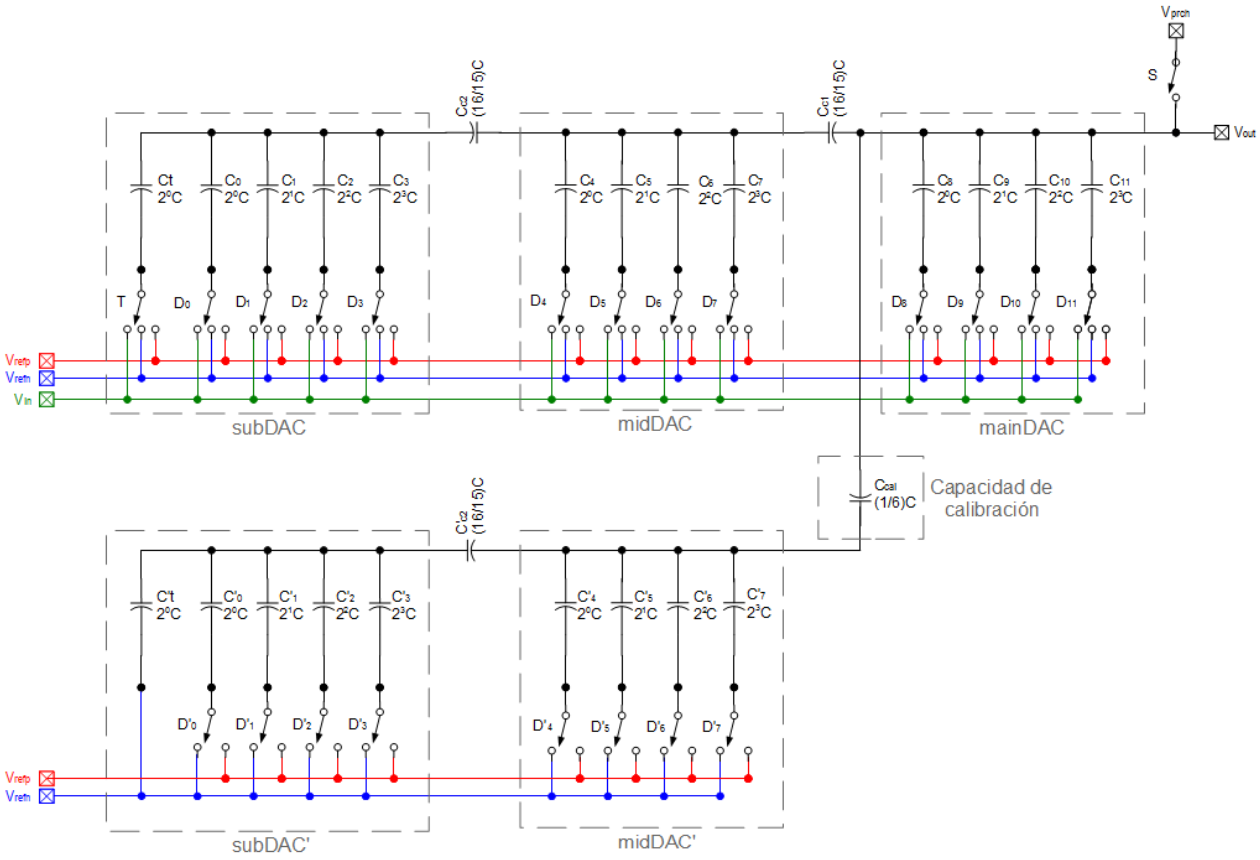


Figura 8.7: Arquitectura final del DAC

8.3 LAYOUT

Como se ha introducido anteriormente, los problemas de linealidad del DAC son originados por el *matching* entre capacidades y el efecto de las capacidades parásitas del *layout*. Es decir, las características del *layout* implementado son las que van a definir los problemas de linealidad del DAC.

En este punto se van a describir las características fundamentales a tener en cuenta a la hora de implementar el *layout* de un DAC capacitivo, y posteriormente, se detallarán las peculiaridades utilizadas en este DAC con el objetivo de conseguir un compromiso entre: una linealidad no demasiado desastrosa y que pueda ser compensada por la calibración, y un área lo suficientemente reducida como para cumplir las especificaciones de este diseño.

8.3.1 Características generales

A la hora de implementar el *layout* de un DAC de tipo capacitivo, uno de los principales problemas es controlar el ratio entre capacidades, a medida que el número de bits del convertidor aumenta, el ratio entra la capacidad del MSB y del LSB es hace más difícil de controlar. Por ejemplo, para un DAC de 3 bits, implementado con sus correspondientes tres capacidades, el ratio entre éstas se va a ver claramente afectado por el *fringe effect* (efecto de bordes), véase la figura 8.8.

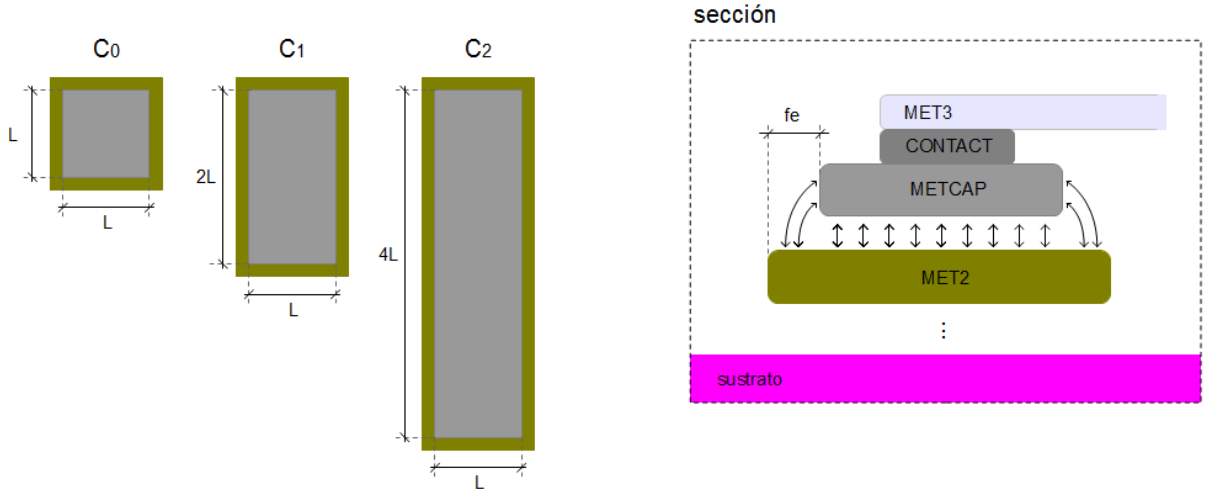


Figura 8.8: Layout para un DAC capacitivo de 3 bits (I)

C: *capacidad unitaria*

C_L: *capacidad de fringe – effect para un lado L*

$$C_2 = 4 \cdot C + 10 \cdot C_L \quad \textbf{(8.23)}$$

$$C_1 = 2 \cdot C + 6 \cdot C_L \quad \textbf{(8.24)}$$

$$C_0 = \cdot C + 4 \cdot C_L \quad \textbf{(8.25)}$$

La solución a este problema consiste en implementar el *array* capacitivo a partir de la interconexión de capacidades iguales, capacidades unitarias. Con lo que el *fringe effect*, que afecta por igual a cada capacidad unitaria, no modificará el ratio entre capacidades, quedando éste inalterado.

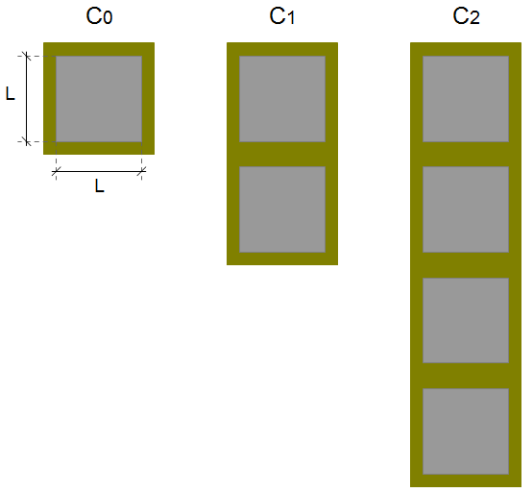


Figura 8.9: Layout para un DAC capacitivo de 3 bits (II)

C: *capacidad unitaria*

C_L: *capacidad de fringe – effect para un lado L*

$$C_2 = 4 \cdot C + 16 \cdot C_L \quad \textbf{(8.26)}$$

$$C_1 = 2 \cdot C + 8 \cdot C_L \quad \textbf{(8.27)}$$

$$C_0 = \cdot C + 4 \cdot C_L \quad \textbf{(8.28)}$$

Otro de los problemas que afecta al *matching* de las capacidades del DAC es el crecimiento no uniforme del oxido de silicio (aislante) en el proceso de fabricación. Este efecto provoca que el ratio entre las capacidades se vea alterado, pues el valor de las capacidades unitarias no va a ser exactamente el mismo en todas ellas, va a depender del gradiente del oxido de silicio y de la posición de cada capacidad unitaria dentro de la oblea.

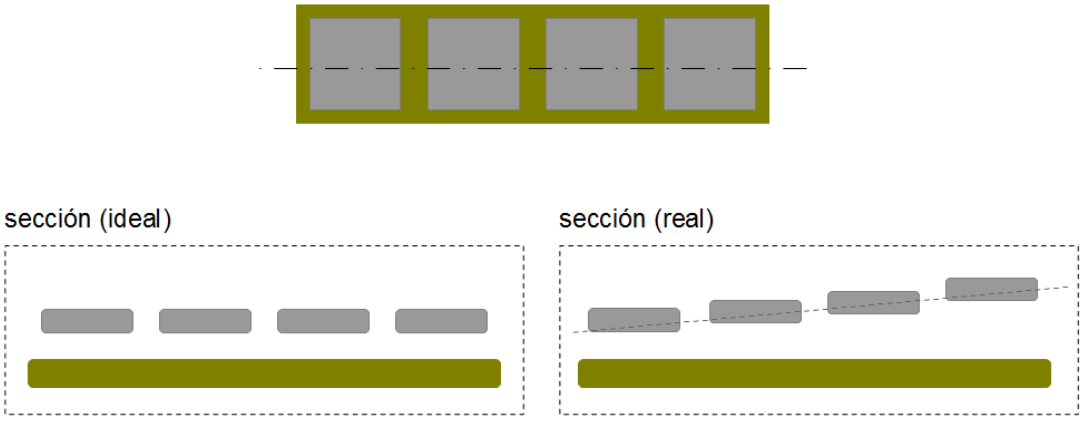


Figura 8.10: Crecimiento no uniforme del óxido de silicio

Con el fin de cancelar este efecto, se utilizan técnicas especiales para la distribución de las capacidades unitarias dentro de la oblea, la más popular es la distribución en centroide común. El inconveniente de estas técnicas es que dificultan el rutado entre las capacidades, el cual puede llegar a ser muy complejo para DAC's de resoluciones elevadas.

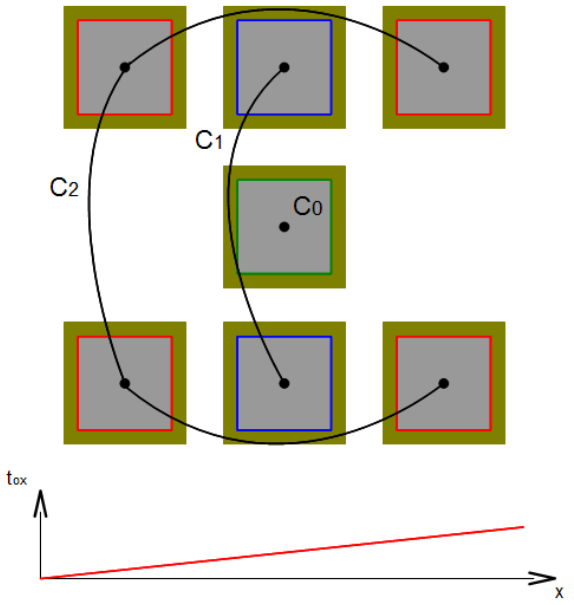


Figura 8.11: Layout para un DAC capacitivo de 3 bits (III)

Habitualmente no se utiliza centroide común en la implementación de DAC's de este tipo, pues como se observa, la compensación solo funciona para gradientes de primer orden y la complejidad del rutado aumenta considerablemente. Normalmente, se intenta elegir una distribución para las

capacidades unitarias lo más regular posible, pero manteniendo en todo momento un rutado sencillo.

Otro de los elementos a tener muy en cuenta al elaborar el *layout* de un DAC capacitivo son las capacidades parásitas debidas al rutado. Entre el metal que se utiliza para interconectar entre si los *top-plates* de cada capacidad y el metal que define las propias capacidades unitarias va a sintetizarse una capacidad parásita. Estas capacidades quedan en paralelo con las capacidades unitarias, modificando su valor y afectando de esta forma al *matching* y a la linealidad del DAC. Para evitar que el ratio entre capacidades quede modificado, el rutado debe realizarse de tal manera que el parásito asociado a cada capacidad este escalado de forma binaria. Veámoslo en el siguiente ejemplo de rutado:

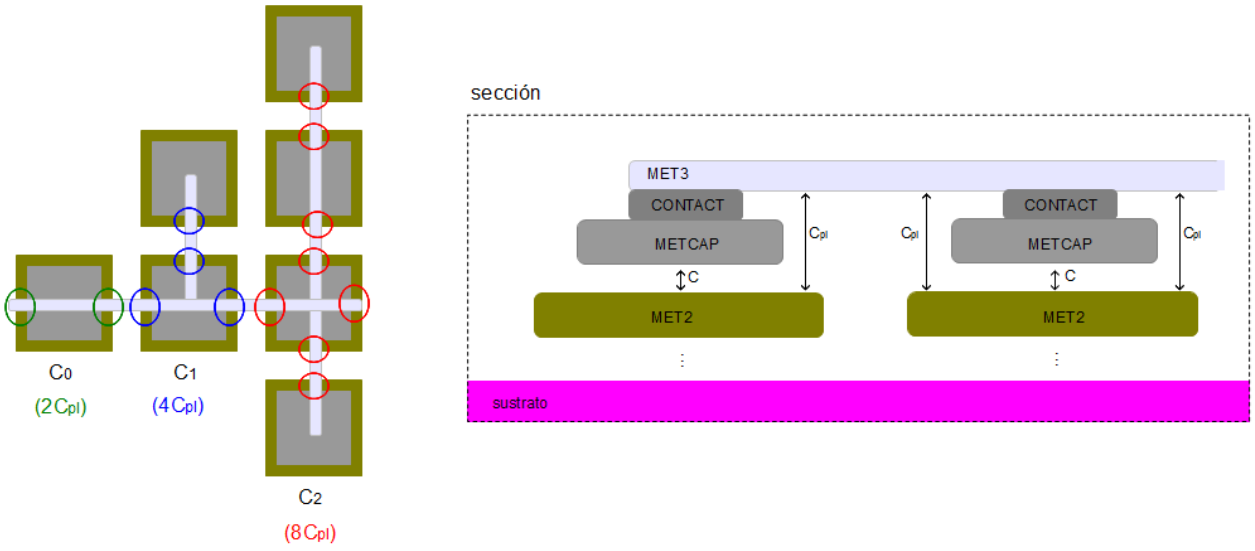


Figura 8.12: Layout para un DAC capacitivo de 3 bits (IV)

C: *capacidad unitaria*

C_{pl}: *capacidad parásita asociada a una tira de metal del bottom – plate*

$$C_2 = 4 \cdot C + 8 \cdot C_{pl} \quad \textbf{(8.29)}$$

$$C_1 = 2 \cdot C + 4 \cdot C_{pl} \quad \textbf{(8.30)}$$

$$C_0 = \cdot C + 2 \cdot C_{pl} \quad \textbf{(8.31)}$$

Por último, conseguir que el entorno de todas las capacidades sea exactamente el mismo, también mejorará las características de *matching* del *layout*. Para ello se hace uso de las denominadas capacidades *dummies*, es decir, capacidades idénticas a las capacidades unitarias que están cortocircuitadas y no tienen relevancia en el funcionamiento del circuito, simplemente tienen la finalidad de homogenizar el *layout*. Habitualmente, las *dummies* se sitúan en el contorno del *layout*.

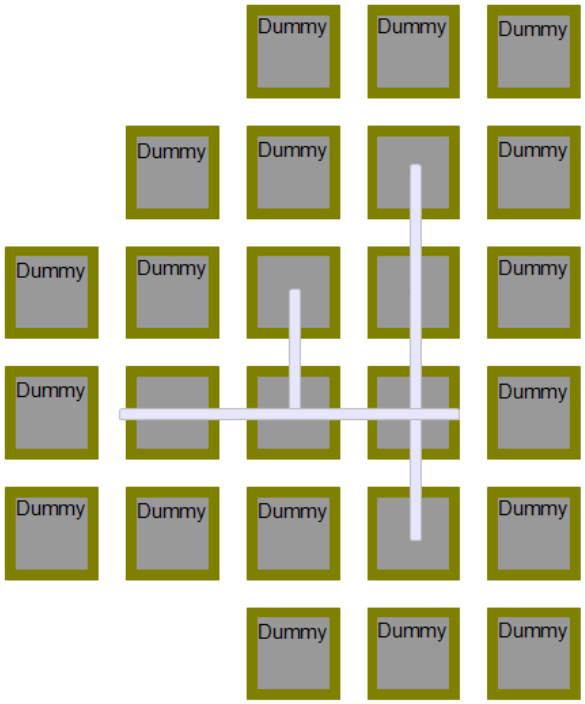


Figura 8.13: Layout para un DAC capacitivo de 3 bits (V)

8.3.2 Capacidad unidad

El proceso C35 de *Austriamicrosystems* permite la implementación de dos tipos de capacidades pasivas: las capacidades de polisilicio (Cpoly) y las capacidades de metal (Cmim). Cada una de ellas presenta sus propias características pero las mayores diferencias entre ambas son:

- Las Cmim sintetizan capacidades con valores más exactos que las Cpoly, es decir, sus tolerancias son menores.
- Las Cmim se encuentran más alejadas del sustrato que las Cpoly, lo cual se traduce en una menor capacidad parásita a sustrato.
- El ratio capacidad/área es mayor en las Cmim que en las Cpoly, es decir, para una cierta área el valor de la capacidad Cmim es mayor que el de la Cpoly.

- El ratio *matching*/capacidad conseguido para capacidades Cmim es mayor que el que se obtiene para capacidades Cpoly.

Para la implementación del DAC capacitivo, las características más relevantes a tener en cuenta son el *matching* y el valor de los parásitos a sustrato. En cuanto a los parásitos, las capacidades de metal suponen una cierta ventaja frente a las de polisilicio, pues sus valores son menores. Por otro lado, en el *matching*, las diferencias entre ambas no son significativas, pues aunque el factor *matching*/capacidad es superior en las Cmim, como estas requieren de áreas menores para sintetizar una misma capacidad, se llega a la conclusión de que el ratio *matching*/área de ambas es muy similar.

Por lo tanto, después de hacer este pequeño análisis de los dos tipos de capacidades se ha optado por el uso de las Cmim, ya que sus características de *matching* son similares a las de Cpoly pero sus parásitos, que influirán sobre la linealidad del DAC, son menores.

El valor elegido para la capacidad unitaria (C) ha sido el mínimo impuesto por las reglas de diseño para una capacidad de metal. El valor de C ha quedado en torno a unos 70fF aproximadamente. De esta forma, eligiendo el valor de capacidad mínimo, se conseguirá un área aún más pequeña.

La única limitación en cuanto al valor de C viene impuesta por el ruido $K \cdot T / C$ del muestreo. En este caso, al estar muestreando la señal de entrada en todas las capacidades del convDAC, la capacidad equivalente durante el muestreo valdrá:

$$C_{tot} = 2^0 \cdot C + 2^0 \cdot C + 2^1 \cdot C + 2^2 \cdot C + 2^3 \cdot C = 16 \cdot C = 16 \cdot 70\, fF \approx 1120\, fF \qquad \textbf{(8.32)}$$

por lo que la tensión eficaz de ruido (V_{ruido}) que se muestreará junto a la señal de entrada será de:

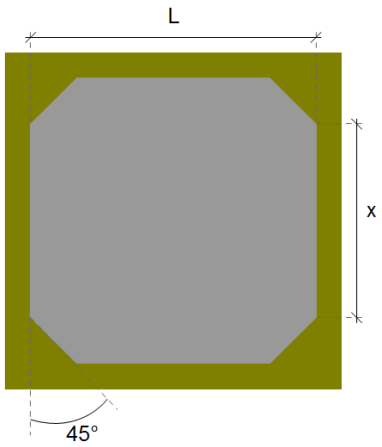
$$\begin{cases} V_{ruido} = \sqrt{\dfrac{K \cdot T}{C}} \\ K: \quad cte \quad Boltzman \quad (1.3806504 \cdot 10^{-23}\, {}^{J}\!/\!_{K}) \\ T: \quad Temperatura \quad (300K) \\ C: \quad capacidad \quad muestreo \quad (1120\, fF) \end{cases} \longrightarrow \quad V_{ruido} = 60.80\mu V << V_{LSB}$$

La cual es mucho menor que la tensión del LSB del convertidor ($V_{LSB} \approx 610.35\mu V$), por lo que el valor de C elegido ($\sim 70\ fF$) es suficiente para que el ruido $K \cdot T/C$ no sea significativo.

En la figura 8.14, se muestra el *layout* que implementa una capacidad unitaria. El valor de la capacidad sintetizada depende no solo del área, sino también del perímetro del METCAP, de tal manera que:

$$C = C_a(fF/\mu m^2) \cdot Área(\mu m^2) + C_p(fF/\mu m) \cdot Área(\mu m) \quad \textbf{(8.33)}$$

Lo que interesa en estos casos es utilizar una geometría del METCAP que maximice su área con respecto al perímetro, para de esta forma conseguir que la capacidad sintetizada sea más dependiente del área que del perímetro. Pues el perímetro es más sensible a variaciones durante el proceso de fabricación que el área, y de esta forma se consigue una capacidad más estable.



Maximiza (área vs. Perímetro)

$$x = \sqrt{1+\sqrt{2}} \cdot \left[2 - \sqrt{1+\sqrt{2}}\right] \cdot L \quad \rightarrow \quad x \approx 0.7 \cdot L \quad \textbf{(8.33)}$$

$$\begin{cases} A = 0.995 \cdot L^2 \\ P = 3.65 \cdot L \end{cases} \longrightarrow \frac{A}{P} = 0.262 \quad \textbf{(8.34)}$$

Figura 8.14: Layout de la capacidad unitaria C

8.3.3 Estilo de layout

Para la implementación del DAC (conjunto convDAC+calDAC) se opta por un estilo de *layout* sencillo y lo más regular/modular posible, cuya principal característica reside en el uso de un área muy reducida a cambio de sacrificar el *matching* entre capacidades.

La estructura básica del *layout* está formada por cinco bloques idénticos, cada uno de los cuales implementa un subDAC de 4 bits más su capacidad de acoplo asociada, y un sexto bloque que sintetiza la capacidad de calibración C_{cal}. En la siguiente figura se muestra la distribución de estos seis bloques y las partes del DAC que sintetiza cada uno de ellos.

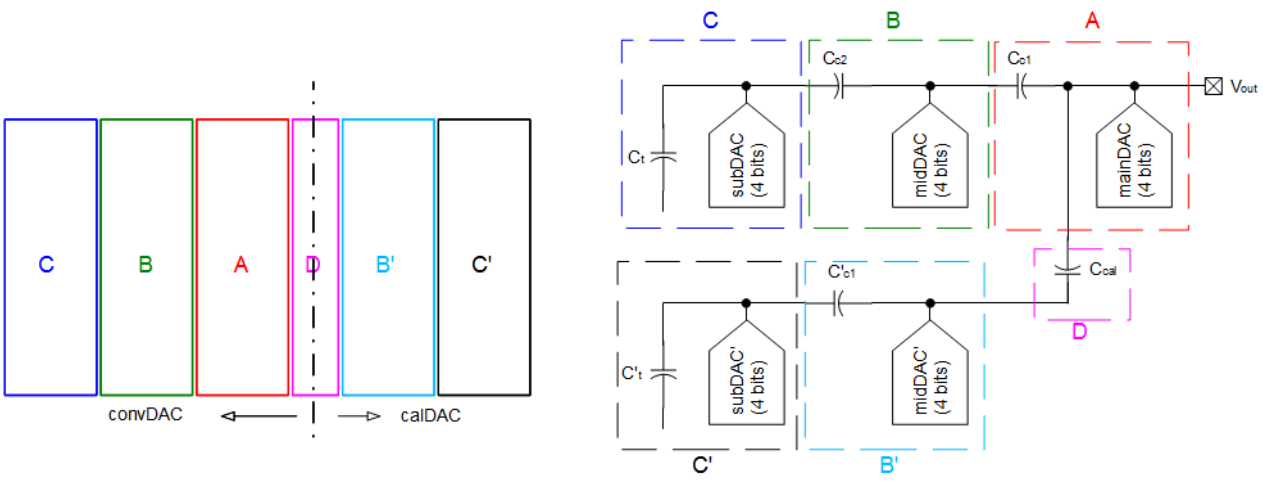


Figura 8.15: Estructura general del layout del DAC

Como se puede observar, se ha intentado situar cada uno de los seis bloques de tal manera que presenten una cierta simetría con respecto al **eje y,** con la finalidad de compensar al máximo el efecto del gradiente del oxido sin que el rutado de las capacidades se complique en exceso.

Por otro lado, cada uno de los cinco bloques que sintetizan un subDAC de 4 bits más su capacidad de acoplo, distribuyen sus capacidades de forma simétrica con respecto al **eje x**, intentando conseguir de esta manera un mejor *matching*. Por su parte, el bloque D, que implementa la capacidad de calibración C_{cal} de valor $C/6$, utiliza cinco capacidades unitarias conectadas en serie para sintetizarla, destacar que se conectan cinco y no seis capacidades en serie, pues sumando el efecto de las capacidades parásitas se consigue aproximadamente el valor de C_{cal} requerido. Además el C_{cal} no necesita estar *matcheado* con ninguna otra capacidad del DAC con lo que su valor no es crítico y puede permitirse una cierta tolerancia en él.

En cuanto al rutado se ha intentado hacer de la forma más sencilla posible, empleando metal 1 (MET1) para las conexiones de *bottom-plate* y el metal 3 (MET3) para el rutado de los *top-plates*. El estilo de rutado empelado permite conseguir que las capacidades parásitas asociadas a cada capacidad queden escaladas de forma binaria, manteniéndose de esta manera el ratio entre ellas, además se ha intentado rutar para que los parásitos de metal 3 (MET3) a sustrato (SUB) que puedan afectar a la linealidad del DAC sean los mínimos posibles.

Como últimas consideraciones sobre el *layout* añadir que:

- Se han empleado barreras de metal 1 – metal 2 – metal 3 para aislar cada capacidad unitaria de las de su alrededor y de esta forma eliminar posibles parásitos entre ellas que

pudieran influir de forma seria sobre la linealidad del DAC.

- Se ha utilizado un anillo de guarda de hasta metal 4 rodeando a todo el DAC, para mantener el circuito aislado del exterior.

- Se han añadido capacidades *dummies* en aquellas zonas del *layout* donde se han requerido, con la finalidad de mantener unas condiciones de contorno homogéneas en todas las capacidades unitarias.

- Las conexiones de las entradas digitales y la salida analógica se llevan al exterior por un mismo lado, para que la posterior conexión sea más sencilla y por lo tanto se requiera de un área de interconexionado menor.

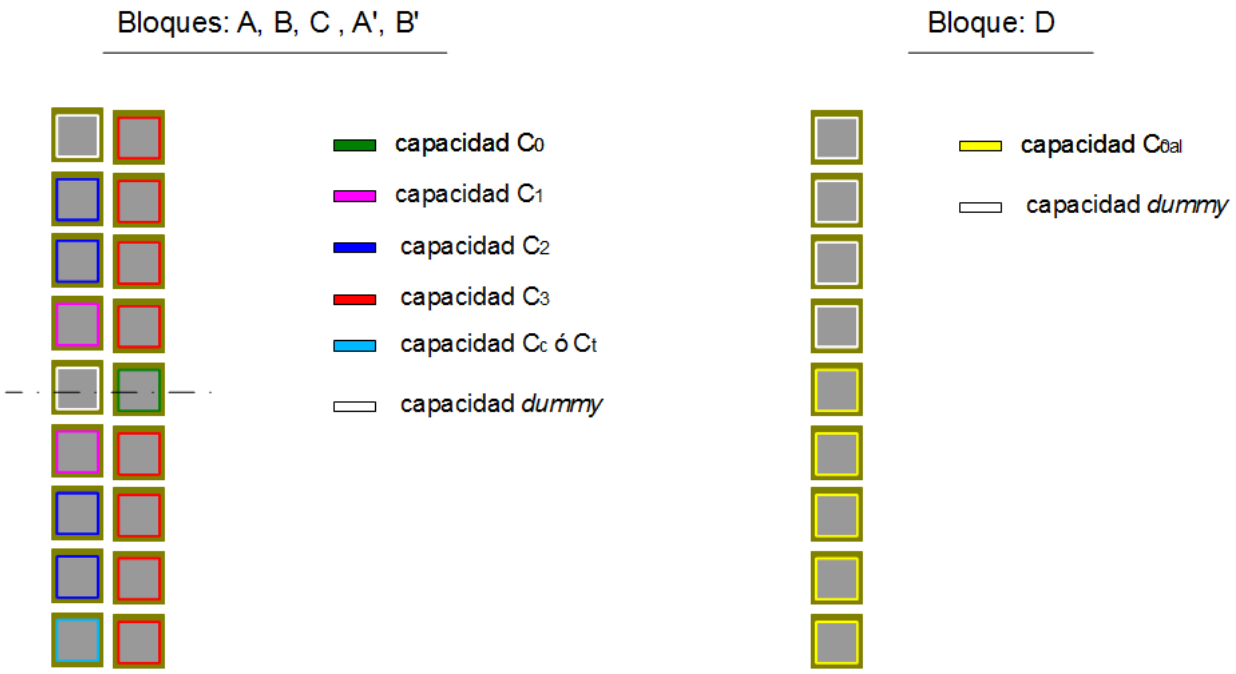


Figura 8.16: Distribución de las capacidades de cada subDAC y de la C_{cal}

Antes de concluir con la descripción del *layout* del DAC, añadir que en el anexo B de este documento se muestran diferentes capturas del *layout* implementado, resaltando los detalles y las características más relevantes del mismo.

8.3.4 Extracción. Capacidades parásitas

Después del diseño del *layout*, de su implementación con las reglas de diseño correspondientes y de haber superado la comprobación LVS, se procede a realizar la extracción del mismo. A partir de la extracción se obtienen las capacidades parásitas que se han sintetizado y se puede determinar el efecto de cada una de ellas sobre la linealidad del DAC.

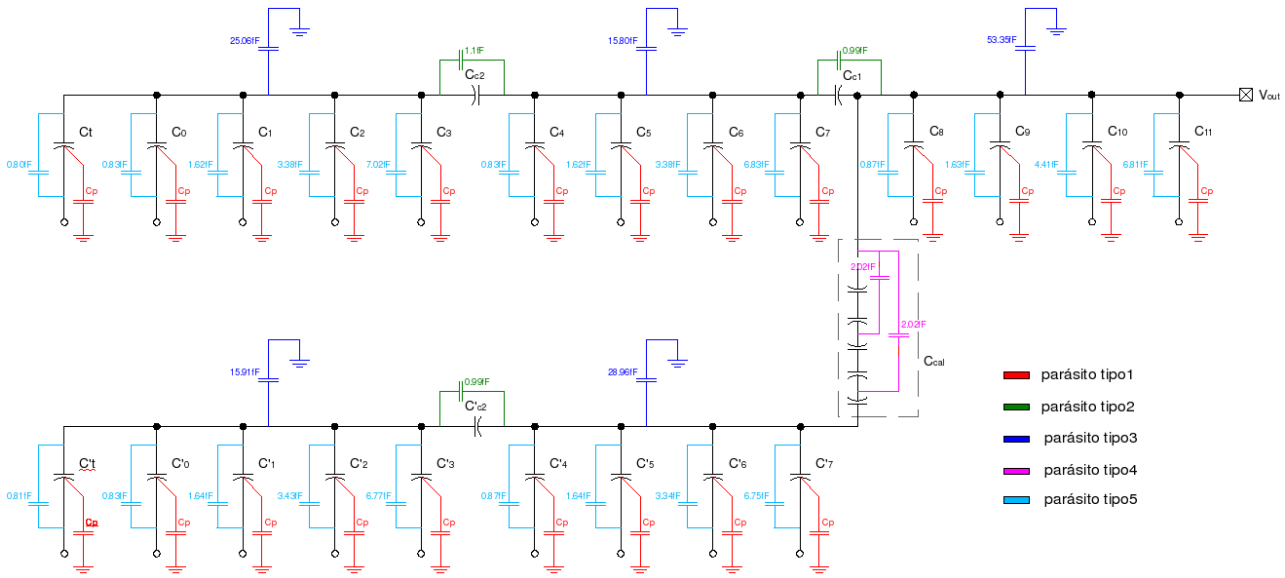


Figura 8.17: Esquemático del DAC con las capacidades parásitas asociadas

Parásitos de tipo 1: definidos por las capacidades parásitas entre *bottom-plate* y sustrato de cada capacidad. No afectan para nada a la linealidad del circuito, pues en ese nodo se va a forzar siempre una tensión V_{refn} ó V_{refp} .

Parásitos de tipo 2: definidos por las capacidades parásitas entre metal 3 y metal 2 generadas por el rutado entre las capacidades de acoplo y los subDAC's. Sí afectan a la linealidad del DAC, pues modifican el valor de las capacidades de acoplo con lo que la atenuación entre subDAC's se ve alterada.

Parásitos de tipo 3: definidos por dos contribuciones: las capacidades parásitas entre el *bottom-plate* y el sustrato de los C_c's y las generadas entre metal 3 y sustrato debidas al rutado de los *top-plates* de cada subDAC. Al igual que ocurre con los parásitos de tipo 2, la linealidad del DAC se ve afectada pues se modifica la atenuación entre subDAC's.

Parásitos de tipo 4: definidos por el rutado de la capacidad C_{cal} . No influyen sobre la linealidad,

simplemente modifican el valor de C_{cal}, el cual no es crítico en ningún caso.

Parásitos de tipo 5: definidos por las capacidades parásitas entre metal 3 y metal 2 debidas al rutado de los *top-plates* de cada subDAC. No afectan a la linealidad, pues estos parásitos están escalados de forma binaria con lo que el ratio entre capacidades se mantiene.

Como se observa en la figura, el número de capacidades parásitas es bastante elevado, sin embargo sólo las de tipo 2 y 3 van a afectar a la linealidad del circuito. Es de destacar que los parásitos asociados al calDAC van a influir sobre la linealidad de éste, pero no de forma considerable. Sin embargo, las capacidades parásitas del convDAC si que van a traducirse en errores de linealidad apreciables, los cuales van a tener que ser compensados por la calibración.

8.4 APLICACIÓN DE LA CALIBRACIÓN

En el capítulo siete de este documento se describe con detalle el funcionamiento del sistema de calibración a implementar. Su explicación parte de un DAC de conversión (convDAC) y un DAC de calibración (calDAC) genéricos (M y N bits respectivamente), y tras recorrer los pasos oportunos, obtiene las ecuaciones necesarias para el cálculo de los valores de calibración asociados a cada una de las N capacidades del convDAC:

$$V_{cal,N-1} = -V_{error,N-1} = \frac{(V_{res,N-1} - V_{OS})}{2} \qquad \textbf{(7.23)}$$

$$V_{cal,j} = -V_{error,j} = \frac{1}{2} \cdot \left[(V_{res,j} - V_{OS}) - \sum_{k=j+1}^{N-1} V_{cal,k} \right] \quad , j \in [N-2, 0] \qquad \textbf{(7.33)}$$

Como se deduce de los resultados obtenidos, un convDAC genérico con errores de *matching* en todas sus capacidades (ε_k), requerirá de la calibración de sus N capacidades. Una vez calculados los códigos de calibración para cada capacidad (véanse las expresiones **(7.23)** y **(7.33)**), durante el proceso de conversión se aplicará sobre el calDAC el código de calibración correspondiente a la suma de los códigos de calibración asociados a cada una de las N capacidades que se encuentren activas, como se indica en la ecuación:

$$código(V_{cal}) = \sum_{k=0}^{N-1} D_k \cdot código(V_{cal,k}) \qquad \textbf{(7.36)}$$

y de esta forma, los errores de linealidad generados por las imperfecciones de las capacidades serán compensados por el calDAC.

El caso descrito anteriormente se corresponde con un caso genérico, en el cual se ha supuesto que las N capacidades del convDAC van a sufrir errores de *matching* apreciables. En el caso concreto del DAC de este proyecto, que se ha implementado mediante una estructura capacitiva 4-4-4, se estima que con el *layout* elaborado se alcanzarán hasta 8 bits de *matching*, es decir, las ocho capacidades menos significativas del conDAC presentarán unas ε_k tan pequeñas que no generarán errores de linealidad apreciables. Sin embargo, las cuatro capacidades más significativas: C_{11}, C_{10}, C_9 y C_8, las cuales se corresponden con el mainDAC, sí producirán graves errores de linealidad sobre la respuesta final del DAC.

Teniendo en cuenta esto, el DAC diseñado no requeriría la calibración de sus 12 capacidades, sino que sería suficiente con calibrar sólo las capacidades de $C_{11} - C_8$. Teóricamente, esto sólo será cierto si se consideraran despreciables las capacidades parásitas asociadas al circuito, lo cual no va a ocurrir en la realidad, con lo que finalmente se ha tenido que llegar a un compromiso, mediante el cual se consiguen compensar los errores de linealidad generados por las capacidades parásitas y el *matching*, calibrando sólo las 8 capacidades MSB del DAC.

A continuación se presentan las ecuaciones, para el cálculo de los valores de calibración asociados a cada una de las capacidades del DAC:

$$\begin{cases} V_{cal,11} = -V_{error,11} = \dfrac{(V_{res,11} - V_{OS})}{2} \\ V_{cal,j} = -V_{error,j} = \dfrac{1}{2} \cdot \left[(V_{res,j} - V_{OS}) - \sum_{k=j+1}^{11} V_{cal,k} \right] \quad , j \in [10,4] \\ V_{cal,j} = -V_{error,j} = -\dfrac{\sum_{k=j+1}^{11} V_{cal,k}}{2} \quad , j \in [3,0] \end{cases} \tag{8.35}$$

CAPÍTULO 9

Lógica de control

9.1 ESPECIFICACIONES

Para gobernar el funcionamiento de cada uno de los bloques del SAR ADC y ejecutar la secuencia de pasos necesarios para realizar la búsqueda SAR, se requiere de la utilización de un bloque digital de control. Además, en este caso en el que se emplea un sistema de calibración, al bloque de control también se le atribuyen otras tareas relacionadas con el proceso de calibración del DAC.

Fundamentalmente, la lógica de control va a ser implementada mediante dos máquinas de estados: la primera se encargará de ejecutar el algoritmo de búsqueda SAR y de aplicar los códigos de calibración del calDAC requeridos en cada momento (fase de conversión), y la segunda será la encargada de ejecutar la fase de calibración, en la que se necesita una secuencia de pasos específica para medir los errores de linealidad del DAC y enviarlos al exterior para que se calculen los códigos de calibración asociados a cada capacidad.

A continuación se describen todas las líneas de entrada/salida requeridas para el control del sistema y la funcionalidad de cada una de ellas:

Entradas:

`clk` : reloj del sistema. Mediante esta entrada se le pasa a la lógica de control el reloj requerido para las máquinas de estados.

por_n : *power or reset* (activo a nivel bajo). Entrada asíncrona que deshabilita las máquinas de estado, pasando a un estado de espera.

start : inicio de conversión. Entrada síncrona que indica el inicio de la conversión, abandonando el estado de espera.

cal_en : habilitación de calibración. Entrada síncrona que determina el modo de funcionamiento: conversión (0) ó calibración (1), ejecutándose en cada caso la máquina de estados correspondiente.

cal_reg_sel<2:0> : selección del registro de calibración. Entrada síncrona que determina la capacidad a calibrar cuando se está en modo de calibración (`cal_en` = 1)

cal_reg_i<7:0> : registro de calibración i (i = 0,…,11). Entradas de 8 bits para leer la memoria externa donde se almacenan los códigos de calibración de cada capacidad.

comp : resultado de comparación. Entrada síncrona que indica el resultado de la última comparación: DACp > DACn (1) , DACn < DACp (0)

cal_reg0<7:0>	código calibración C0
cal_reg1<7:0>	código calibración C0
cal_reg2<7:0>	código calibración C0
cal_reg3<7:0>	código calibración C0
cal_reg4<7:0>	código calibración C0
cal_reg5<7:0>	código calibración C0
cal_reg6<7:0>	código calibración C0
cal_reg7<7:0>	código calibración C0
cal_reg8<7:0>	código calibración C0
cal_reg9<7:0>	código calibración C0
cal_reg10<7:0>	código calibración C0
cal_reg11<7:0>	código calibración C0

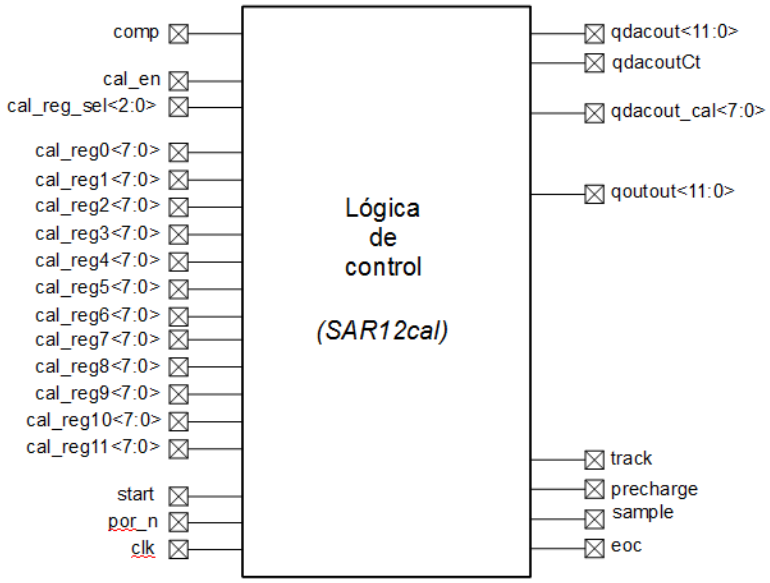


Figura 9.1: Líneas de entrada/salida del bloque de control

Salidas:

precharge : precarga. Salida síncrona que indica el estado de los *switches* de precarga: *switches_precarga* ON (1) , *switches_pregarga* OFF (0).

track : seguimiento. Salida síncrona que indica si el DAC está en modo seguimiento (1) ó en modo conversión (0).

Sample : muestreo. Salida síncrona que indica el estado del *switch* de muestreo: *switch_muestreo* ON (1) , *switch_muestreo* OFF (0).

qdacout<11:0> : código de búsqueda. Salida síncrona que indica el código de búsqueda de cada iteración.

qdacoutCt : capacidad terminal. Salida síncrona que indica el valor (0 ó 1) a aplicar sobre la capacidad terminal.

qoutout_cal<7:0> : código de calibración. Salida síncrona que indica el código de calibración asociado a cada código de búsqueda.

qoutout<11:0> : resultado de conversión. Salida síncrona que indica el resultado de la conversión. Salida valida sólo cuando `eoc` = 1.

9.2 MÁQUINA DE ESTADOS

Las máquinas de estados a implementar en el bloque digital de control van a ser muy similares en cuanto a estados y funcionamiento, pues fundamentalmente ambas máquinas ejecutan un algoritmo SAR. La diferencia principal entre ambas reside en que en una el algoritmo SAR se aplica sobre el calDAC y en la otra sobre el convDAC.

La máquina de estados que se emplea durante la fase de conversión (FSMconv) aplica la búsqueda SAR sobre el DAC de conversión, para encontrar el código digital de 12 bits que representa a la tensión analógica de entrada. Además de esto, también actúa sobre el calDAC, calculando y aplicando sobre éste, el código de calibración requerido para compensar los errores de linealidad del convDAC en cada código.

Por otra parte, la máquina de estados de la fase de calibración (FSMcal), aprovecha el algoritmo SAR sobre el calDAC, para una vez muestreada la tensión de error asociada a una determinada capacidad del convDAC, buscar el código del calDAC que la representa. La FSMcal se ejecutará una vez por capacidad del convDAC a calibrar.

A continuación, se muestra una figura con los diagramas de estados de cada una de las FSM, y se describen con detalle el comportamiento de las señales y las operaciones realizadas en cada uno de los estados.

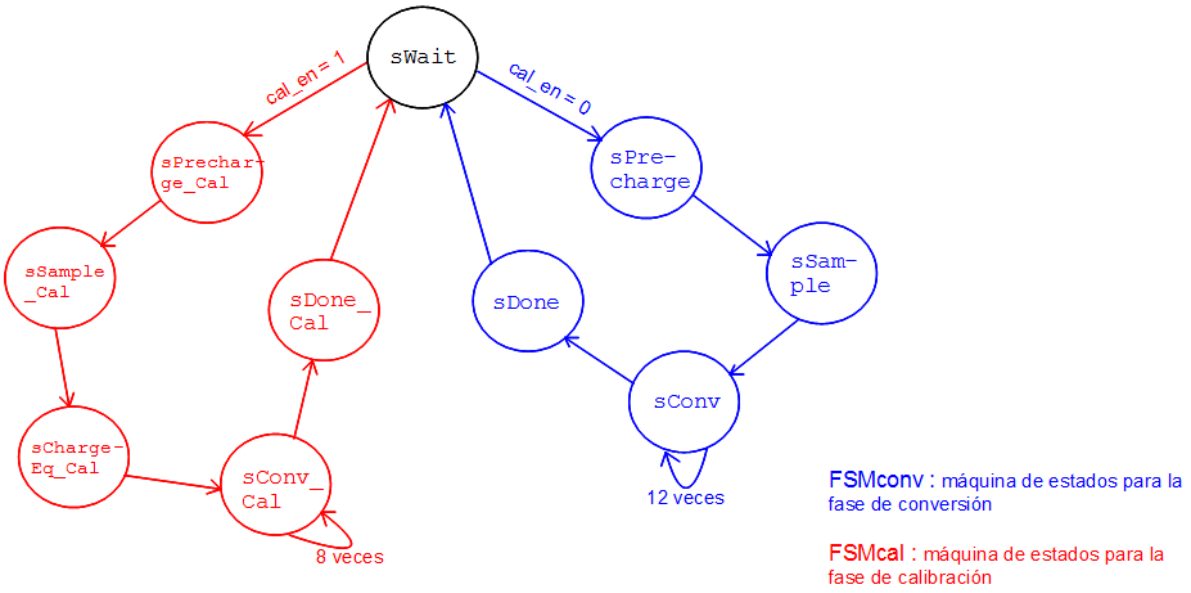


Figura 9.2: Diagrama de estados de las FSM´s

Descripción de los estados:

sWait : estado de espera. La máquina de estados espera a que se active la señal `start` (`start` = 1) para ponerse en funcionamiento. En función de tener `cal_en` = 0 ó `cal_en` = 1 se ejecutará la FSMconv ó la FSMcal, respectivamente.

FSMconv:

sPrecharge : estado de precarga. En este estado se activan las señales `precharge` y `track`, con lo que las salidas de los DAC's se fijan a V_{refn} y V_{refp} y los *bottom-plates* de las capacidades se conectan a V_{in}. Además, la señal `qdacout_cal<7:0>` = 10…0 forzando el código de calibración neutro.

sSample : estado de muestreo. En este estado se desactiva la señal `precharge` y se activa la señal `sample`, quedando las señales `track` y `qdacout_cal<7:0>` como en el estado anterior.

sConv : estado de conversión. En este estado la señales `sample` y `track` se desactivan, muestreando de esta manera la V_{in}. Al mismo tiempo la señal `qdacout<11:0>` = 1,…0 y la `qdacout_cal<7:0>` es igual al código de calibración del bit11.

El estado `sConv` se repite 12 veces; aplicando el algoritmo de búsqueda SAR sobre `qdacout<11:0>` y calculando el código de calibración a aplicar en `qdacout_cal<7:0>` en cada caso. El proceso realizado en cada una de las iteraciones es el siguiente:

1- Se pone a 1 el nuevo bit de `qdacout<11:0>` a buscar.

2- Se comprueba el valor de `comp`

· Si `comp` = 1

El bit de `qdacout<11:0>` anterior al de búsqueda se resetea a 0

· Si `comp` = 0

El bit de `qdacout<11:0>` anterior al de búsqueda se deja a 1

3- Se suman los códigos de calibración de todos los bits activos de qdacout<11:0>, obteniéndose así el código a aplicar en qdacout_cal<7:0>

(el proceso se repite para cada uno de los bits de qdacout<11:0>, empezando por la búsqueda del bit MSB y terminando por la búsqueda del bit LSB)

sDone : estado de fin de búsqueda. En este estado se activa la señale eoc, para indicar el fin de la conversión y se pasa el valor actual de qdacout<11:0> a qoutout<11:0>.

FSMcal:

sPrecharge_Cal : estado de precarga. En este estado se activa la señal precharge y se fija el código qdacout_cal<7:0> = 10…0. Por otro lado el valor de qdacout<11:0> no es relevante en este estado.

sSample_Cal : estado de muestreo. En este estado se desactiva la señal precharge y se activa la señal sample. El valor de qdacout_cal<7:0> se mantiene del estado anterior, mientras que la línea qdacoutCt se activa y el valor de qdacout<11:0> se fija en función del error de capacidad que se quiere medir (cal_reg_sel<2:0>).

Por ejemplo, si la señal cal_reg_sel<2:0> indica que se quiere medir el error de la capacidad C_{10}, entonces: qdacout<11:0> = 001111111111.

sChargeEq_Cal : estado de muestreo (II). Se mantienen todas las señales exactamente igual que en el estado anterior, a excepción de sample que pasa a valer 0. Con esta acción, se muestrea la tensión del convDAC generada por el código qdacout<11:0> junto a qdacoutCt que se está aplicando.

sConv_Cal : estado de conversión. En este estado la señales sample y precharge permanecen desactivadas, mientras que la señal qdacoutCt se fija a 0 y el valor de qdacout<11:0> cambia de nuevo en función de la capacidad que se está calibrando (cal_reg_sel<2:0>).

Siguiendo con el ejemplo anterior, si la señal cal_reg_sel<2:0> indica que se quiere medir el error de la capacidad C_{10}, entonces: qdacout<11:0> = 010000000000.

En estas condiciones se aplica el algoritmo de búsqueda SAR sobre el calDAC, repitiéndose el

estado `sConv_Cal` 8 veces hasta encontrar la tensión de error de la capacidad que se está midiendo. El proceso SAR se ejecuta de la misma forma que en la FSMconv, pero ahora con la señal `qdacout_cal<7:0>`:

1- Se pone a 1 el nuevo bit de `qdacout_cal<7:0>` a buscar.

2- Se comprueba el valor de `comp`

· Si `comp` = 1

El bit de `qdacout_cal<7:0>` anterior al de búsqueda se resetea a 0

· Si `comp` = 0

El bit de `qdacout_cal<7:0>` anterior al de búsqueda se deja a 1

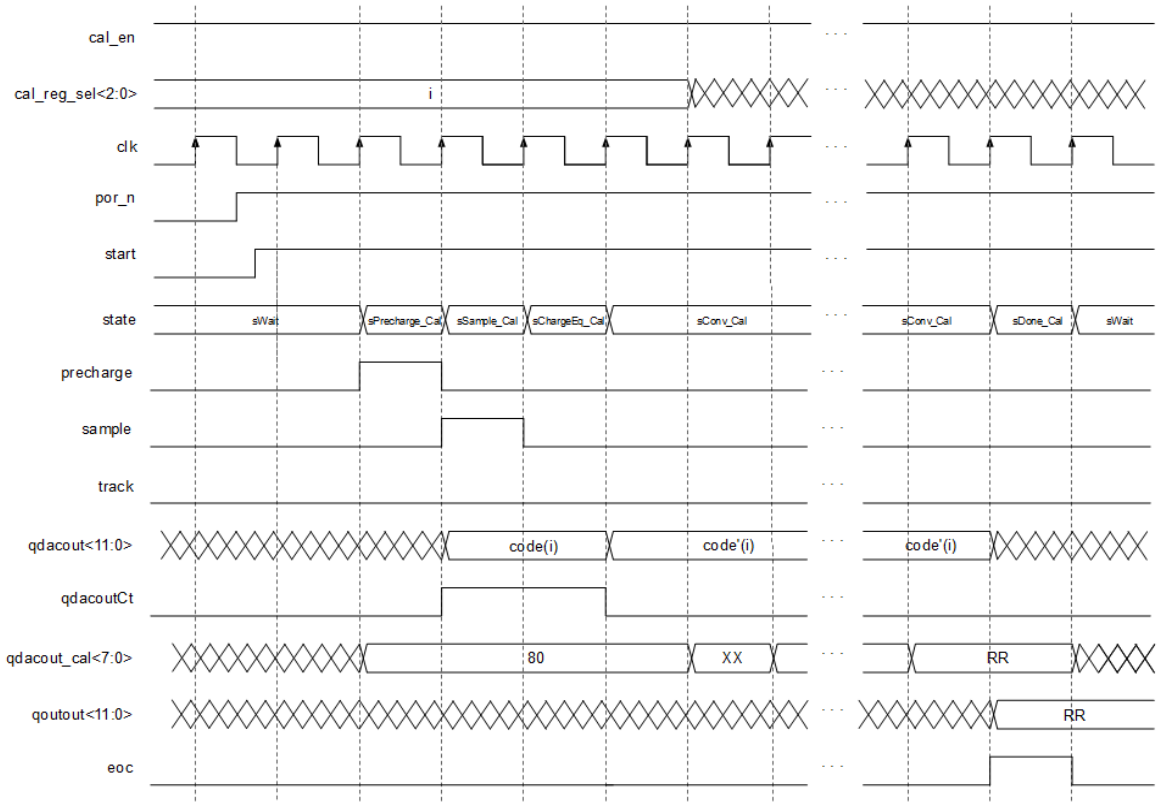
(el proceso se repite para cada uno de los bits de `qdacout_cal<7:0>`, empezando por la búsqueda del bit MSB y terminando por la búsqueda del bit LSB)

`sDone` : estado de fin de búsqueda. En este estado se activa la señale `eoc`, para indicar el fin de la búsqueda, y se pasa el valor actual de `qdacout_cal<7:0>` a los 8 bits menos significativos de `qoutout<11:0>`.

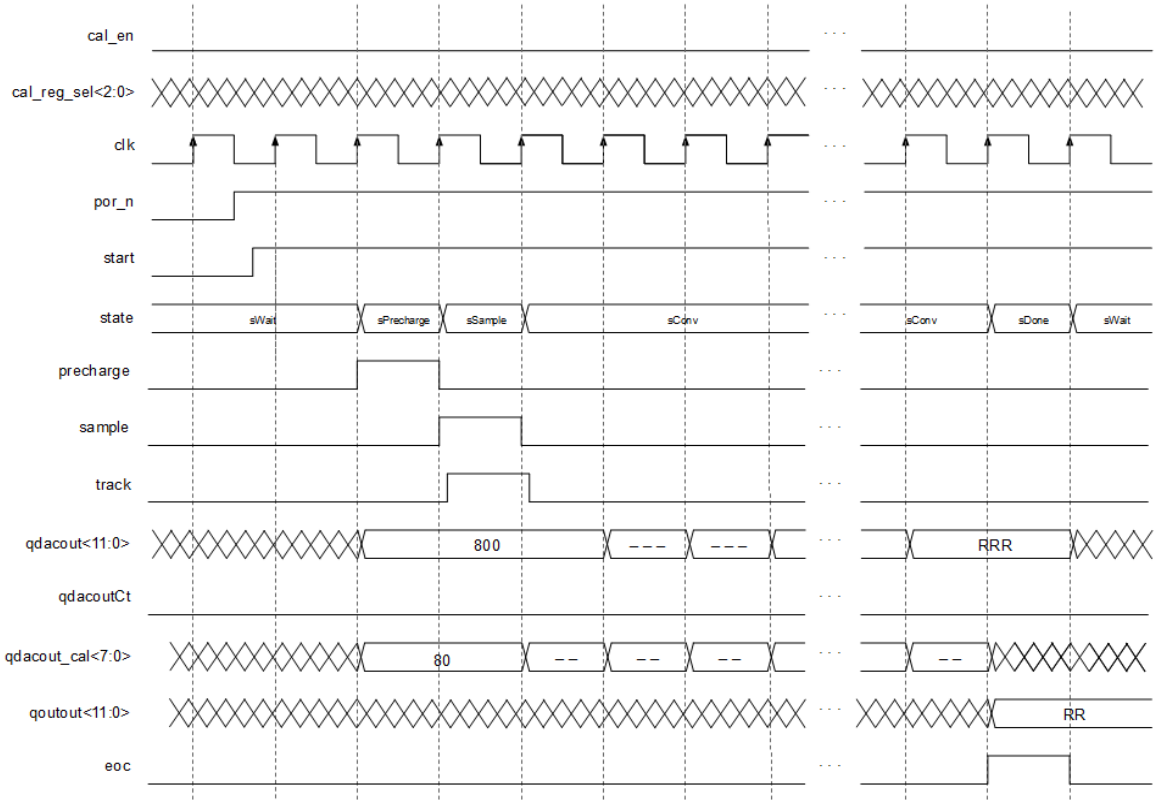
Observando los diagramas de estado de ambas máquinas es de destacar el estado adicional que presenta la FSMcal con respecto a la FSMconv, a pesar de que como se ha visto, las tareas realizadas en ambas máquinas son básicamente las mismas: precarga, muestreo, conversión y fin de búsqueda. La razón de incluir el estado adicional (`sChargeEq_Cal`) en la máquina de estados de calibración es el poder realizar de forma correcta el muestreo de la tensión de error. En la máquina FSMconv este muestreo no se realiza pues simplemente se muestrea la V_{in}, por lo que no se requiere de este estado adicional.

Para finalizar con la descripción de la lógica de control, se muestra a continuación un cronograma en donde se representa la secuencia de funcionamiento de cada máquina, con sus estados y las señales activas en cada momento.

FSMcal



FSMconv



9.3 DESCRIPCIÓN VERILOG Y SÍNTESIS

Las máquinas de estados definidas anteriormente, además de la lógica de control (*controlpath*), van a requerir de una lógica adicional (*datapath*) para realizar las labores de: cálculo de los códigos de calibración para cada valor de búsqueda, y ejecución del algoritmo SAR.

La parte del *datapath* va a ser la más crítica a la hora de hacer la descripción *Verilog* de la lógica digital, a continuación se explica la topología empleada para su implementación.

Para la implementación del algoritmo SAR, se requiere utilizar una estrategia lo más sencilla posible, para reducir al máximo la lógica y en consecuencia el tamaño final del bloque. Pues no hay que olvidar que, en realidad se van a necesitar dos algoritmos SAR, uno para el convDAC (en la FSMconv) y otro para el calDAC (en la FSMcal).

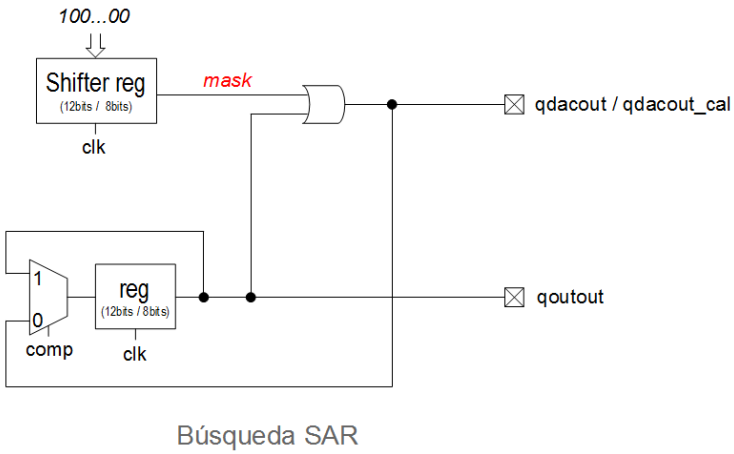


Figura 9.3: Diagrama de bloques de la implementación del algoritmo SAR

La búsqueda SAR se ha implementado como se muestra en la figura. El código de búsqueda para cada iteración se obtiene a partir de la OR entre dos señales: `qoutout` y `mask`. La señal `mask`, define una máscara que va cambiando en cada iteración gracias a un registro de desplazamiento, e indica en todo momento el bit que se está buscando. Por su parte, la señal `qoutout` se corresponde con la última palabra de búsqueda aplicada, actualizada con el bit encontrado. Al realizar la OR entre estas dos señales se tiene como resultado una palabra que contiene todos los bits encontrados hasta el momento y el próximo bit a buscar activo a 1, es decir, se tiene la siguiente palabra de búsqueda.

Como ejemplo, en la siguiente tabla se muestra la evolución de cada una de las señales para un

proceso SAR de 3 bits.

Iter. (k)	Salida del comparador (k-1)	qdacout / qdacout_cal (k-1)	reg (k-1)	reg (k)	qoutout (k)	mask (k)	qdacout / qdacout_cal (k)
1	X	XXX	XXX	000	000	100	100
2	0	100	000	100	100	010	110
3	1	110	100	100	100	001	101
4	0	101	100	101	101	000	101 (result.)

Tabla 9.1: Ejemplo de funcionamiento de un proceso SAR de 3 bits

En cuanto al cálculo de los códigos de calibración para cada palabra que se aplica sobre el convDAC, se ha diseñado un proceso mediante el cual, a partir del código de calibración de la palabra de búsqueda anterior se obtiene el código de calibración para la nueva palabra de búsqueda. De esta forma, con un simple sistema basado en un sumador y un registro (*calibration reg*) se consiguen realizar estos cálculos.

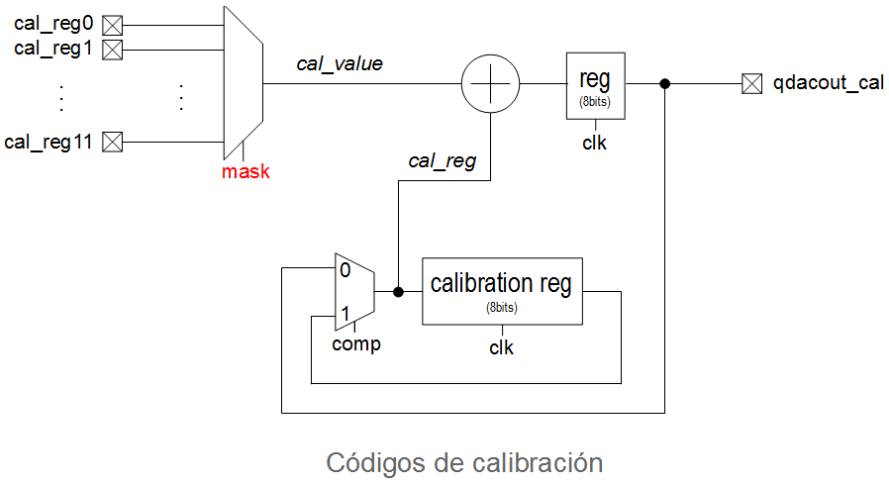


Figura 9.4: Diagrama de bloques de la implementación del cálculo de los códigos de calibración

El procedimiento seguido es el siguiente: para cada iteración, el código de calibración a aplicar se calcula como la suma de los valores: `cal_value` y `cal_reg`. El valor `cal_value` se corresponde con el código de calibración asociado al bit que se está buscando en esa iteración y `cal_reg` puede tomar valores distintos en función del bit encontrado en la iteración anterior, es decir:

Para la iteración k:

· Si `comp` = 0 (último bit encontrado 1)

`cal_reg` = valor de calibración anterior (`qdacout_cal` en k-1)

· Si `comp` = 1 (último bit encontrado 0)

`cal_reg` = valor de *calibration reg* (`qdacout_cal` en k-2)

Siguiendo con el ejemplo anterior de proceso SAR de 3 bits, a continuación se muestra la secuencia seguida para la obtención de los códigos de calibración asociados a cada palabra de búsqueda:

Iter. (k)	mask (k)	Salida del comparador (k-1)	qdacout (k)	qdacout_cal (k-1)	calibration reg (k-1)	cal_reg (k)	cal_value (k)	qdacout_cal (k)
1	100	X	100	X	X	0	cal_bit2	cal_bit2
2	010	0	110	cal_bit2	0	cal_bit2	cal_bit1	cal_bit2 + cal_bit1
3	001	1	101	cal_bit2 + cal_bit1	cal_bit2	cal_bit2	cal_bit0	cal_bit2 + cal_bit0
4	000	0	101 (result.)	cal_bit2 + cal_bit0	cal_bit2	cal_bit2 + cal_bit0	0	cal_bit2 + cal_bit0

Tabla 9.2: Ejemplo de funcionamiento del cálculo de los códigos de calibración para un SAR de 3 bits

Recordando que, el código de calibración necesario para compensar el error de linealidad generado por una cierta palabra de búsqueda, es igual a la suma de los códigos de calibración asociados a cada una de las capacidades del convDAC activas, es decir:

$$code\_cal(D_{N-1}...D_0) = \sum_{k=0}^{k=N-1} D_k \cdot cal\_bit_k \qquad \textbf{(9.1)}$$

,donde cal_bit_k es el código de calibración asociado a la capacidad C_k

se puede observar con el ejemplo anterior, que siguiendo el procedimiento planteado, los códigos de calibración calculados son los esperados.

Para concluir este capítulo, se presenta el código *Verilog* que se ha escrito para implementar las máquinas de estado requeridas. Además se muestra un diagrama de bloques que representa, a

grandes rasgos, la lógica sintetizada. Destacar también que, tras optimizar las opciones del sintetizador para conseguir un área mínima, se ha obtenido como resultado un área de 0.0056mm^2, a falta del rutado (el cual supone un incremento de un 10-15% del área).

```
//Verilog HDL for "SIM_E_EBU", "SAR12cal" "functional"


`define NBITS_SAR 12   //numero de bits del SAR
`define NBITS_CAL 8   //numero de bits de calibracion


module SAR12cal ( clk, por_n, start, cal_en, cal_reg_sel, cal_reg0
,cal_reg1, cal_reg2, cal_reg3, cal_reg4, cal_reg5, cal_reg6, cal_reg7,
cal_reg8, cal_reg9,cal_reg10, cal_reg11, comp, eocout, track, precharge,
precharge_n, qdacout, qdacout_cal, qoutout, qdacoutCt, sample, sample_n );

  //puertos in/out
  input clk;
  input por_n;
  input start;
  input cal_en;
  input [2:0] cal_reg_sel;
  input signed [`NBITS_CAL-1:0] cal_reg0, cal_reg1, cal_reg2, cal_reg3,
cal_reg4, cal_reg5, cal_reg6, cal_reg7, cal_reg8, cal_reg9, cal_reg10,
cal_reg11;
  input comp;

  output reg eocout;
  output reg track;
  output reg precharge;
  output reg precharge_n;
  output reg sample;
  output reg sample_n;

  output [`NBITS_SAR-1:0] qdacout;
  output reg qdacoutCt;
  output [`NBITS_CAL-1:0] qdacout_cal;
  output reg [`NBITS_SAR-1:0] qoutout;

  //parametros
  parameter sWait = 4'b0000;
  parameter sPrecharge = 4'b0100, sSample = 4'b0101, sConv = 4'b0110, sDone
= 4'b0111;
  parameter sPrecharge_Cal = 4'b1000, sSample_Cal = 4'b1001, sChargeEq_Cal
= 4'b1010, sConv_Cal = 4'b1011, sDone_Cal = 4'b1100;

  //variables internas
  reg [3:0] state;    //almacena el estado actual
  reg [`NBITS_CAL-1:0] calibration_reg;   //registro acumulador de la
calibracion

  reg [`NBITS_SAR-1:0] mask;    //mascara para busqueda SAR del convDAC
  reg [`NBITS_CAL-1:0]  mask_cal;     //mascara para busqueda SAR del
calDAC
```

```
reg [`NBITS_CAL-1:0] qoutout_cal;   //variable busqueda SAR del calDAC

reg [`NBITS_CAL-1:0] cal_bit;   //salida del MUX
reg [`NBITS_CAL-1:0] cal_reg;   //salida del MUX


//MAQUINA DE ESTADOS
always @(posedge clk or negedge por_n) begin

  if (!por_n) begin
    state <= sWait;

    mask <= 12'h000;
    qoutout <= 12'h000;
    mask_cal <= 8'h80;
    qoutout_cal <= 8'h00;

  end

  else case (state)    //paso al siguiente estado

    ///ESTADO DE ESPERA (y PRECARGA)
    sWait :
      begin
        mask <= 12'h000;
        qoutout <= 12'h000;
        mask_cal <= 8'h80;
        qoutout_cal <= 8'h00;

        //pasa al siguiente estado
        if (start && !cal_en)   state <= sPrecharge;  //CONV
        else if (start && cal_en)  state <= sPrecharge_Cal;  //BUSQ CAL
        else  state <= sWait;  //IDLE

      end


    /// CONVERSION ////////////////////////////////////////////////////
    ///Estado de precarga
    sPrecharge :
      begin
        //siguiente estado
        state <= sSample;

      end


    ///Estado de muestreo
    sSample :
      begin
        //siguiente estado
        state <= sConv;

        //actualizacion de salidas
        mask <= 12'h800;
        mask_cal <= cal_bit + cal_reg;   //sumador registrado!

      end
```

```
///Estado de conversion
sConv :
  begin

     //busqueda SAR convDAC
     if (!comp) qoutout <= qoutout | mask;

     if (mask[0] == 1'b0) begin
        mask <= mask >> 1;   //siguiente 'bit activo'
     end
     else begin
        state <= sDone;  //siguiente estado
     end

     //calibracion calDAC
     mask_cal <= cal_bit + cal_reg;   //sumador registrado!

  end


///Estado de fin de conversion
sDone :
  begin
    //siguiente estado
    state <= sWait;

  end


/// BUSQ. CALIBRACION ////////////////////////////////////////////
///Estado de precarga
sPrecharge_Cal :
  begin
    //siguiente estado
    state <= sSample_Cal;

    //actualizacion de salidas
      case(cal_reg_sel)

        3'd7:  mask <= 12'h7ff;  //dacout <= 0111_1111_1111

        3'd6:  mask <= 12'h3ff;  //dacout <= 0011_1111_1111

        3'd5:  mask <= 12'h1ff; //dacout <= 0001_1111_1111

        3'd4:  mask <= 12'h0ff;  //dacout <= 0000_1111_1111

        3'd3:  mask <= 12'h07f;  //dacout <= 0000_0111_1111

        3'd2:  mask <= 12'h03f;  //dacout <= 0000_0011_1111

        3'd1:  mask <= 12'h01f; //dacout <= 0000_0001_1111

        3'd0:  mask <= 12'h00f;  //dacout <= 0000_0000_1111

      endcase

  end
```

```
///Estado de muestreo
sSample_Cal :
  begin
    //siguiente estado
    state <= sChargeEq_Cal;

  end


///Estado de muestreo2
sChargeEq_Cal :
  begin
    //siguiente estado
    state <= sConv_Cal;

    //actualizacoin de salidas
    case(cal_reg_sel)

      3'd7:  mask <= 12'h800;  //dacout <= 1000_0000_0000

      3'd6:  mask <= 12'h400;  //dacout <= 0100_0000_0000

      3'd5:  mask <= 12'h200; //dacout <= 0010_0000_0000

      3'd4:  mask <= 12'h100;  //dacout <= 0001_0000_0000

      3'd3:  mask <= 12'h080;  //dacout <= 0000_1000_0000

      3'd2:  mask <= 12'h040;  //dacout <= 0000_0100_0000

      3'd1:  mask <= 12'h020; //dacout <= 0000_0010_0000

      3'd0:  mask <= 12'h010;  //dacout <= 0000_0001_0000

    endcase

  end


///Estado de busqueda
sConv_Cal :
  begin
     //busqueda SAR calDAC
     if (!comp)  qoutout_cal <= qoutout_cal | mask_cal;

     if (mask_cal[0] == 1'b0) begin
        mask_cal <= mask_cal >> 1;   //siguiente 'bit activo'
     end
     else begin
        state <= sDone_Cal;  //siguiente estado
        qoutout[7:0] <= qoutout_cal; //valor encontrado
     end

  end


///Estado de almacenamiento
sDone_Cal :
  begin
```

```
          //siguiente estado
          state <= sWait;

        end


      ///Estado por defecto (reseta la FSM)
      default:
        begin
          state <= sWait;

          mask <= 12'h000;
          qoutout <= 12'h000;
          mask_cal <= 8'h80;
          qoutout_cal <= 8'h00;

        end

    endcase

  end


  //MUX: cal_bit
  always @(cal_reg11 or cal_reg10 or cal_reg9 or cal_reg8 or cal_reg7 or
cal_reg6 or cal_reg5 or cal_reg4 or cal_reg3 or cal_reg2 or cal_reg1 or
cal_reg0 or mask) begin

    case(mask)

      12'd2048: cal_bit = cal_reg10;

      12'd1024: cal_bit = cal_reg9;

      12'd512:  cal_bit = cal_reg8;

      12'd256:  cal_bit = cal_reg7;

      12'd128:  cal_bit = cal_reg6;

      12'd64:   cal_bit = cal_reg5;

      12'd32:   cal_bit = cal_reg4;

      12'd16:   cal_bit = cal_reg3;

      12'd8:   cal_bit = cal_reg2;

      12'd4:   cal_bit = cal_reg1;

      12'd2:   cal_bit = cal_reg0;

      12'd0:    cal_bit = cal_reg11;

      default: cal_bit = 8'h00;

    endcase

  end
```

```
//MUX: cal_reg
always @(calibration_reg or mask_cal or comp) begin

  case(comp)

    1'd0: cal_reg = mask_cal;

    1'd1: cal_reg = calibration_reg;

  endcase

end


//REG: calibration_reg
always @(posedge clk or negedge por_n) begin

  if (!por_n) calibration_reg <= 8'h80;

  else case(state)

    sWait:  calibration_reg <= 8'h80;

    default:  calibration_reg <= cal_reg;

  endcase

end


//SALIDAS COMBINACIONALES (registradas para evitar glitches!)
always @(posedge clk or negedge por_n) begin

  if (!por_n) begin

    precharge <= 1'b1;
    precharge_n <= 1'b0;
    sample <= 1'b0;
    sample_n <= 1'b1;
    track <= 1'b0;
    eocout <= 1'b0;

    qdacoutCt <= 1'b0;

  end

  else begin

    precharge <= (state==sWait) | (state==sDone) | (state==sDone_Cal);
    precharge_n <= !precharge;
    sample <= (state==sPrecharge) | (state==sPrecharge_Cal);
    sample_n <= !sample;
    track <= (state==sPrecharge);
    eocout <= (state==sDone) | (state==sDone_Cal);

    qdacoutCt <= (state==sPrecharge_Cal) | (state==sSample_Cal);

  end

end
```

```
  assign qdacout = qoutout | mask;
  assign qdacout_cal = qoutout_cal | mask_cal;

endmodule
```

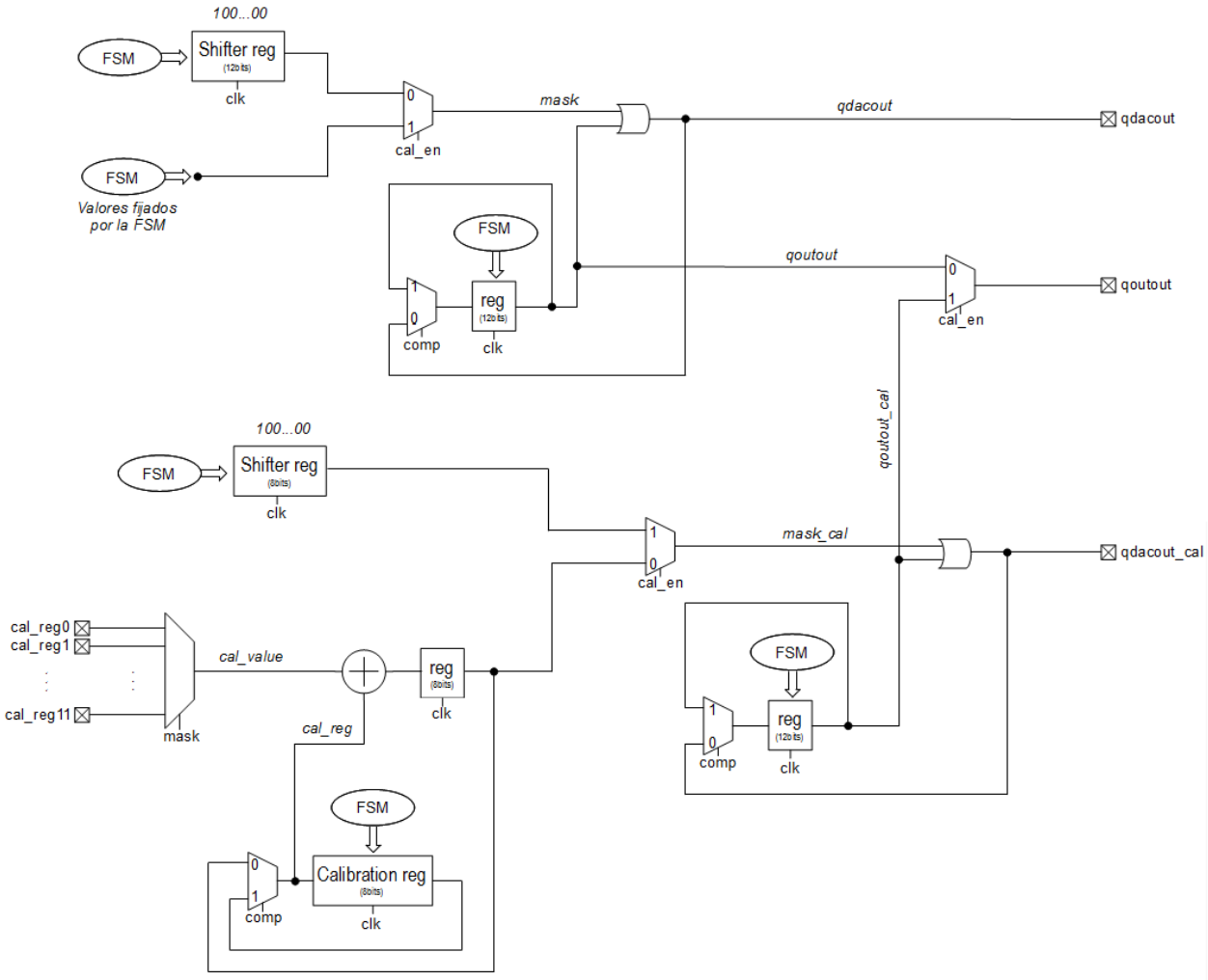


Figura 9.5: Diagrama de bloques de la lógica sintetizada

CAPÍTULO 10

Comparador

10.1 ESPECIFICACIONES

En el capítulo quinto, al presentarse las especificaciones generales para el diseño de este ADC y analizarse en que medida contribuían a ellas cada uno de los bloques, se llegó a la conclusión de que el comparador iba a ser el bloque crítico en cuanto al consumo, pues la lógica digital y los DAC's presentan un consumo despreciable frente al requerido por el comparador. Por lo tanto, recordando que el presupuesto de consumo para este diseño es de unos 30µA, el comparador a diseñar adoptará esta especificación y su consumo deberá estar en torno a esa cifra.

Una de las especificaciones más limitantes a la hora del diseño del comparador va a ser la del bajo consumo, pero además de ésta, la velocidad va a jugar también un papel fundamental. De toda la cadena del ADC: lógica digital, DAC's y comparador, éste último va a ser el bloque más lento y en consecuencia, el que va a limitar el *throughput* máximo que se va a alcanzar. Recordemos que para este ADC se impuso uno *throughput* mínimo de unos 10kSPS, por lo que el comparador, en el peor de los casos, deberá ser capaz de realizar una comparación en:

$$1 \ \textit{conversión} = 16 \ \textit{ciclos de clk} \quad \textbf{(10.1)}$$

$$(4 \ \textit{ciclos pregarga/muestreo}, \ 12 \ \textit{ciclos conversión})$$

$$10kSPS = 10000\frac{conversiones}{segundo} = \left(16\frac{ciclos}{conversión}\right)\cdot\left(10000\frac{conversiones}{segundo}\right) = \\ = \left(160000\frac{ciclos}{segundo}\right) = 16kHz \quad \rightarrow \quad T = 6.25\mu s \tag{10.2}$$

(en el peor de los casos, el comparador va a tener que resolver una comparación en 6.25us)

de los cálculos anteriores se deduce la frecuencia mínima del reloj del sistema; que va a ser de 160kHz, para conseguir los 10kSPS. Para tener un poco de margen y no diseñar en el límite de la especificación, se va a definir una frecuencia para el reloj de 166kHz ($T_{clk} \approx 6\mu s$). Por lo tanto, con este reloj, el comparador va a tener que poder comparar las señales a su entrada en un tiempo de 6µs (caso peor).

Como última especificación a destacar, hay que señalar la resolución, es decir, la mínima diferencia entre las señales de entrada que el comparador es capaz de discernir. Haciendo un análisis general es obvio que, como mínimo, el comparador debe ser capaz de distinguir una diferencia de tensión de 1LSB de los DAC's, o lo que es lo mismo:

$$\mathrm{V}_{resol} = V_{LSB} = \frac{V_{fs}}{2^N} \tag{10.3}$$

el número de bits de los DAC's es de $N = 12bits$, y típicamente se toma una $V_{fs} = 2.5V$, con lo que:

$$\mathrm{V}_{resol} = V_{LSB} = \frac{2.5}{2^{12}} = 610.35\mu V \tag{10.4}$$

El caso descrito arriba se corresponde con un caso ideal, es decir, considerando DAC's ideales y un comparador ideal (con impedancia de entrada infinita). En la realidad esto no va a ser así, con lo que las entradas del comparador se verán un poco atenuadas y las tensiones mínimas a comparar serán menores que las calculadas. Después de varias simulaciones y ensayos se ha llegado a la conclusión de que con un comparador de 15 bits de resolución se tiene suficiente margen para que el comparador trabaje sin ningún tipo de problema. Con lo que:

$$Resol = 15bits \quad \rightarrow \quad V_{LSB}\,|_{15bits} = \frac{V_{fs}}{2^N} = \frac{2.5}{2^{15}} = 76.30\mu V \tag{10.5}$$

Para concluir, en la tabla 10.1 se muestra un resumen de las especificaciones a cumplir por el comparador.

Resolución	76.30µV (15bits)
Velocidad	166k comparaciones por segundo
Consumo	~30µA

Tabla 10.1: Especificaciones del comparador

10.2 TOPOLOGÍA

La topología elegida para la implementación de un comparador con las características descritas anteriormente, se basa en una de las topologías más comúnmente utilizadas en SAR ADC's, la cual consta de dos bloques (preamplificador y *latch*) y trabaja en dos modos de operación (*tracking* y *latching*).

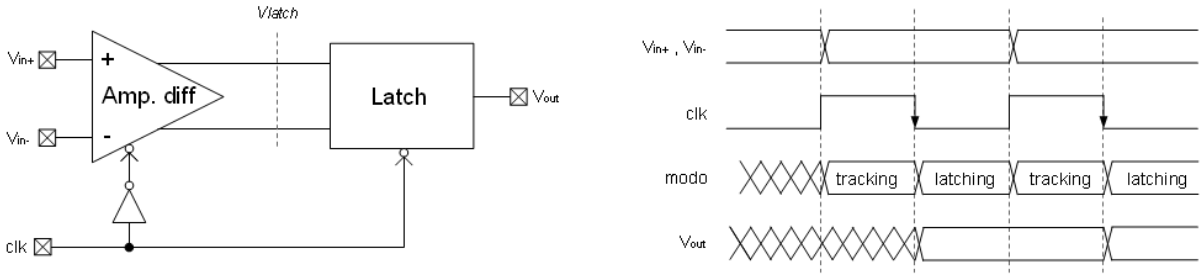


Figura 10.1: Topología básica del comparador. Cronograma de funcionamiento

El funcionamiento de este tipo de comparadores es el siguiente: el modo normal de funcionamiento es el modo *tracking*, en el cual el preamplificador está habilitado y va siguiendo las tensiones V_{in+} y V_{in-} a comparar, amplificándolas hasta los valores mínimos de entrada requeridos por el *latch*. En el momento en el que se desea realizar una comparación, se pasa al modo de *latching*, en el cual el amplificador se deshabilita y el *latch* muestrea (*strobe*) la tensión a su entrada, amplificándola y generando a su salida (V_{out}) el nivel lógico correspondiente.

En este tipo de topología, el elemento crítico es el preamplificador, pues es el que va a determinar las especificaciones de resolución y velocidad del comparador. Por un lado, ajustando su ganancia, se consigue fijar la resolución del comparador, pues para que el *latch* funcione correctamente necesita de unos valores de tensión mínimos a su entrada, de tal manera que para ganancias del preamplificador mayores, las tensiones V_{in+} y V_{in-} a comprar podrán ser más pequeñas, lo cual se traduce en una resolución del comparador mayor.

En cuanto a la velocidad, es de destacar que el *latch*, por su propia naturaleza, va a ser sobradamente más veloz que el preamplificador, de tal manera que el tiempo que necesite el preamplificador para amplificar las tensiones de entrada hasta los valores de tensión adecuados, será el que limite la velocidad del comparador.

10.2.1 Sistema de cancelación de offset

En este tipo de topologías, en la cuales se requiere del uso de preamplificadores, aparece un problema añadido cuando la resolución del comparador es elevada, como ocurre en este caso. El *offset* del preamplificador va a ser varios órdenes de magnitud mayor que la tensión mínima de entrada, con lo que la comparación se va a ver perturbada. Para un preamplificador típico, el *offset* a su entrada está comprendido entre unos 3mV y 6mV. Si se requiere de una resolución de 15 bits, la tensión de entrada mínima a comparar será de 76.30µV, con lo que es claro que el *offset* influirá sobre el resultado de la comparación.

Para solucionar este problema, se requiere de la implementación de un sistema de cancelación de *offset*, mediante el cual compensar y anular el *offset* del preamplificador.

A continuación, se muestra la topología final del comparador, en la cual se ha implementado un preamplificador de dos etapas y su correspondiente sistema de cancelación de *offset*.

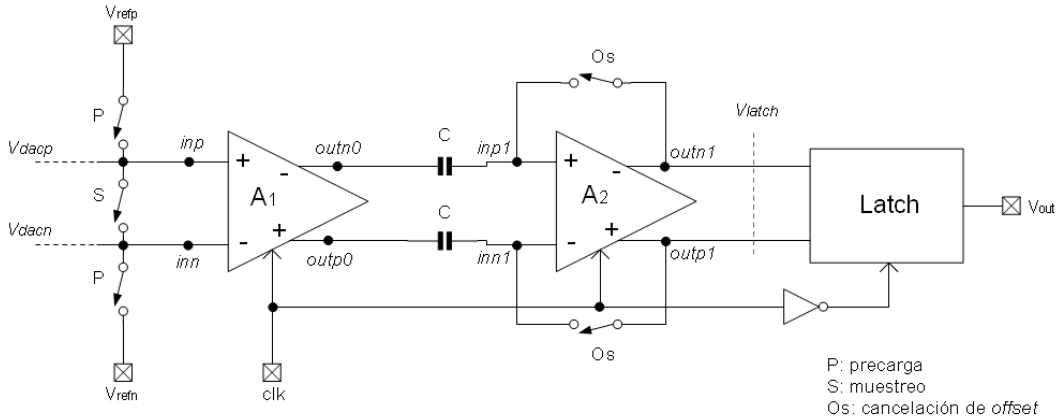


Figura 10.2: Topología del comparador implementado

Como se puede apreciar en la figura, el sistema de cancelación de *offset* está formado simplemente por dos capacidades y un conjunto de *switches*. El objetivo de este sistema es almacenar el *offset* de ambos amplificadores en las capacidades, de tal manera que durante el proceso de comparación, la tensión almacenada en ellas compense el efecto del *offset*. De esta forma, se añade un nuevo modo de funcionamiento al comparador: modo de cancelación de *offset*, durante el cual se procederá a almacenar la tensión de *offset* de los amplificadores en las capacidades.

El modo de cancelación de *offset* se ejecuta una vez por conversión del ADC, aprovechando el estado de muestreo (`sSample`) del convertidor, en el cual los *switches* de precarga (P) y muestreo (S) están en la posición requerida para almacenar el *offset*, y no es

necesario que el comparador realice ninguna comparación.

Teniendo en cuenta el nuevo modo de cancelación de *offset*, la secuencia de funcionamiento que seguirá el comparador para cada conversión del ADC será la siguiente:

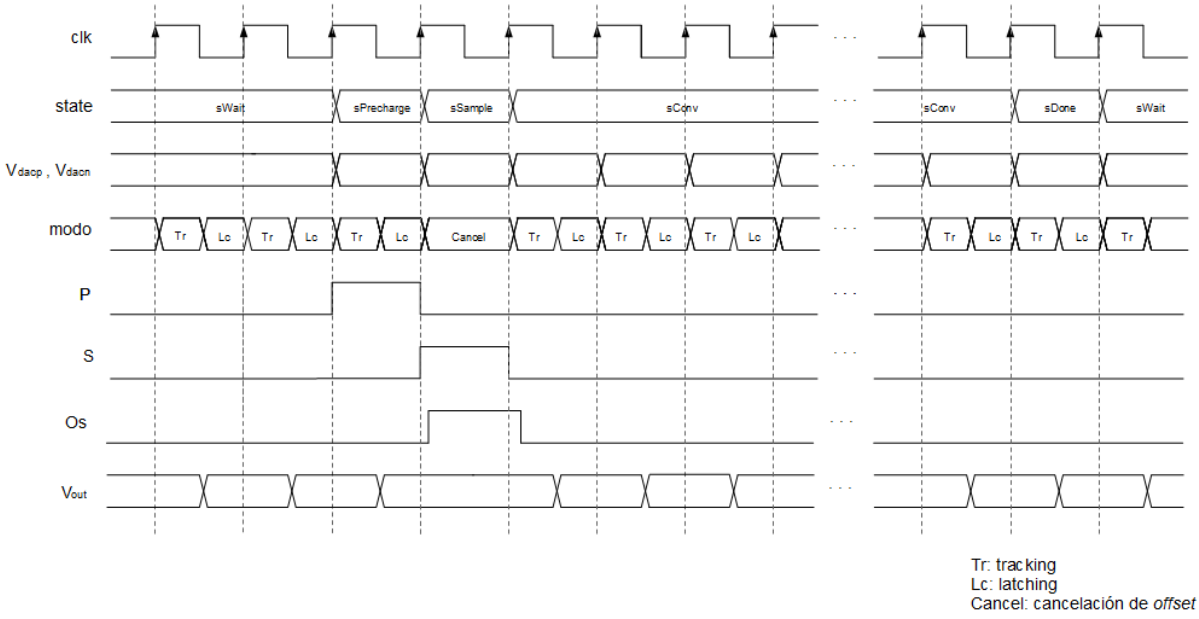


Figura 10.3: Cronograma de funcionamiento del comparador

Durante el modo de cancelación de *offset*, los *switches* P y S están fijando sobre la entrada del primer amplificador la tensión de modo común (V_{com}), es decir, la tensión diferencial de entrada del primer amplificador es cero, con lo que a la salida de éste se tendrá su correspondiente tensión de *offset* de salida (V_{OS1}).

Por otro lado, los *switches* Os se mantienen cerrados, definiendo un cortocircuito entre la salida y la entrada del amplificador 2, de esta forma, se tiene una realimentación que queda determinada por:

$$(outp1 - outn1) = \frac{A_2}{1 + A_2} \cdot V_{OS2} \approx V_{OS2} \qquad \textbf{(10.6)}$$

donde V_{OS2} es la tensión de *offset* de entrada del amplificador 2.

En esta situación, se tiene que sobre las capacidades de cancelación de *offset* (C) se están almacenando dos tensiones: por un lado la tensión de *offset* de salida del amplificador 1 (V_{OS1}), y por otro lado, una tensión muy similar a la tensión de *offset* de entrada del amplificador 2 (V_{OS2}).

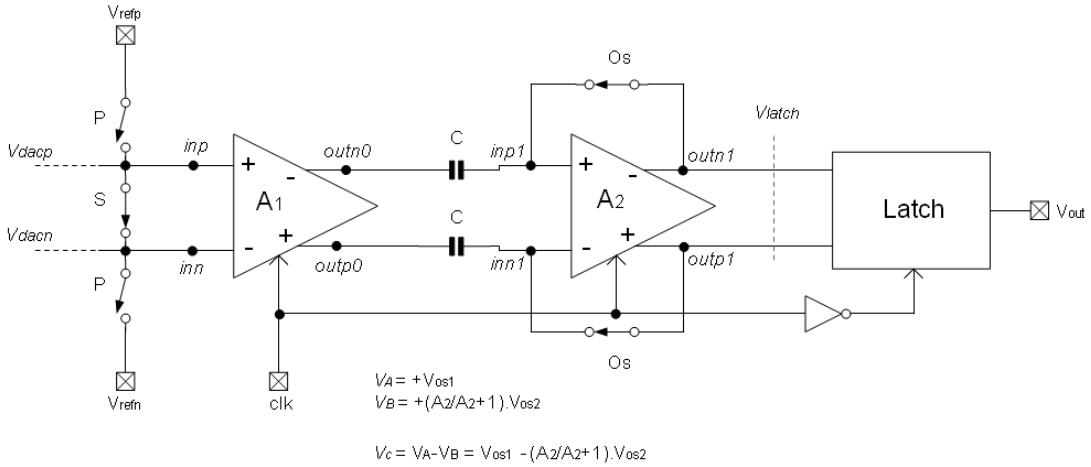


Figura 10.4: Comparador en modo cancelación de offset

Con ello se tiene que, durante el modo de *tracking*, el *offset* de salida del amplificador 1 quedará totalmente compensado gracias a la tensión almacenada en las capacidades (C), y el *offset* de entrada del amplificador 2 se compensará también, pero no totalmente, pues como se ha visto en la expresión 10.6, la tensión almacenada en C no es exactamente V_{OS2} .

$$(outp0 - outn0) = A_1 \cdot \left[\left(inp + \frac{V_{OS1}}{A_1} \right) - inn \right] \tag{10.7}$$

$$(inp1 - inn1) = -(outp0 - outn0) + V_C = -A_1 \cdot \left[\left(inp + \frac{V_{OS1}}{A_1} \right) - inn \right] + \left[V_{OS1} - \frac{A_2}{A_2 + 1} \cdot V_{OS2} \right] =$$

$$= A_1 \cdot (inp - inn) - A_1 \cdot \frac{V_{OS1}}{A_1} - \frac{A_2}{A_2 + 1} \cdot V_{OS2} + V_{OS1} = A_1 \cdot (inp - inn) - \frac{A_2}{A_2 + 1} \cdot V_{OS2} \tag{10.8}$$

$$(outp1 - outn1) = A_2 \cdot \left[\; inp1 + V_{OS2} \; - inn1 \right] = A_2 \cdot \; inp1 - inn1 \; + A_2 \cdot V_{OS2} =$$

$$= -A_2 \cdot A_1 \cdot \; inp - inn \; - A_2 \cdot \left(\frac{A_2}{A_2 + 1} \cdot V_{OS2} \right) + A_2 \cdot V_{OS2} = \tag{10.9}$$

$$= -A_2 \cdot A_1 \cdot \; inp - inn \; + \frac{A_2}{A_2 + 1} \cdot V_{OS2} \approx -A_2 \cdot A_1 \cdot \; inp - inn \; + V_{OS2}$$

Según se demuestra en las ecuaciones anteriores, el *offset* residual que quedará a la salida de las dos etapas de amplificación será aproximadamente igual al *offset* de entrada del amplificador 2 (V_{OS2}). Este *offset* residual no va a ser significativo en este punto, pues las señales a comparar ya han sido

amplificadas anteriormente, con lo que a la salida de la última etapa de amplificación el V_{OS2} será mucho menor que las señales de interés, con lo que la comparación no se verá afectada.

Para concluir con el sistema de cancelación de *offset*, señalar algunos aspectos más en cuanto a su diseño:

- Como se puede observar en el cronograma de la figura 10.3, la secuencia de encendido y apagado de los *switches* para la cancelación de *offset* es la siguiente:

	Cancelación de *offset*	*Tracking / Latching*
P	OFF	OFF
S	ON	OFF
Os	ON	OFF

Tabla 10.2: Estado de los switches de cancelación de offset

En donde, los *switches* Os se comportan exactamente de la misma forma que el *switch* S, pero con un pequeño *delay* añadido (~2.8ns). Este *delay* es necesario para que el *offset* de los amplificadores se almacene de forma correcta en las capacidades C.

- El dimensionamiento de las capacidades de cancelación de *offset* (C), va a estar determinado fundamentalmente por el ruido $K \cdot T / C$. Se elegirá un valor de C lo más pequeño posible, para favorecer la velocidad de los preamplificadores (carga menor) y reducir el área, y a la vez, lo suficientemente grande como para que el ruido $K \cdot T / C$ no perturbe a la comparación.

Teniendo en cuenta lo anterior, se ha elegido un valor de capacidad de 2pF, para el cual se tiene un ruido $K \cdot T / C$ muy por debajo de la resolución del comparador:

$$\begin{cases} V_{ruido} = \sqrt{\dfrac{K \cdot T}{C}} \\ K: \quad cte \quad Boltzman \quad (1.3806504 \cdot 10^{-23}\ {}^{J}\!/\!_{K}) \\ T: \quad Temperatura \quad (300K) \\ C: \quad capacidad \quad muestreo \quad (2\,pF) \end{cases} \longrightarrow \quad V_{ruido} = 45.50\mu V < V_{resol} = 76.30\mu V$$

10.3 LATCH

El principio de funcionamiento del *latch* se puede explicar fácilmente a partir de su circuito equivalente: dos amplificadores inversores idénticos conectados entre sí de la siguiente forma:

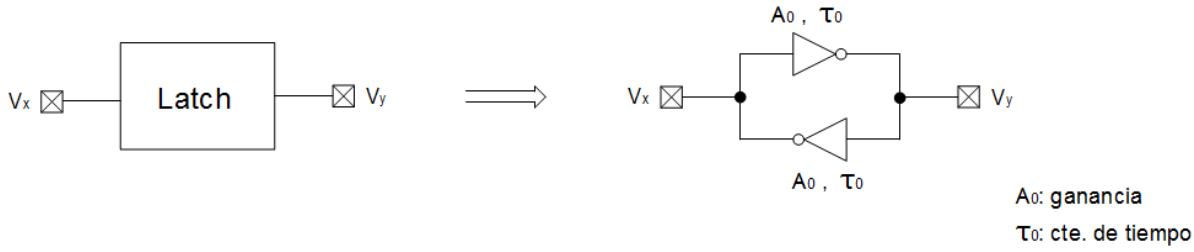


Figura 10.5: Circuito equivalente del latch

Analizando el circuito, se puede llegar a que:

$$\begin{cases} \tau_0 \cdot \dfrac{dV_x}{dt} + V_x = -A_0 \cdot V_y \\ \tau_0 \cdot \dfrac{dV_y}{dt} + V_y = -A_0 \cdot V_x \end{cases} \longrightarrow \quad \tau_0 \cdot \frac{d(V_x - V_y)}{dt} = -(1 - A_0) \cdot (V_x - V_y) \qquad \textbf{(10.10)}$$

si en el instante de muestreo (t = 0) se tiene:

$$(V_x - V_y)\big|_{t=0} = V_{xy0} \qquad \textbf{(10.11)}$$

entonces:

$$V_x - V_y = V_{xy0} \cdot e^{\frac{(A_0 - 1) \cdot t}{\tau_0}} \qquad \textbf{(10.12)}$$

,donde la constante de tiempo es: $\dfrac{\tau_0}{(A_0 - 1)}$

De la expresión anterior se deduce el comportamiento del circuito: a partir de una tensión V_{xy0} inicial (tensión muestreada) el *latch* generará una tensión estable $(V_x - V_y)$, pues los amplificadores tienden a saturarse con una constante de tiempo $\dfrac{\tau_0}{(A_0 - 1)}$. Habitualmente, para un *latch* típico se tiene una $A_0 >> 1$, con lo que la constante de tiempo será bastante pequeña.

El tiempo que tarda el *latch* en pasar de una tensión inicial V_{xy0} muy pequeña a su entrada, a una tensión final estable (V_{xy1}) dependerá directamente de la constante de tiempo de los amplificadores y del valor de tensión inicial V_{xy0}.

$$T = \frac{\tau_0}{A_0 - 1} \cdot \ln\left(\frac{V_{xy1}}{V_{xy0}}\right) \qquad \textbf{(10.13)}$$

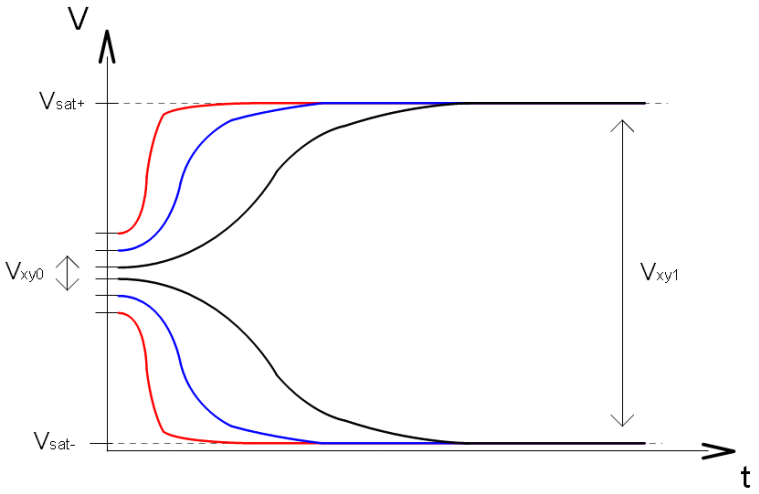


Figura 10.6: Evolución de la tensión en bornes del latch

Como se observa en la gráfica, a medida que la tensión V_{xy0} muestreada es mayor, el tiempo requerido por el *latch* para alcanzar un estado estable disminuye.

A partir de lo anterior, es posible definir el fenómeno de metaestabilidad, que ocurre en aquellos casos en los que el valor V_{xy0} muestreado es tan pequeño que el *latch* necesitaría de un tiempo infinito para estabilizarse, es decir, nunca llega a un estado estable, apareciendo de esta forma un valor desconocido a la salida. Por otro lado, también es posible la metaestabilidad en aquellos casos en los que el tiempo de muestreo T_s (tiempo entre dos muestreos consecutivos de V_{xy0}) fuera insuficiente para que el *latch*, teniendo a su entrada una V_{xy0} determinada, no llegara a estabilizarse.

Para evitar la metaestabilidad, es necesario garantizar que en el tiempo de muestreo disponible (T_s), el *latch* va a ser capaz de alcanzar un estado estable para la tensión V_{xy0} más pequeña posible a su entrada.

En este proyecto, no se contempla el diseño de un *latch* específico para el comparador, con lo que se ha reutilizado uno disponible en las librerías de la empresa (*Austriamicrosystems*). Después de haber simulado y medido los parámetros de interés de este *latch*, se han obtenido las siguientes características:

$Vin|_{min}$ para un Ts = 3µs	30mV
Consumo	~6.2µA

Tabla 10.3: Especificaciones del latch a utilizar

10.4 PREAMPLIFICADOR

10.4.1 Especificaciones

Las características de ganancia y velocidad del preamplificador, junto a su consumo, son las de mayor relevancia a la hora de abordar su diseño. Pues como se introdujo anteriormente, el preamplificador va a constituir el elemento crítico en cuanto a la velocidad y resolución alcanzadas por el comparador.

Para poder satisfacer las especificaciones fijadas para el comparador, se requiere que el preamplificador a diseñar cumpla con lo siguiente:

- La ganancia total del preamplificador ha de ser la suficiente como para mantener, en el peor de los casos, una tensión mínima de 30mV a la entrada del *latch*, y evitar de esta forma la posibilidad de un estado metaestable.

Minima tensión a comparar: $V_{resol} = 76.30\mu V$

Minima tension a la entrada del latch: $V_{latch}\,|_{min} = 30mV$

$$\downarrow$$

$$A_{min} = \frac{V_{latch}\,|_{min}}{V_{resol}} = \frac{30mV}{76.30\mu V} = 393.18 \approx 52dB \qquad \textbf{(10.14)}$$

Con esta ganancia mínima del preamplificador, se asegura una resolución para el comparador de 15 bits.

- La velocidad del preamplificador ha de ser la suficiente como para, en el peor de los casos, amplificar la señal de entrada hasta los valores de tensión adecuados en un tiempo menor a $T_{clk}/2$; pues el tiempo disponible para amplificar se corresponde con el tiempo en el que el comparador permanece en modo *tracking*.

Es decir, utilizando un reloj de 166kHz, como se ha impuesto anteriormente, el tiempo máximo disponible para amplificar la señal de entrada es de:

$$f_{clk} = 166kHz \quad \rightarrow \quad T_{clk} \approx 6\mu s$$

$$T_{comp} = T_{tracking} = \frac{T_{clk}}{2} = \frac{6\mu s}{2} = 3\mu s \qquad \textbf{(10.15)}$$

- El consumo total del preamplificador no debe superar los 23.8µA, para satisfacer de esta forma, la especificación de consumo del comparador.

En resumen, las especificaciones a cumplir por el preamplificador son: ganancia mínima de 52dB, tiempo máximo para la amplificación de 3µs y consumo máximo de 23.8µA. Para conseguir todo esto, no va a ser factible utilizar un sólo preamplificador, por lo que se ha optado por dividir la ganancia y utilizar dos etapas amplificadoras idénticas. Con lo cual, las especificaciones fijadas para cada una de estas dos etapas quedan como:

Ganancia mínima	26dB
$T_{amplificación}$ máximo	3µs
Consumo máximo	~11.9µA

Tabla 10.4: Especificaciones del preamplificador

Añadir para concluir que, la ganancia de cada etapa deberá ser un poco superior a la mínima especificada en la tabla, pues la atenuación introducida por las capacidades de cancelación de *offset* no se ha tenido en cuenta en el cálculo anterior.

10.4.2 Topología

La topología a implementar para las etapas de preamplificación, ha de ser una topología de tipo *fully-differential*, con una ganancia relativamente grande y un consumo muy reducido.

La primera arquitectura que se plantea utilizar es, la de un simple amplificador diferencial con carga de diodos. En este caso, se diseña un par diferencial de tipo N, para aprovechar al máximo la mayor ganancia intrínseca de los transistores NMOS, pero a pesar de ello, para un presupuesto de corriente tan reducido no es posible alcanzar la ganancia requerida.

$$g_{m1} = g_{m2} = g_{mA}$$
$$g_{m3} = g_{m4} = g_{mB}$$

$$\begin{cases} V_{outp} = -\dfrac{V_{inp} \cdot g_{mA}}{g_{mB}} \\ V_{outn} = -\dfrac{V_{inn} \cdot g_{mA}}{g_{mB}} \end{cases} \longrightarrow \quad A_d = \frac{(V_{outp} - V_{outn})}{(V_{inp} - V_{inn})} = -\frac{g_{mA}}{g_{mB}} \qquad \textbf{(10.15)}$$

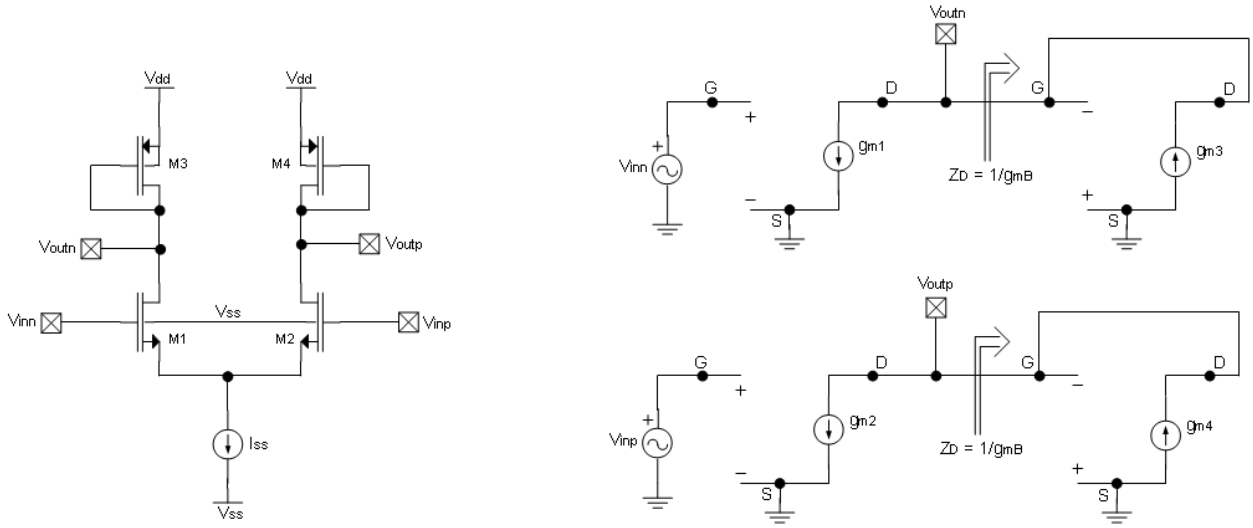


Figura 10.7: Amplificador diferencial con carga de diodos

Para mejorar sus prestaciones se recurre al uso de la realimentación, la cual permite multiplicar el factor de ganancia conseguido anteriormente. Ahora el problema radica en la velocidad, se consigue una ganancia elevada pero la velocidad queda muy mermada, pues para un consumo de corriente tan pequeño el amplificador es demasiado lento.

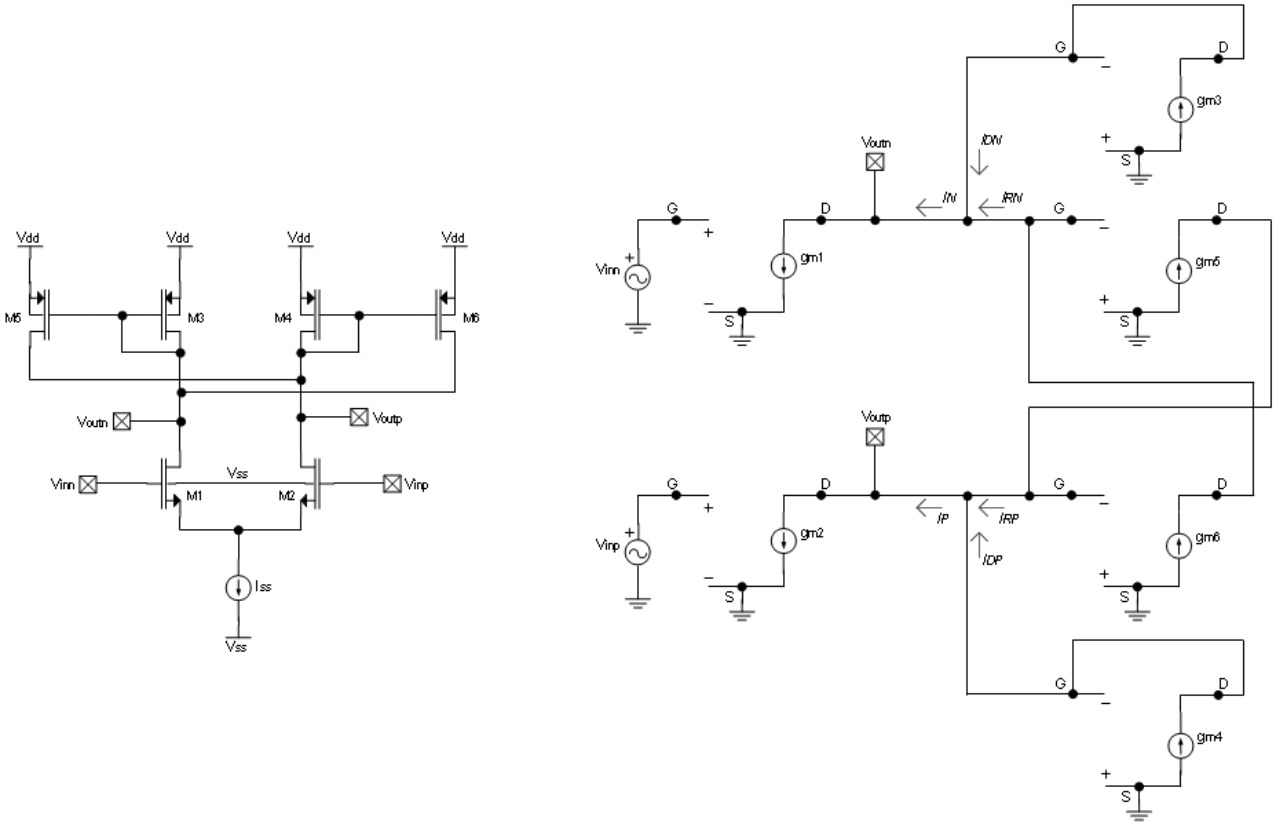


Figura 10.8: Amplificador diferencial con realimentación

$$\begin{aligned} &g_{m1} = g_{m2} = g_{mA} \\ &g_{m3} = g_{m4} = g_{mB} \\ &g_{m5} = g_{m6} = g_{mC} \end{aligned} \quad \Bigg| \quad \begin{aligned} &I_N = g_{m1} \cdot V_{inn} = g_{mA} \cdot V_{inn} \\ &I_{DN} = -g_{m3} \cdot V_{outn} = -g_{mB} \cdot V_{outn} \\ &I_P = g_{m2} \cdot V_{inp} = g_{mA} \cdot V_{inp} \\ &I_{DP} = -g_{m4} \cdot V_{outp} = -g_{mB} \cdot V_{outp} \end{aligned} \quad \begin{aligned} &Espejos \quad de \quad corriente: \\ &I_{RN} = I_{DP} \cdot \frac{g_{m6}}{g_{m4}} = I_{DP} \cdot \frac{g_{mC}}{g_{mB}} \\ &I_{RP} = I_{DN} \cdot \frac{g_{m5}}{g_{m3}} = I_{DN} \cdot \frac{g_{mC}}{g_{mB}} \end{aligned}$$

Definiendo las ecuaciones de corriente de los nodos de salida (V_{outn} ,V_{outn}), queda que:

$$\begin{cases} I_{DN} + I_{RN} = I_N \\ I_{DP} + I_{RP} = I_P \end{cases} \qquad \textbf{(10.16)}$$

Desarrollando el sistema de ecuaciones **(10.16)** se llega a que:

$$\begin{cases} -g_{mB} \cdot V_{outn} + (-g_{mB} \cdot V_{outp}) \cdot \dfrac{g_{mC}}{g_{mB}} = g_{mA} \cdot V_{inn} \\ -g_{mB} \cdot V_{outp} + (-g_{mB} \cdot V_{outn}) \cdot \dfrac{g_{mC}}{g_{mB}} = g_{mA} \cdot V_{inp} \end{cases} \qquad \textbf{(10.17)}$$

$$\downarrow$$

$$V_{inn} = \frac{-g_{mB} \cdot V_{outn} - g_{mC} \cdot V_{outp}}{g_{mA}} \qquad V_{inp} = \frac{-g_{mB} \cdot V_{outp} - g_{mC} \cdot V_{outn}}{g_{mA}} \longrightarrow A_d = \frac{(V_{outp} - V_{outn})}{(V_{inp} - V_{inn})} = \frac{g_{mA}}{g_{mC} - g_{mB}} \qquad \textbf{(10.18)}$$

De la ecuación obtenida se observa como, la realimentación hace aumentar la ganancia diferencial del amplificador.

Para solucionar el problema con la estructura anterior, se plantea el utilizar una nueva arquitectura. La nueva arquitectura se basa en el uso de dos lazos de realimentación: un interno y otro externo. Con esta solución, la ganancia total queda repartida entre el lazo interno y externo, lo cual da mejores resultados en cuanto al ratio ganancia-velocidad.

En la siguiente figura se muestra el esquemático de la nueva topología con doble lazo de realimentación:

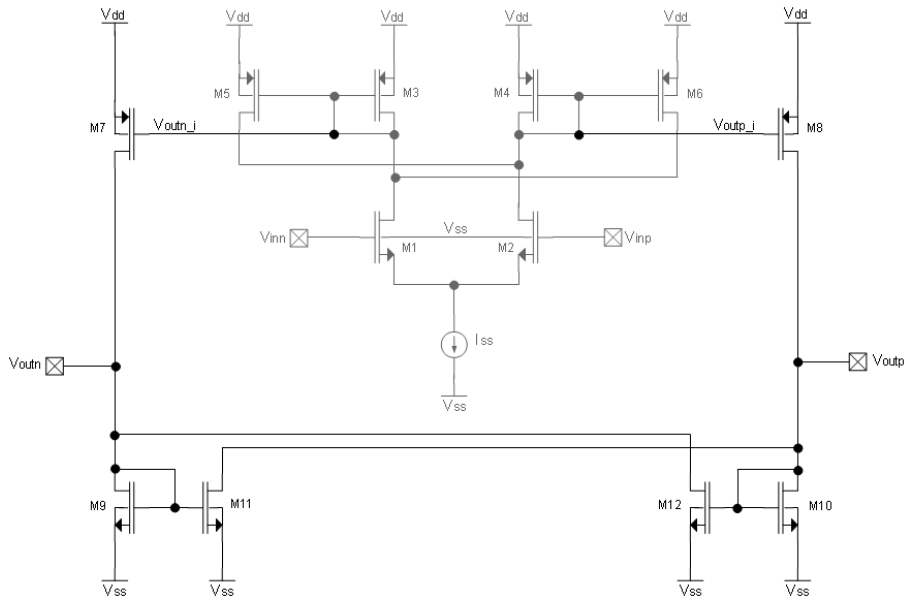


Figura 10.9: Amplificador diferencial con doble realimentación

Como se puede observar en el esquemático, el lazo de realimentación externo es idéntico al interno, con la única salvedad de que el interno está implementado con transistores P y el externo con transistores N. De esta forma, el análisis en pequeña señal del circuito se puede realizar de manera muy similar al anterior:

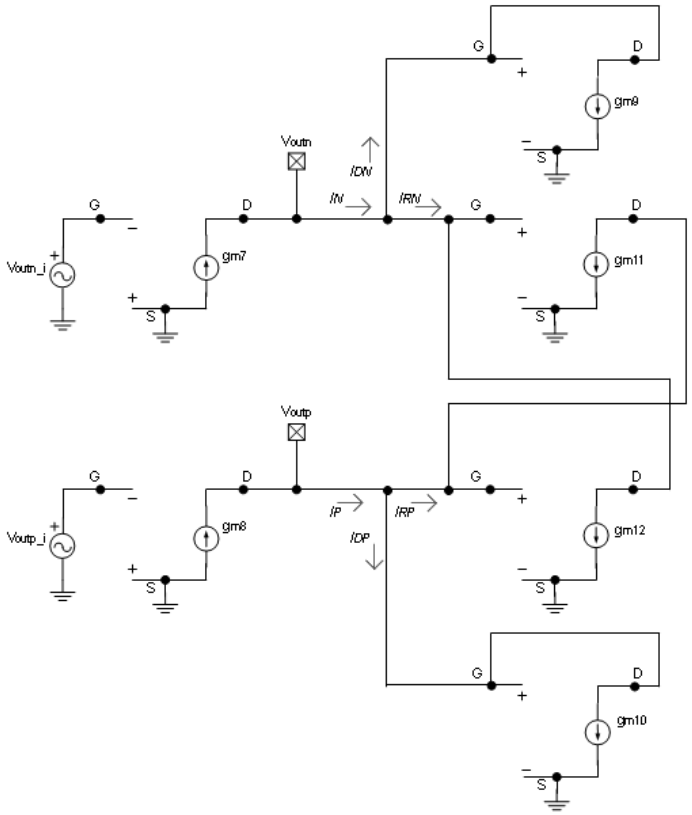


Figura 10.10: Circuito equivalente en pequeña señal del amplificador diferencial con doble realimentación

$$g_{m7} = g_{m8} = g_{mD}$$
$$g_{m9} = g_{m10} = g_{mE}$$
$$g_{m11} = g_{m12} = g_{mF}$$

$$I_N = -g_{m7} \cdot V_{outn\_i} = -g_{mD} \cdot V_{outn\_i}$$
$$I_{DN} = g_{m9} \cdot V_{outn} = g_{mE} \cdot V_{outn}$$
$$I_P = -g_{m8} \cdot V_{outp\_i} = -g_{mD} \cdot V_{outp\_i}$$
$$I_{DP} = g_{m10} \cdot V_{outp} = g_{mE} \cdot V_{outp}$$

Espejos de corriente:

$$I_{RN} = I_{DP} \cdot \frac{g_{m12}}{g_{m10}} = I_{DP} \cdot \frac{g_{mF}}{g_{mE}}$$
$$I_{RP} = I_{DN} \cdot \frac{g_{m11}}{g_{m9}} = I_{DN} \cdot \frac{g_{mF}}{g_{mE}}$$

Definiendo las ecuaciones de corriente de los nodos de salida (V_{outn} ,V_{outn}), queda que:

$$\begin{cases} I_{DN} + I_{RN} = I_N \\ I_{DP} + I_{RP} = I_P \end{cases} \quad \textbf{(10.19)}$$

Desarrollando el sistema de ecuaciones (**10.19)** se llega a que:

$$\begin{cases} g_{mE} \cdot V_{outn} + g_{mE} \cdot V_{outp} \cdot \dfrac{g_{mF}}{g_{mE}} = -g_{mD} \cdot V_{outn\_i} \\ g_{mE} \cdot V_{outp} + g_{mE} \cdot V_{outn} \cdot \dfrac{g_{mE}}{g_{mF}} = -g_{mD} \cdot V_{outp\_i} \end{cases} \quad \textbf{(10.20)}$$

$$\downarrow$$

$$V_{outn\_i} = \frac{-g_{mE} \cdot V_{outn} - g_{mF} \cdot V_{outp}}{g_{mD}}$$
$$V_{outp\_i} = \frac{-g_{mE} \cdot V_{outp} - g_{mF} \cdot V_{outn}}{g_{mD}}$$
$$\longrightarrow \quad A_{d\_e} = \frac{(V_{outp} - V_{outn})}{(V_{outp\_i} - V_{outn\_i})} = \frac{g_{mD}}{g_{mF} - g_{mE}} \quad \textbf{(10.21)}$$

De esta forma se ha obtenido la ganancia del circuito debida a la realimentación externa (A_{d_e}). La ganancia del circuito debida a la realimentación interna (A_{d_e}) es la misma que la calculada para el circuito de la figura 10.8, el cual presentaba una configuración con una sola realimentación. De esta manera se puede concluir que, la ganancia total de este circuito con doble lazo de realimentación es:

$$A_d = \frac{(V_{outp} - V_{outn})}{(V_{inp} - V_{inn})} = A_{d\_i} \cdot A_{d\_e} =$$
$$= \frac{(V_{outp\_i} - V_{outn\_i})}{(V_{inp} - V_{inn})} \cdot \frac{(V_{outp} - V_{outn})}{(V_{outp\_i} - V_{outn\_i})} = \left(\frac{g_{mA}}{g_{mC} - g_{mB}} \right) \cdot \left(\frac{g_{mD}}{g_{mF} - g_{mE}} \right) \quad \textbf{(10.22)}$$

Con las expresiones obtenidas del análisis en pequeña señal, es posible deducir el comportamiento de esta nueva arquitectura. La ganancia interna (A_{d_i}) y externa (A_{d_e}) responden a la misma característica, con lo que su comportamiento es equivalente al variar sus respectivos transistores:

Ganancia interna (A_{d_i})		**Ganancia externa (A_{d_e})**	
Tamaño par diferencial $(W/L)_{1,2}$	**$(W/L)_{1,2}$** mayor → **A_{d_i}** mayor *	**Tamaño par diferencial $(W/L)_{7,8}$**	**$(W/L)_{7,8}$** mayor → **A_{d_e}** mayor *
Ratio entre par diferencial y diodo $Ri = (W/L)_{1,2} / (W/L)_{3,4}$	**Ri** mayor → **A_{d_i}** mayor	**Ratio entre par diferencial y diodo $Re = (W/L)_{7,8} / (W/L)_{9,10}$**	**Re** mayor → **A_{d_e}** mayor
Ratio de corriente realimentada $(W/L)_{5,6} = \alpha i \cdot (W/L)_{3,4}$	**αi** mayor → **A_{d_i}** mayor	**Ratio de corriente realimentada $(W/L)_{11,12} = \alpha e \cdot (W/L)_{9,10}$**	**αe** mayor → **A_{d_e}** mayor

***** Si se realimenta mucha corriente la ganancia aumenta considerablemente, pero la corriente que pasa por M3,M9 / M4,M10 para polarizar los diodos disminuye rápidamente, con lo que a la mínima variación de ésta, los diodos pasarán a corte y el amplificador saturará. Es decir, el rango dinámico de entrada queda reducido al aumentar el valor de **αi / αe**.

Tabla 10.5: Comportamiento de la ganancia en función del dimensionamiento de los transistores

Con la topología presentada anteriormente, a simple vista parece factible conseguir la ganancia, la velocidad y el consumo para el preamplificador requeridos por las especificaciones. Pero existe un pequeño inconveniente en cuanto a la velocidad: para el peor caso, es decir, cuando el preamplificador debe pasar de estar saturado (tensión a su entrada muy elevada) a tener a su salida la tensión mínima (tensión a la entrada V_{resol}) el tiempo que requiere para desaturarse es muy elevado, lo cual se traduce en el incumplimiento de las especificación de velocidad para este caso.

Para conseguir mejores prestaciones en cuanto a velocidad, y de esta forma asegurar que en el caso peor se cumplan las especificaciones establecidas, se ha diseñado un sistema de *power-down*/*reset*, mediante el cual, se fuerzan las salidas del amplificador a V_{dd} y se desconectan las alimentaciones, de esta forma se consigue que el consumo del preamplificador en este estado sea nulo (*power-down*) y además se tenga un *reset* fuerte y muy rápido que permita al preamplificador desaturarse rápidamente y de esta forma cumplir con la especificación de velocidad.

El *power-down/reset* se activará durante unos nanosegundos antes de que el comparador pase a modo *tracking* (modo en el que el preamplificador está amplificando). Más adelante, se presentan

algunas gráficas de simulaciones del comparador, donde se puede observar claramente cómo actúa este sistema de *power-down/reset*.

Para implementar el *power-down/reset*, se añaden 8 transistores más a la topología ya presentada. Estos nuevos transistores realizan la función de *switches* y se encargan de, por una parte, forzar las salidas internas (V_{outn_i} ,V_{outp_i}) a V_{ss} y las salidas externas (V_{outn} ,V_{outp}) a V_{dd} , y por otra, cortar el paso de corriente por todas las ramas del circuito, para que en este estado el consumo sea nulo.

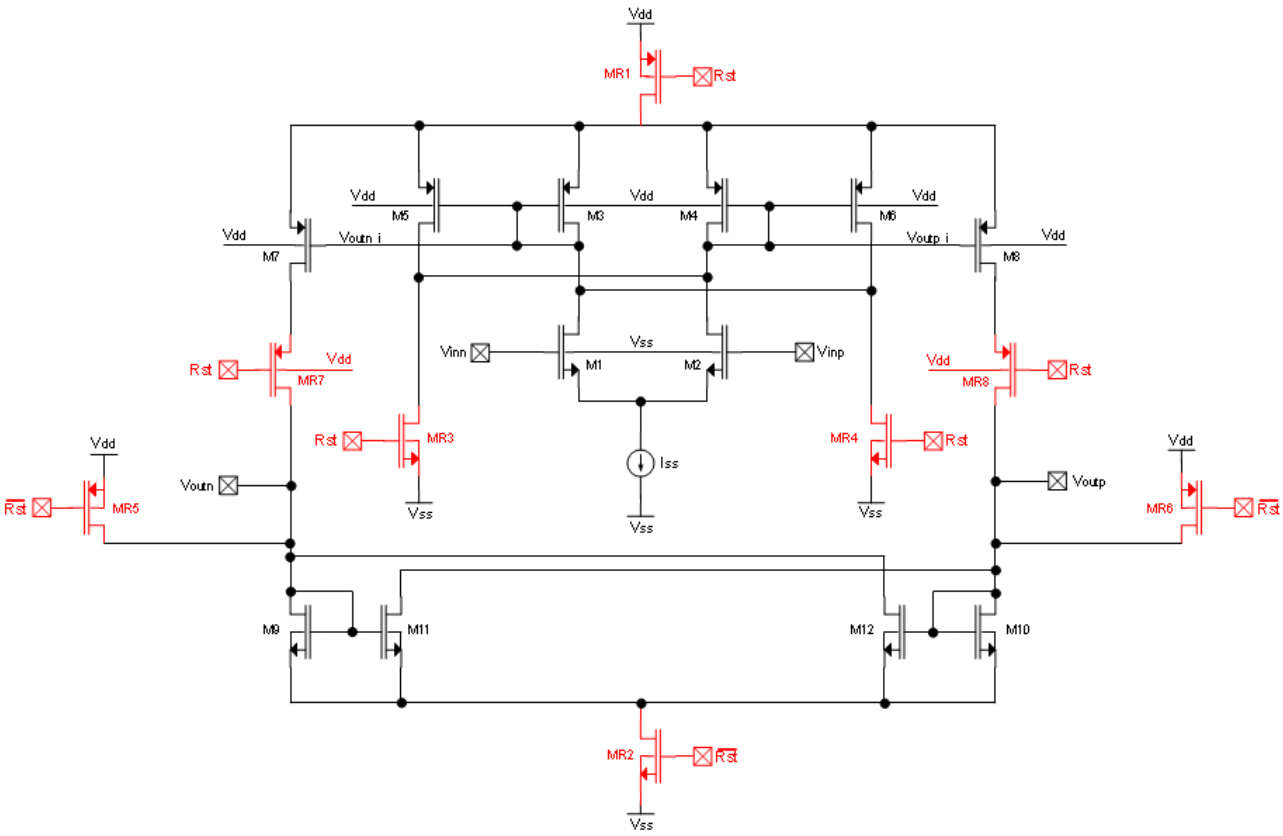


Figura 10.11: Amplificador diferencial con doble realimentación y sistema de power-down/reset

Para concluir, señalar que en el anexo A de este mismo documento, se adjunta el esquemático de la etapa amplificadora diseñada, donde se puede apreciar el dimensionamiento de los transistores utilizados.

10.4.3 Simulaciones

Para caracterizar totalmente la etapa amplificadora diseñada, se han realizado distintas simulaciones en DC, AC, TRAN y MC, con la finalidad de medir los parámetros más relevantes de ésta. Señalar también, que para las simulaciones se ha empleado el análisis de *corners*, considerando los casos peores (*worst case*) de los parámetros de proceso de los CMOS y la variación de: la tensión de alimentación (V_{supply}), la corriente de polarización (I_{bias}) y la temperatura (*Temp*).

En la figura 10.12 se presenta una tabla con todos los parámetros medidos (valores mínimos, máximos y típicos) y los *corners* que se han simulado.

De todos los parámetros medidos, los más relevantes para este caso particular son los señalados a continuación:

- Ganancia: la ganancia mínima conseguida está en torno a unos 32dB, superior a los 26dB que se habían definido como especificación.

- Velocidad: en la tabla se puede observar como el *Settling Time Rise* presenta un valor típico de unos 2.5µs, menor que los 3µs definidos en la especificación. Sin embargo, el *Settling Time Rise* máximo es de uno 4.6µs, con lo que este caso queda fuera de especificaciones; pero esto no va a ser preocupante en cuanto al funcionamiento global del comparador, pues se ha comprobado que al tener una ganancia mínima bastante superior a la requerida, el preamplificador alcanza una tensión de salida suficiente, a pesar de no llegar hasta el final de su amplificación.

- Consumo: el consumo máximo medido es de 12.1µA algo superior a los 11.9µA que se define en las especificaciones. El exceso es tan solo de 0.2µA, lo cual es prácticamente despreciable con lo que lo consideraremos como válido.

- Ruido: la tensión eficaz de ruido máxima está en torno a $46\mu V_{rms}$, la cual no va a influir sobre la tensión mínima a amplificar que es de V_{resol} = 76.30µV.

- *Offset*: la tensión de *offset* máxima medida es de 7mV, bastante elevada con respecto a la V_{resol}, pero que no es preocupante pues el sistema de cancelación de *offset* del comparador se encargará de compensarla.

Simulation Results (Process: c35b4)

Parameter	Symbol	Spec-Min	Min	Typ/Mean	Max	Spec-Max	StdDev	Unit	CPK
Sim-Type: ac (Corner)									
Conditions									
Temperature Range	Tjunc		-40 (2)	27	125 (6)		-	deg	
Power Supply Range	Vdd		3.0 (1)	3.0	3.6 (10)		-	V	
Input Common Mode Voltage (1)	Vcmin		1.250 (1)	1.250	1.250 (1)		-	V	
Common Mode Voltage (2)	Vcm		1.250 (1)	1.250	1.250 (1)		-	V	
Bias Current	Ibias		1.600 (2)	2.000	2.400 (18)		-	uA	
Parallel Diff. Load Capacitance	Cpload		2.000 (15)	2.000	2.001 (13)		-	pF	
Parallel Diff. Load Resistance	Rpload		20000.000 (12)	20000.000	20000.000 (11)		-	kOhm	
AC Parameter									
Open Loop Gain	A0		29 (8)	32	33 (33)		-	dB	
Gain Bandwidth	GBW		2.099 (7)	3.414	5.248 (26)		-	MHz	
Bandwidth	BW		86 (9)	127	188 (28)		-	kHz	
Phase Margin	PM		40 (19)	47	52 (16)		-	deg	
Noise Parameter									
Equiv. Input Voltage Noise RMS @ 1Hz - 1MHz	EIVNRMS		19.885 (26)	26.934	43.041 (7)		-	uVrms	
Equiv. Input Voltage Noise at 1Hz	EIVN1Hz		0.895 (1)	0.895	4.047 (16)		-	uV/sqrt(Hz)	
Equiv. Input Voltage Noise at 100kHz	EIVN100kHz		19.465 (26)	26.856	42.327 (9)		-	nV/sqrt(Hz)	
Sim-Type: acsweep (Corner)									
Input Parameter									
PeakPeak Diff. Input Range Low (3)	DIRL		-0.363 (31)	-0.281	-0.217 (2)		-	V	
PeakPeak Diff. Input Range High (3)	DIRH		0.217 (2)	0.281	0.363 (31)		-	V	
Common Mode Input Range High (Vdd - CMIRH) (4)	CMIRH		1.000 (1)	1.000	1.600 (10)		-	V	
Output Parameter									
Common Mode Output Range Low (5)	CMORL		-100.000 (32)	1.240	1.240 (11)		-	V	
Common Mode Output Range High (Vdd - CMORH) (5)	CMORH		-96.400 (32)	1.760	2.360 (11)		-	V	
Sim-Type: tran (Corner)									
Isource (6)	Isource		0.000 (2)	0.005	0.008 (31)		-	mA	
Isink (6)	Isink		0.002 (7)	0.003	0.008 (26)		-	mA	
DC Parameter									
Current Consumption no Load	Idd_nl		7.004 (6)	9.298	12.187 (27)		-	uA	
Power Consumption no Load	Pdiss_nl		21.011 (6)	27.893	43.875 (27)		-	uW	
Current Consumption with Load	Idd		7.004 (6)	9.298	12.187 (27)		-	uA	
Power Consumption with Load	Pdiss		21.011 (6)	27.893	43.875 (27)		-	uW	
Current Consumption Power Down	Idd_pd		0.073 (5)	0.076	199.605 (32)		-	nA	
Power Consumption Power Down	Pdiss_pd		0.218 (5)	0.227	718.577 (32)		-	nW	
Transient Parameter									
Slew Rate Rise (7)	SRP		0.715 (9)	0.912	1.122 (28)		-	V/us	
Slew Rate Fall (7)	SRN		0.538 (2)	0.742	0.993 (27)		-	V/us	
Settling Time Rise (0.1%) (7)	TSP		1.875 (24)	2.485	4.627 (7)		-	us	
Settling Time Fall (0.1%) (7)	TSN		1.921 (24)	2.580	4.894 (7)		-	us	
Overshoot Rise (7)	OVSP		2.102 (11)	5.697	8.927 (22)		-	%	
Overshoot Fall (7)	OVSN		0.000 (6)	5.416	11.901 (3)		-	%	
CM Slew Rate Rise (8)	SRP_cmfb		0.110 (23)	0.147	0.340 (29)		-	V/us	
CM Slew Rate Fall (8)	SRN_cmfb		0.119 (23)	0.156	0.303 (29)		-	V/us	
CM Settling Time Rise (0.1%) (8)	TSP_cmfb		25.293 (26)	30.716	34.178 (7)		-	us	
CM Settling Time Fall (0.1%) (8)	TSN_cmfb		29.021 (26)	35.752	45.362 (7)		-	us	
CM Overshoot Rise (8)	OVSP_cmfb		83.345 (19)	141.475	473.835 (17)		-	%	
CM Overshoot Fall (8)	OVSN_cmfb		93.502 (19)	152.168	490.633 (17)		-	%	
Wakeup Time (9)	Twakeup		27.357 (26)	33.042	39.609 (7)		-	us	
THD @ 1.0kHz,1.0Vpp	THD		-81.557 (23)	-69.270	-58.171 (2)		-	dBc	
Sim-Type: mc (MonteCarlo)									
MonteCarlo DC Parameter									
Output Referred Offset FD	Vos_27		-7.049	0.447	6.112		2.351	mV	
MonteCarlo AC Parameter									
Pos. Power Supply Rejection Ratio @1Hz	PSRRP1Hz_27		83.823	98.850	136.261		8.453	dB	
Pos. Power Supply Rejection Ratio @100kHz	PSRRP100kHz_27		74.485	85.073	104.275		5.945	dB	
Neg. Power Supply Rejection Ratio @1Hz	PSRRN1Hz_27		46.096	57.973	86.987		8.397	dB	
Neg. Power Supply Rejection Ratio @100kHz	PSRRN100kHz_27		46.641	58.297	82.134		7.856	dB	
Input Common Mode Rejection Ratio @1Hz	ICMRR1Hz_27		43.557	58.457	87.053		9.415	dB	
Input Common Mode Rejection Ratio @100kHz	ICMRR100kHz_27		43.561	58.433	87.065		9.371	dB	
Output Common Mode Rejection Ratio @1Hz	OCMRR1Hz_27		204.178	217.917	239.374		7.881	dB	
Output Common Mode Rejection Ratio @100kHz	OCMRR100kHz_27		104.245	117.980	139.026		7.869	dB	

Corner Setup (ac, acsweep, tran : dcf-file='fd_opi_corners.dcf')

Process='C35B4' ; Model-Path='/programs/ams_3.70_sr/spectre/c35/soac'								
Corner	**bip.scs**	**cap.scs**	**ind.scs**	**res.scs**	**cmos53.scs**	**Temp**	**Vsupply**	**Ibias**
corner1	tm	tm	tm	tm	tm	27.0	3	2u
corner2	tm	tm	tm	tm	wp	-40	3.0	1.6u
corner3	tm	tm	tm	tm	ws	-40	3.0	1.6u
corner4	tm	tm	tm	tm	wo	-40	3.0	1.6u
corner5	tm	tm	tm	tm	wz	-40	3.0	1.6u
corner6	tm	tm	tm	tm	wp	125	3.0	1.6u
corner7	tm	tm	tm	tm	ws	125	3.0	1.6u
corner8	tm	tm	tm	tm	wo	125	3.0	1.6u
corner9	tm	tm	tm	tm	wz	125	3.0	1.6u
corner10	tm	tm	tm	tm	wp	-40	3.6	1.6u
corner11	tm	tm	tm	tm	ws	-40	3.6	1.6u
corner12	tm	tm	tm	tm	wo	-40	3.6	1.6u
corner13	tm	tm	tm	tm	wz	-40	3.6	1.6u
corner14	tm	tm	tm	tm	wp	125	3.6	1.6u
corner15	tm	tm	tm	tm	ws	125	3.6	1.6u
corner16	tm	tm	tm	tm	wo	125	3.6	1.6u
corner17	tm	tm	tm	tm	wz	125	3.6	1.6u
corner18	tm	tm	tm	tm	wp	-40	3.0	2.4u
corner19	tm	tm	tm	tm	ws	-40	3.0	2.4u
corner20	tm	tm	tm	tm	wo	-40	3.0	2.4u
corner21	tm	tm	tm	tm	wz	-40	3.0	2.4u
corner22	tm	tm	tm	tm	wp	125	3.0	2.4u
corner23	tm	tm	tm	tm	ws	125	3.0	2.4u
corner24	tm	tm	tm	tm	wo	125	3.0	2.4u
corner25	tm	tm	tm	tm	wz	125	3.0	2.4u
corner26	tm	tm	tm	tm	wp	-40	3.6	2.4u
corner27	tm	tm	tm	tm	ws	-40	3.6	2.4u
corner28	tm	tm	tm	tm	wo	-40	3.6	2.4u
corner29	tm	tm	tm	tm	wz	-40	3.6	2.4u
corner30	tm	tm	tm	tm	wp	125	3.6	2.4u
corner31	tm	tm	tm	tm	ws	125	3.6	2.4u
corner32	tm	tm	tm	tm	wo	125	3.6	2.4u
corner33	tm	tm	tm	tm	wz	125	3.6	2.4u

Figura 10.12: Caracterización del amplificador diseñado

10.5 LÓGICA DE CONTROL

Además del *latch* y del preamplificador, el comparador va a requerir de un bloque digital de control que se encargue de generar la secuencia de señales necesarias para gobernar el funcionamiento del circuito, es decir, conmutar correctamente los *switches* de cancelación de *offset*, resetear el preamplificador, habilitar el *latch*…

En la siguiente figura se muestran las señales de entrada/salida requeridas para controlar el funcionamiento del comparador, y la secuencia que siguen durante un ciclo de conversión del ADC:

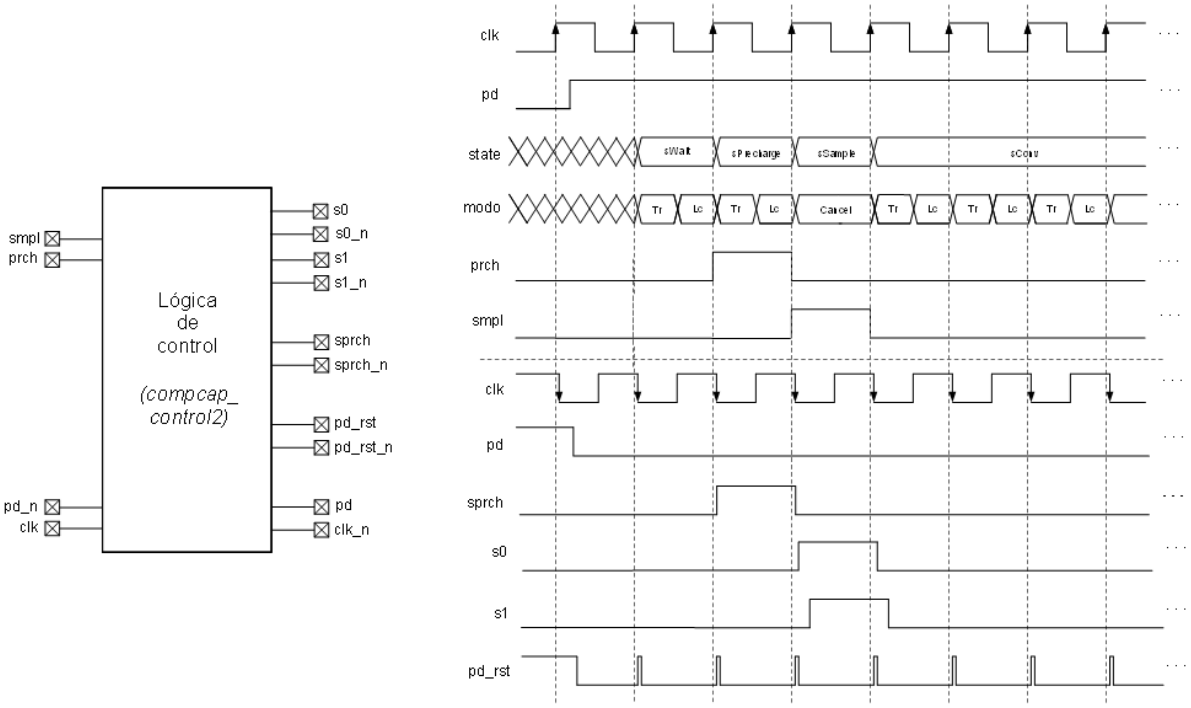


Figura 10.13: Señales de entrada/salida del bloque de control (del comparador)

Entradas:

`clk` : reloj del sistema. Mediante esta entrada se le pasa a la lógica de control el reloj requerido para secuenciar todas las señales de salida.

`pd_n` : *power-down* (activo a nivel bajo). Entrada asíncrona que se corresponde con el `por_n` del sistema. Cuando está activa deshabilita el *latch* y fuerza el *power-down/reset* del preamplificador.

prch : precarga. Entrada síncrona que se corresponde con la señal `precharge` de la lógica de control del ADC (SAR12cal). Se utiliza para conmutar correctamente los *switches* de cancelación de *offset*.

smpl : muestreo. Entrada síncrona que se corresponde con la señal `sample` de la lógica de control del ADC (SAR12cal). Se utiliza para conmutar correctamente los *switches* de cancelación de *offset*.

Salidas:

clk_n : reloj del sistema negado. Señal requerida para el control del *latch*. En el flanco de subida de esta señal el *latch* se habilita, con lo que el comparador pasa del modo *tracking* al modo *latching*.

pd : *power-down*. Señal necesaria junto a `pd_n`, para desactivar el *latch*.

pd_rst : *power-down/reset*. Señal síncrona de salida que se encarga de resetear las etapas preamplificadoras.

pd_rst_n : *power-down/reset* negada.

sprch : *switch* P. Señal síncrona de salida que se encarga de generar la secuencia de activación/desactivación necesaria para el control de los *switches* P.

sprch_n : *switch* P negada.

s0 : *switch* S. Señal síncrona de salida que se encarga de generar la secuencia de activación/desactivación necesaria para el control del *switches* S.

s0_n : *switch* S negada.

s1 : *switch* Os. Señal síncrona de salida que se encarga de generar la secuencia de activación/desactivación necesaria para el control de los *switches* Os.

s1_n : *switch* Os negada.

La lógica digital que implementa la generación de estas señales es extremadamente sencilla, pues se basa en el uso de simples puertas lógicas y bloques de retardo. En el anexo A se muestra una captura del esquemático empleado para la implementación de este bloque.

10.6 SIMULACIONES

Una vez definidas las características del comparador y diseñados sus tres bloques principales: preamplificador, *latch* y lógica de control, de tal forma que se cumpla con las especificaciones requeridas, se pasa a simular el funcionamiento de todo el conjunto.

Con las simulaciones *toplevel* se comprueba que el comparador trabaje correctamente y se mueva dentro de las especificaciones impuestas. En este caso se ha realizado una simulación TRAN, en la cual, las señales de entrada a comparar varían con una frecuencia de 166kHz (el reloj del ADC) y pasan por distintos niveles de tensión extremos, para asegurar de esta forma que, el comparador responderá adecuadamente hasta en los peores casos de señales a comparar.

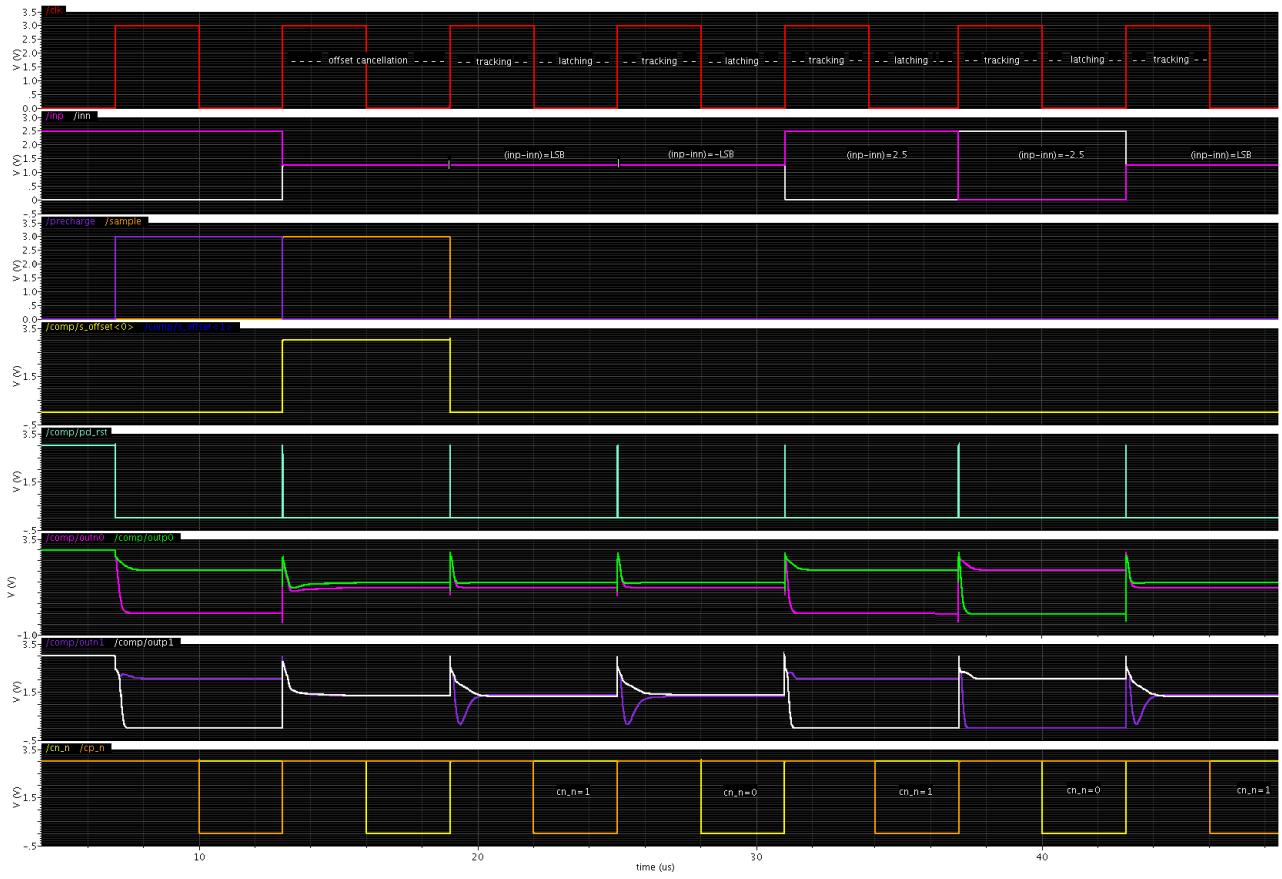


Figura 10.14: Simulación toplevel del comparador

A continuación, se muestra un detalle de la simulación en el cual se puede observar cómo trabaja el sistema de cancelación de *offset*. A la salida de la primera etapa de amplificación ($oun0$, $oup0$) el *offset* ha alterado las señales a comparar, lo que conllevaría una comparación errónea en el caso de

que éste no fuera compensado, pero posteriormente, tras pasar por las capacidades de compensación (C) y por la segunda etapa de amplificación, el *offset* queda compensado y las salidas ($oun1, oup1$) son las correctas.

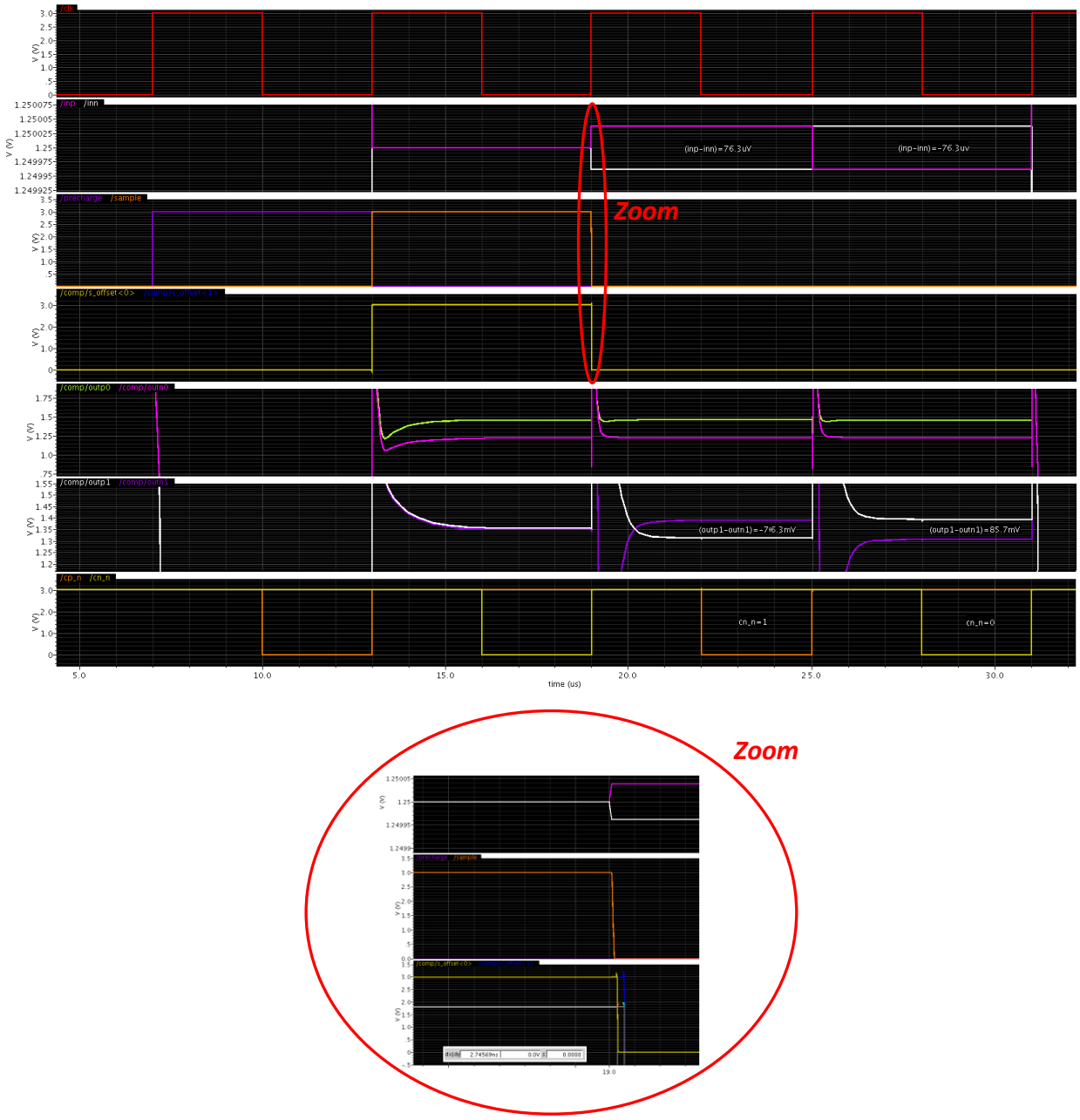


Figura 10.15: Simulación toplevel del comparador (detalle cancelación de offset)

Finalmente, para concluir con las simulaciones, se realiza un análisis de *corners* para corroborar el funcionamiento del comparador en los casos peores (*worst case*) de los parámetros de proceso de los CMOS, y frente a variaciones de: la tensión de alimentación (V_{supply}), la corriente de

polarización (I_{bias}) y la temperatura (*Temp*).

En la siguiente tabla, se muestran los resultados obtenidos y la lista de *corners* que han sido simulados. Se puede observar cómo para el peor de los casos, a la entrada del *latch* se tiene una tensión mínima de 35mV (superior a la tensión mínima de 30mV definida en la especificación) con lo que la comparación se realiza de forma satisfactoria. Por otro lado, es de destacar el dato del consumo, que está alrededor de los 32µA para el caso peor, y alrededor de los 28µA para el caso típico.

Simulation Results (Process: c35b4)

Parameter	Symbol	Spec-Min	Min	Typ/Mean	Max	Spec-Max	StdDev	Unit	CPK
Sim-Type: trans (Corner)									
Conditions									
Temperature Range	Tjunc		-40 (2)	27	125 (6)		-	deg	
Power Supply Range	Vdd		3.0 (1)	3.0	3.6 (10)		-	V	
Input Common Mode Voltage	Vcom		1.250 (1)	1.250	1.250 (1)		-	V	
Input Referred Offset (amplifier0)	Vcom		-	-	-		-	mV	
DC Parameter									
Current Consumption	Idd		22.565 (6)	28.793	36.649 (27)		-	uA	
Power Consumption	Pdiss		67.694 (6)	86.379	131.938 (27)		-	uW	
Transient Outputs (preamplifier0)									
Power Down State	s1_0		-0.000 (22)	0.000	0.000 (1)		-	V	
Precharge State	s2_0		1.374 (2)	2.055	2.846 (31)		-	V	
Sample State (offset cancellation)	s3_0		0.186 (4)	0.237	0.278 (29)		-	V	
Comparation State (input = LSB)	s4_0		0.187 (4)	0.241	0.286 (33)		-	V	
Comparation State (input = -LSB)	s5_0		0.182 (4)	0.235	0.279 (33)		-	V	
Comparation State (input = 2.5)	s6_0		1.374 (2)	2.055	2.846 (31)		-	V	
Comparation State (input = -2.5)	s7_0		-2.846 (31)	-2.055	-1.374 (2)		-	V	
Comparation State (input = LSB)	s8_0		0.187 (4)	0.241	0.286 (33)		-	V	
Transient Outputs (preamplifier1)									
Power Down State	s1_1		-0.000 (1)	-0.000	0.000 (14)		-	V	
Precharge State	s2_1		-2.846 (31)	-2.055	-1.372 (2)		-	V	
Sample State (offset cancellation)	s3_1		0.006 (12)	0.006	0.006 (9)		-	V	
Comparation State (input = LSB)	s4_1		-0.113 (33)	-0.078	-0.036 (15)		-	V	
Comparation State (input = -LSB)	s5_1		0.062 (4)	0.093	0.130 (33)		-	V	
Comparation State (input = 2.5)	s6_1		-2.846 (31)	-2.053	-1.367 (2)		-	V	
Comparation State (input = -2.5)	s7_1		1.369 (2)	2.055	2.846 (31)		-	V	
Comparation State (input = LSB)	s8_1		-1.026 (30)	-0.076	-0.035 (15)		-	V	
Transient Outputs (comparator)									
Power Down State	s1_out		3.000 (8)	3.000	3.600 (29)		-	V	
Power Down State	s1_out_n		3.000 (8)	3.000	3.600 (13)		-	V	
Precharge State	s2_out		3.000 (8)	3.000	3.600 (13)		-	V	
Precharge State	s2_out_n		0.000 (18)	0.000	0.000 (17)		-	V	
Sample State (offset cancellation)	s3_out		0.000 (18)	0.000	0.000 (17)		-	V	
Sample State (offset cancellation)	s3_out_n		3.000 (8)	3.000	3.600 (13)		-	V	

Comparation State (input = LSB)	s4_out		3.000 (8)	3.000	3.600 (13)		-	V	
Comparation State (input = LSB)	s4_out_n		0.000 (2)	0.000	0.000 (33)		-	V	
Comparation State (input = -LSB)	s5_out		0.000 (2)	0.000	0.000 (33)		-	V	
Comparation State (input = -LSB)	s5_out_n		3.000 (8)	3.000	3.600 (13)		-	V	
Comparation State (input = 2.5)	s6_out		3.000 (8)	3.000	3.600 (13)		-	V	
Comparation State (input = 2.5)	s6_out_n		0.000 (18)	0.000	0.000 (33)		-	V	
Comparation State (input = -2.5)	s7_out		0.000 (2)	0.000	0.000 (33)		-	V	
Comparation State (input = -2.5)	s7_out_n		3.000 (8)	3.000	3.600 (13)		-	V	
Comparation State (input = LSB)	s8_out		3.000 (8)	3.000	3.600 (13)		-	V	
Comparation State (input = LSB)	s8_out_n		0.000 (2)	0.000	0.000 (33)		-	V	
Latch Delay									
Comparation State (input = LSB)	s4_delay		1.442 (26)	3.333	6.340 (7)		-	ns	
Comparation State (input = -LSB)	s5_delay		1.416 (26)	3.265	6.111 (7)		-	ns	
Comparation State (input = 2.5)	s6_delay		1.425 (26)	3.286	6.132 (7)		-	ns	
Comparation State (input = -2.5)	s7_delay		1.391 (26)	3.234	6.016 (7)		-	ns	
Comparation State (input = LSB)	s8_delay		1.444 (26)	3.334	6.295 (7)		-	ns	

Corner Setup (trans : dcf-file='comparator2_corners.dcf')

Process='C35B4' ; Model-Path='/programs/ams_3.70_sr/spectre/c35/soac'								
Corner	**bip.scs**	**cap.scs**	**ind.scs**	**res.scs**	**cmos53.scs**	**Temp**	**Vsupply**	**Ibias**
corner1	tm	tm	tm	tm	tm	27	3	2u
corner2	tm	tm	tm	tm	wp	-40	3.0	1.6u
corner3	tm	tm	tm	tm	ws	-40	3.0	1.6u
corner4	tm	tm	tm	tm	wo	-40	3.0	1.6u
corner5	tm	tm	tm	tm	wz	-40	3.0	1.6u
corner6	tm	tm	tm	tm	wp	125	3.0	1.6u
corner7	tm	tm	tm	tm	ws	125	3.0	1.6u
corner8	tm	tm	tm	tm	wo	125	3.0	1.6u
corner9	tm	tm	tm	tm	wz	125	3.0	1.6u
corner10	tm	tm	tm	tm	wp	-40	3.6	1.6u
corner11	tm	tm	tm	tm	ws	-40	3.6	1.6u
corner12	tm	tm	tm	tm	wo	-40	3.6	1.6u
corner13	tm	tm	tm	tm	wz	-40	3.6	1.6u
corner14	tm	tm	tm	tm	wp	125	3.6	1.6u
corner15	tm	tm	tm	tm	ws	125	3.6	1.6u
corner16	tm	tm	tm	tm	wo	125	3.6	1.6u
corner17	tm	tm	tm	tm	wz	125	3.6	1.6u
corner18	tm	tm	tm	tm	wp	-40	3.0	2.4u
corner19	tm	tm	tm	tm	ws	-40	3.0	2.4u
corner20	tm	tm	tm	tm	wo	-40	3.0	2.4u
corner21	tm	tm	tm	tm	wz	-40	3.0	2.4u
corner22	tm	tm	tm	tm	wp	125	3.0	2.4u
corner23	tm	tm	tm	tm	ws	125	3.0	2.4u
corner24	tm	tm	tm	tm	wo	125	3.0	2.4u
corner25	tm	tm	tm	tm	wz	125	3.0	2.4u
corner26	tm	tm	tm	tm	wp	-40	3.6	2.4u
corner27	tm	tm	tm	tm	ws	-40	3.6	2.4u

corner28	tm	tm	tm	tm	wo	-40	3.6	2.4u
corner29	tm	tm	tm	tm	wz	-40	3.6	2.4u
corner30	tm	tm	tm	tm	wp	125	3.6	2.4u
corner31	tm	tm	tm	tm	ws	125	3.6	2.4u
corner32	tm	tm	tm	tm	wo	125	3.6	2.4u
corner33	tm	tm	tm	tm	wz	125	3.6	2.4u

Figura 10.16: Resultados de las simulaciones toplevel del comparador

BLOQUE III

RESULTADOS Y CONCLUSIONES

CAPÍTULO 11

Análisis y simulaciones

En el presente capítulo se presentan las principales simulaciones realizadas durante la consecución de este proyecto. En ellas se pueden observar los aspectos más relevantes del funcionamiento del ADC diseñado, y comprobar que los resultados finales obtenidos se corresponden con lo esperado.

Gran parte de las simulaciones realizadas se han centrado en la evaluación de las características de linealidad del DAC, pues es el bloque principal del convertidor y el que determina directamente la linealidad del ADC. La mayor parte del esfuerzo realizado en este proyecto ha recaído sobre la implementación del sistema de calibración, encargado de compensar las no linealidades del ADC derivadas del uso de un DAC de área muy reducida. Para poder comparar y comprobar el efecto compensatorio de la calibración, todas las simulaciones de linealidad se han repetido para el ADC con calibración y sin calibración.

A continuación se enumeran las distintas simulaciones contenidas en este capítulo:

- Simulaciones *toplevel*. En ellas se muestra la secuencia de funcionamiento seguida por el ADC en sus dos fases de funcionamiento: calibración y conversión. Se representan el estado de las señales de control, del DAC y del comparador en cada momento del proceso de conversión.

- Simulaciones de linealidad. Muestran los resultados de INL y DNL obtenidos tras la simulación del ADC con calibración y sin ella. Se incluye un análisis de la linealidad del DAC de calibración (calDAC).

11.1 SIMULACIONES *TOPLEVEL*

En las siguientes páginas se incluyen dos capturas de los resultados obtenidos tras las simulaciones a nivel funcional (*toplevel*). En ellas se muestran todas aquellas señales relevantes que intervienen durante el proceso de conversión del ADC: señales digitales generadas por la lógica de control, salidas analógicas de los DAC's, salidas de las etapas preamplificadoras internas del comparador, y salida del comparador.

- `DACs_outputs` : salidas de los DAC's.

- `comp_Amp0` : salidas de la primera etapa preamplificadora del comparador.

- `comp_Amp1` : salidas de la segunda etapa preamplificadora del comparador.

- `control_signals` : reloj, señales de *power-down* y reset, estado de las FSM y señales de control relacionadas con la lógica digital de control.

En la primera de las figuras, se observa el comportamiento del ADC cuando se encuentra en plena ejecución de la fase de calibración, es decir, midiendo los errores de *matching* de cada una de las capacidades del convDAC. Concretamente, en esta figura se encuentra buscando el error de *matching* de la capacidad C_{11}.

En la segunda figura, se muestra el estado del ADC cuando se lleva a cabo la fase de conversión, es decir, se ejecuta el algoritmo de búsqueda SAR a la vez que se aplica la calibración necesaria sobre el DAC, para convertir correctamente la tensión analógica a la entrada.

Figura 11.1: Funcionamiento del ADC en la fase de calibración

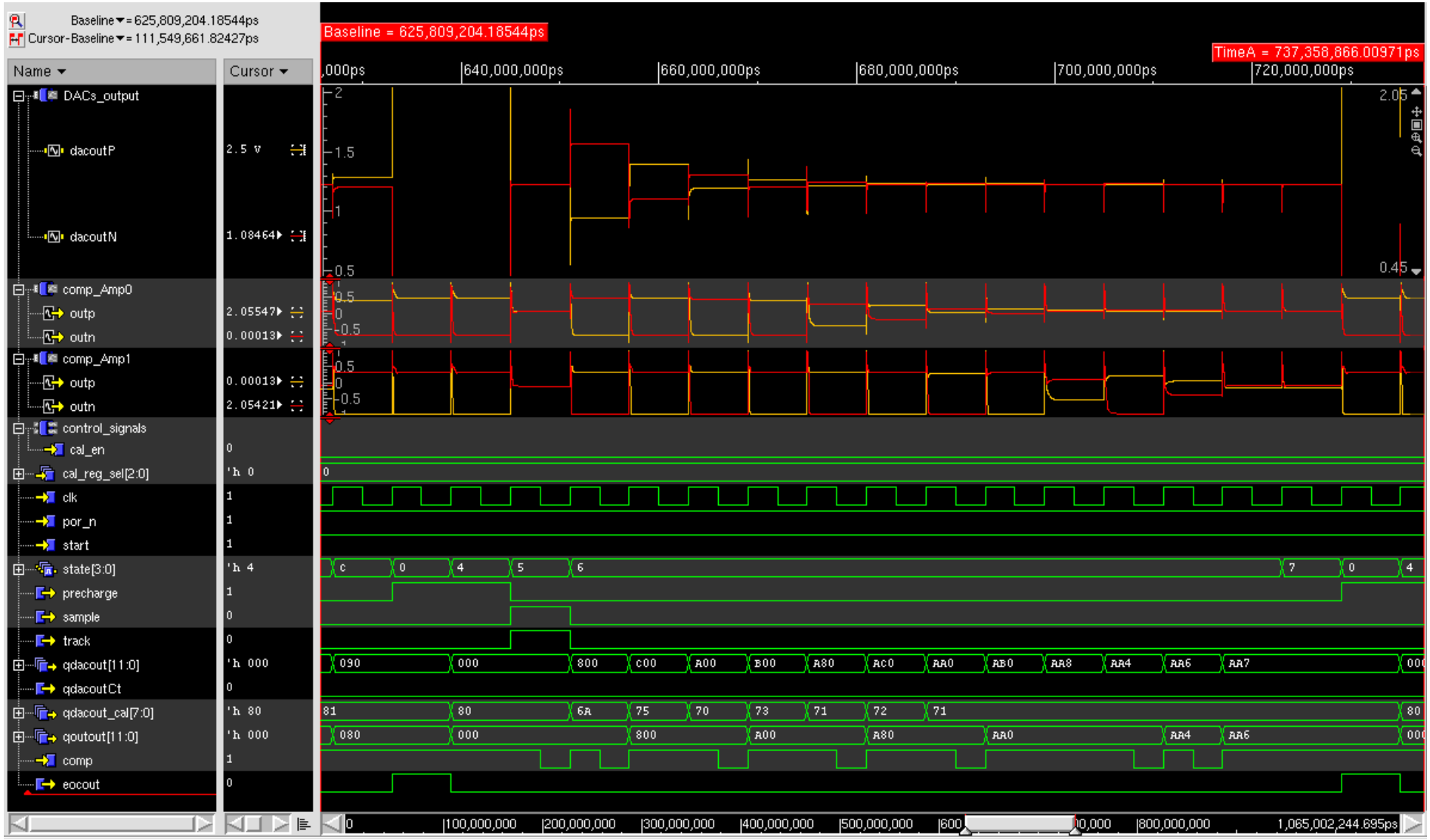


Figura 11.2: Funcionamiento del ADC en la fase de conversión

11.2 SIMULACIONES DE LINEALIDAD

En esta parte se presentan distintas gráficas de INL y DNL obtenidas tras las simulaciones de linealidad del ADC. Las simulaciones de linealidad se han realizado tomando como tensión de entrada del ADC una rampa desde $-V_{ref}/2$ a $V_{ref}/2$ (todo el rango dinámico de tensiones de entrada del ADC) y realizando 16 conversiones por cada escalón de cuantificación del ADC (16 *hits per code*).

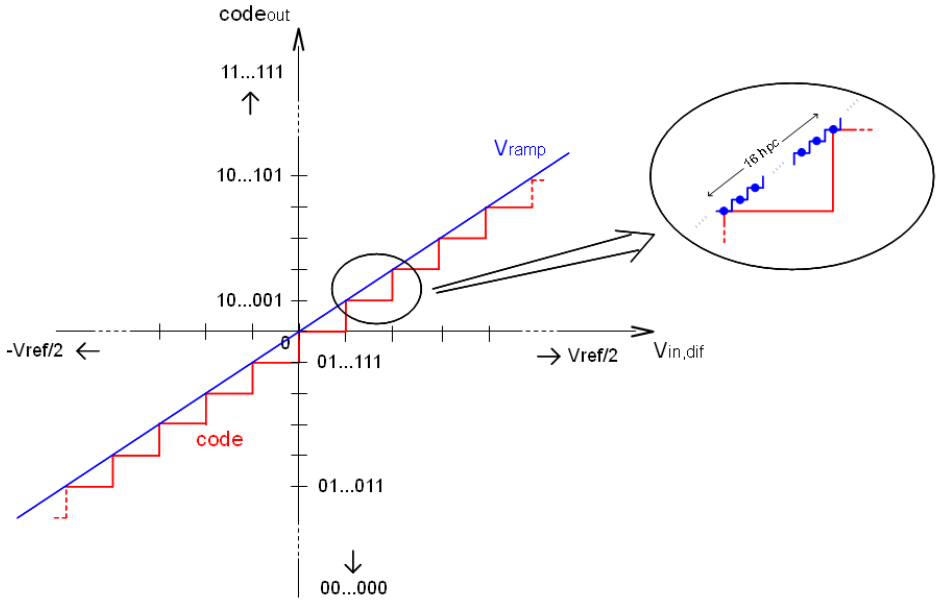


Figura 11.3: Análisis de linealidad con 16hpcl

Como se ha introducido anteriormente, todas las simulaciones de linealidad se han repetido dos veces: una haciendo uso de la calibración y otra sin la calibración del DAC. Ésto, como se verá a continuación, ofrece una visión clara de la potencia del sistema de calibración implementado.

Para modelar los problemas de linealidad de los DAC's, debidos a los parásitos del *layout* y al *matching* entre capacidades, se ha procedido a capturar las capacidades parásitas extraídas del *layout* elaborado e incluirlas en el esquemático de simulación del DAC. Además, para añadir los errores de *matching* de las capacidades, se ha procedido a variar, de forma aleatoria, el valor de cada capacidad en un $\pm 0.5\%$ (se estima un *matching* para el *layout* elaborado de 8 bits).

A continuación, se detallan los aspectos más relevantes de cada una de las simulaciones que se adjuntan en hojas sucesivas:

- Figura 11.4. En ella se muestran las características del INL/DNL del ADC cuando éste prescinde de la calibración. Como se puede observar, los errores de INL son significativos, pues alcanzan valores de hasta $\pm 3.5 LSB$. Incluso, como se deduce de los picos de DNL de $-1 LSB$, se tienen *missing codes* para ciertos códigos del ADC.

- Figura 11.5. Se trata de las gráficas de INL/DNL del mismo DAC de la figura anterior, pero ahora con el sistema de calibración funcionando. Se nota a primera vista, que la calibración ha conseguido compensar los errores de linealidad del DAC, quedando ahora acotados todos ellos entre $\pm 0.4 LSB$ aproximadamente. Se considera que un ADC presenta buenas características de linealidad cuando su INL/DNL se encuentra acotada entre $\pm 0.5 LSB$, con lo que en este caso, la calibración cumple con creces su cometido. También se puede apreciar como, los *missing codes* que se tenían con el ADC sin calibrar han sido compensados.

- Figura 6. En esta figura se representa la linealidad del DAC de calibración (de 8 bits) aislado. Se observa que éste presenta una buena característica de linealidad, pues era de esperar, ya que el *matching* para el *layout* elaborado se estimaba en unos 8 bits.

- Figura 7-10. Estas cuatro figuras muestran la INL y DNL del ADC calibrado y sin calibrar para distintos errores de *matching* en el DAC. En el eje X se representan los códigos del ADC, en el eje Y el valor de la INL/DNL y en el eje Z los distintos casos de errores de *matching* del DAC. Con estas simulaciones se comprueba el correcto funcionamiento de la calibración para distintos posibles errores de *matching* de los DAC's.

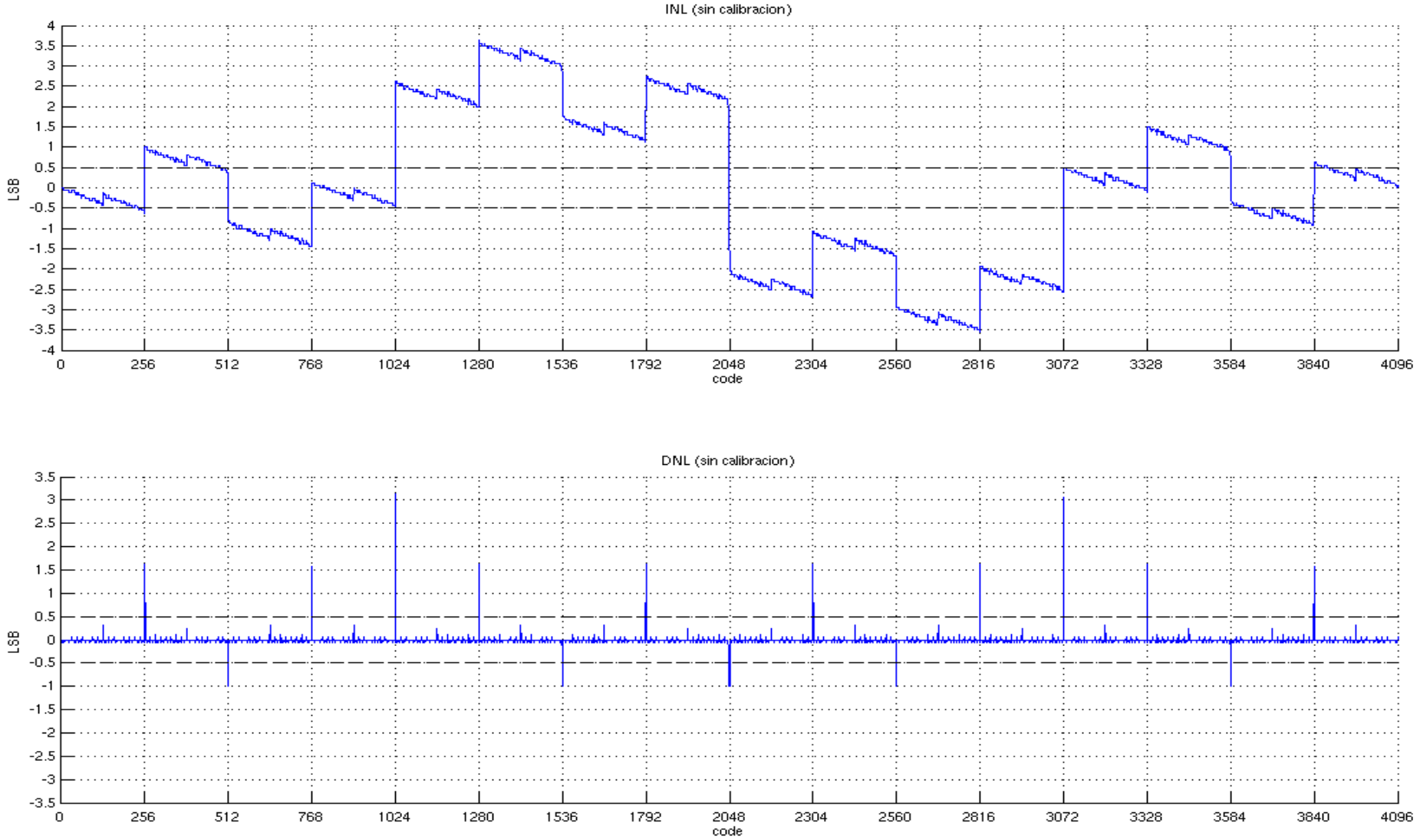


Figura 11.4: INL/DNL del ADC sin calibración

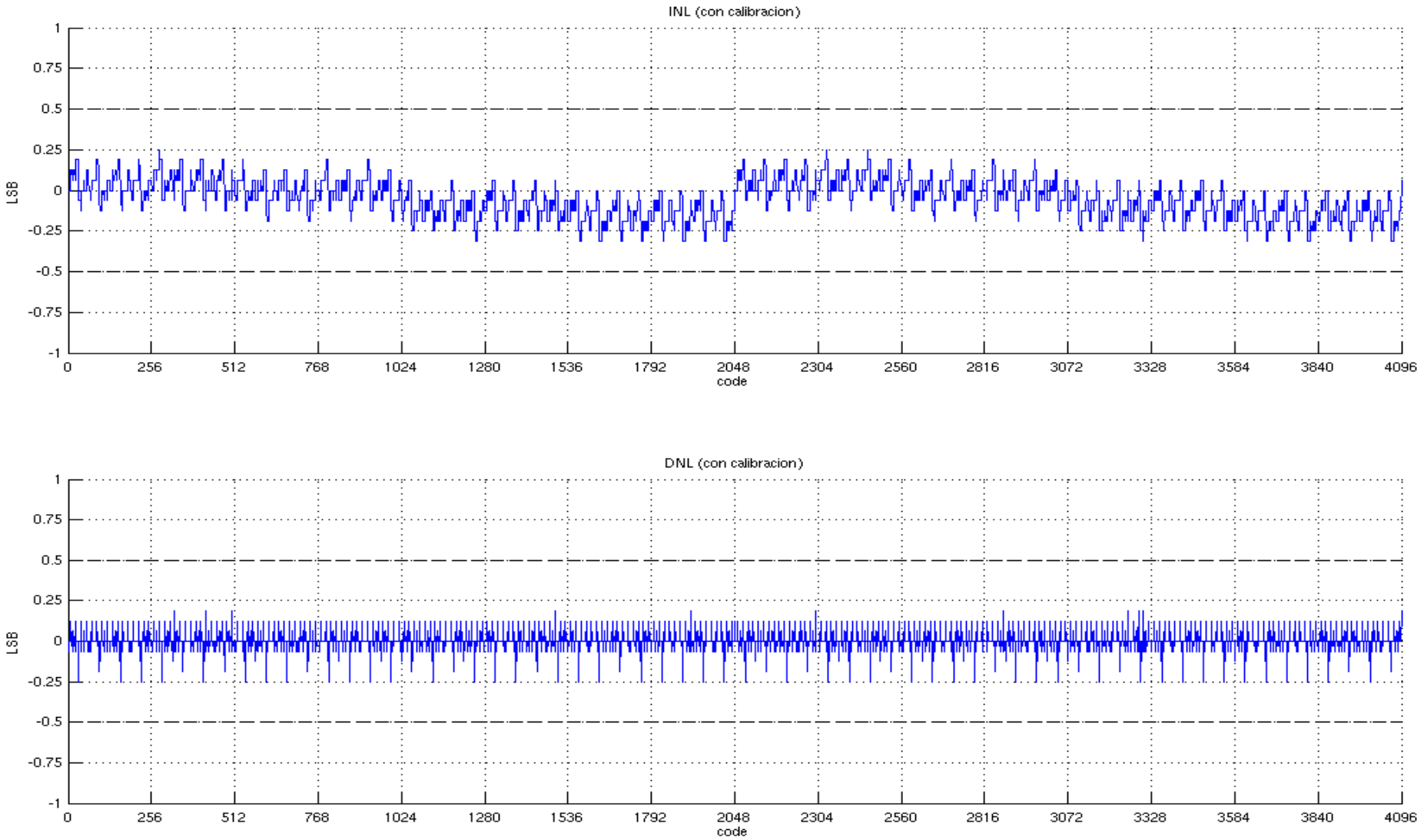


Figura 11.5: INL/DNL del ADC con calibración

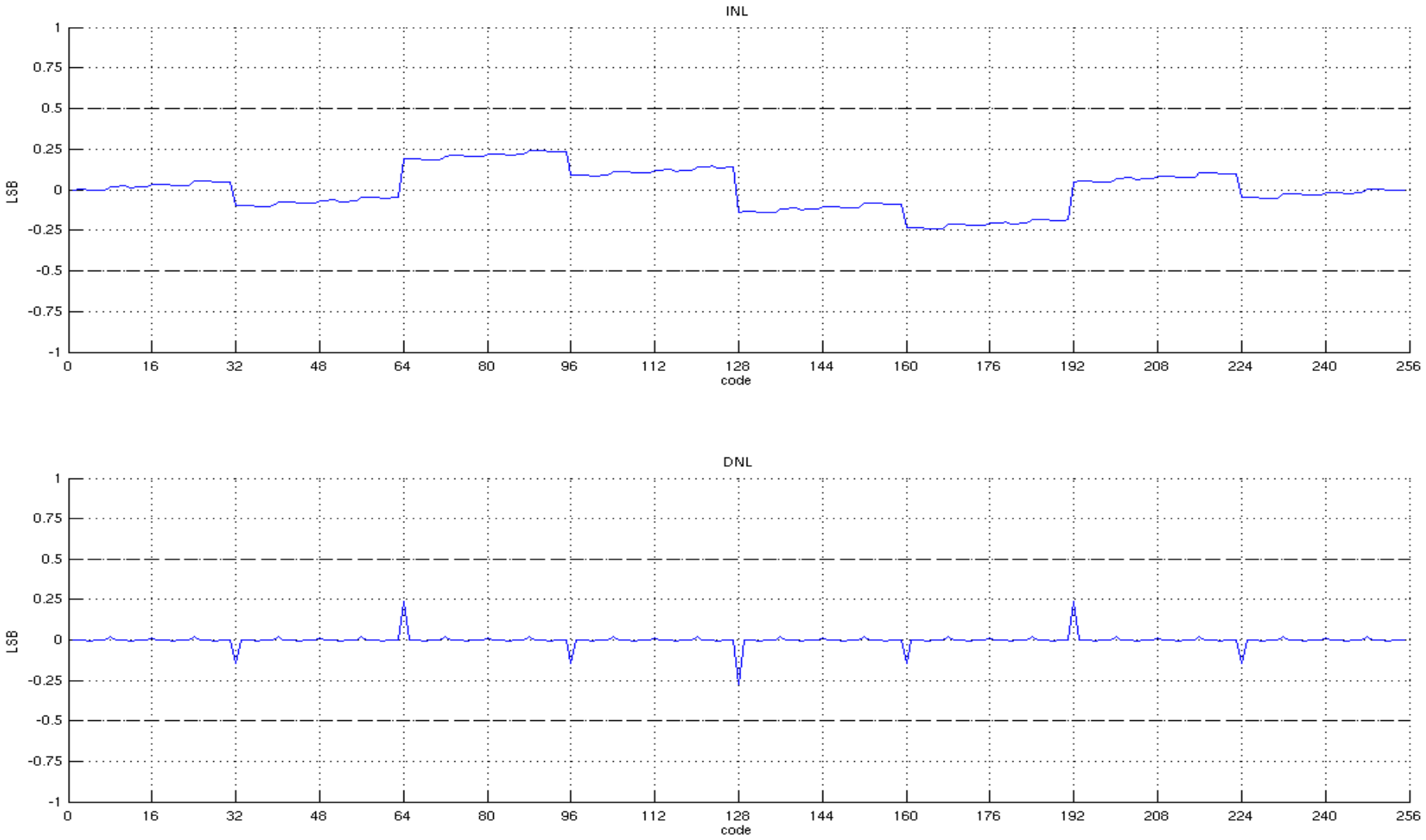


Figura 11.6: INL/DNL del calDAC

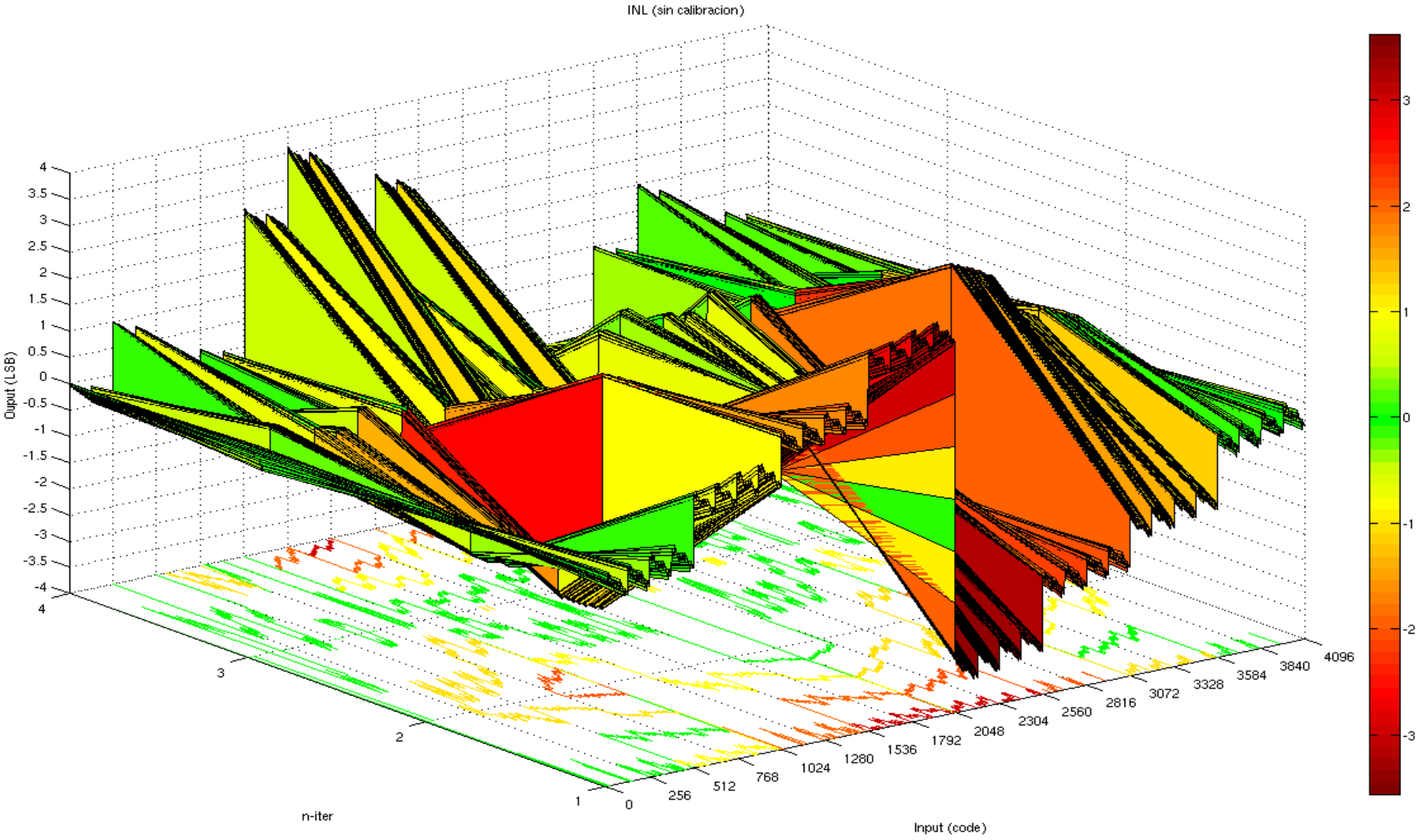


Figura 11.7: INL del ADC sin calibración (para distintos errores de matching del DAC)

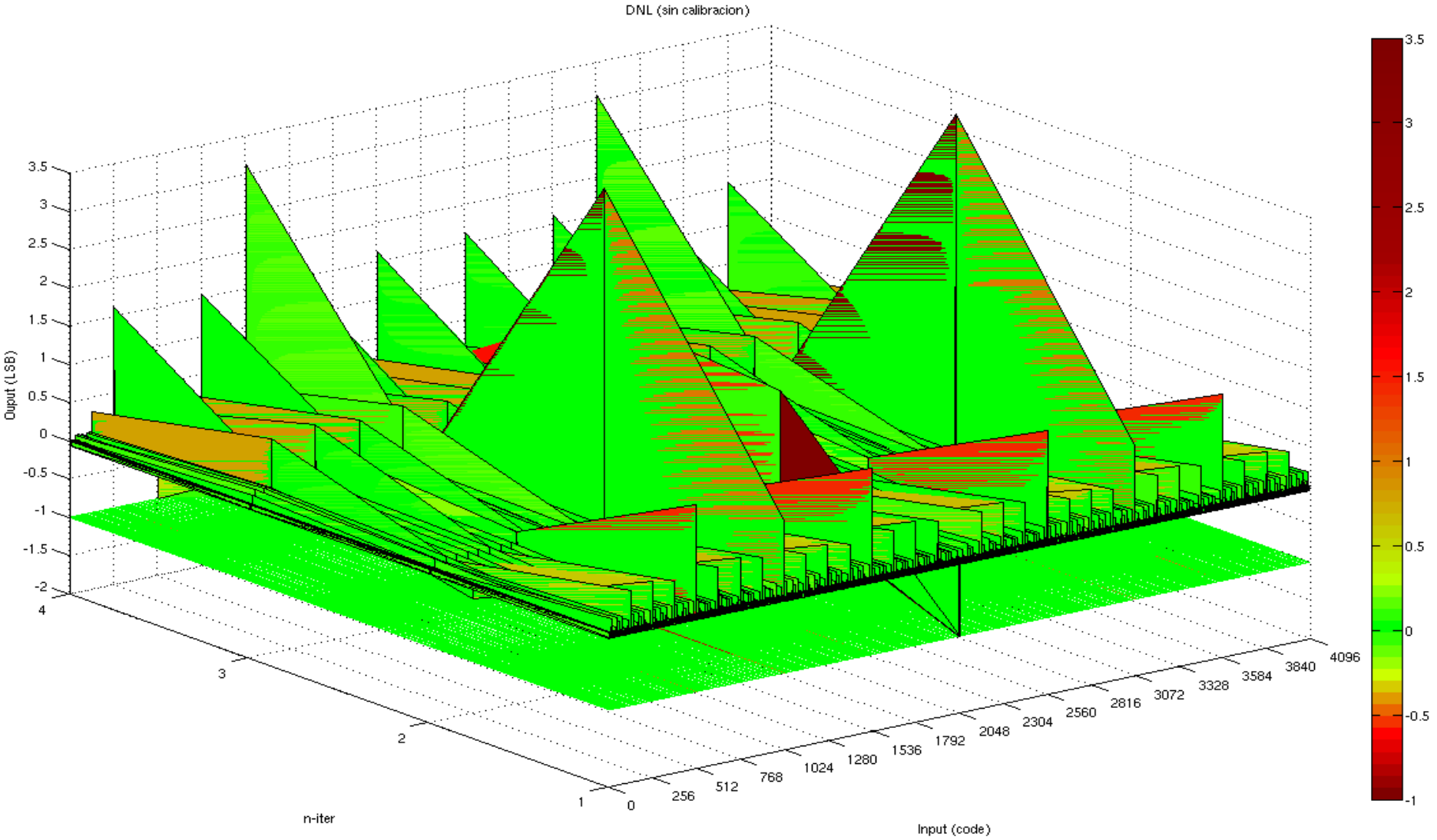


Figura 11.8: DNL del ADC sin calibración (para distintos errores de matching del DAC)

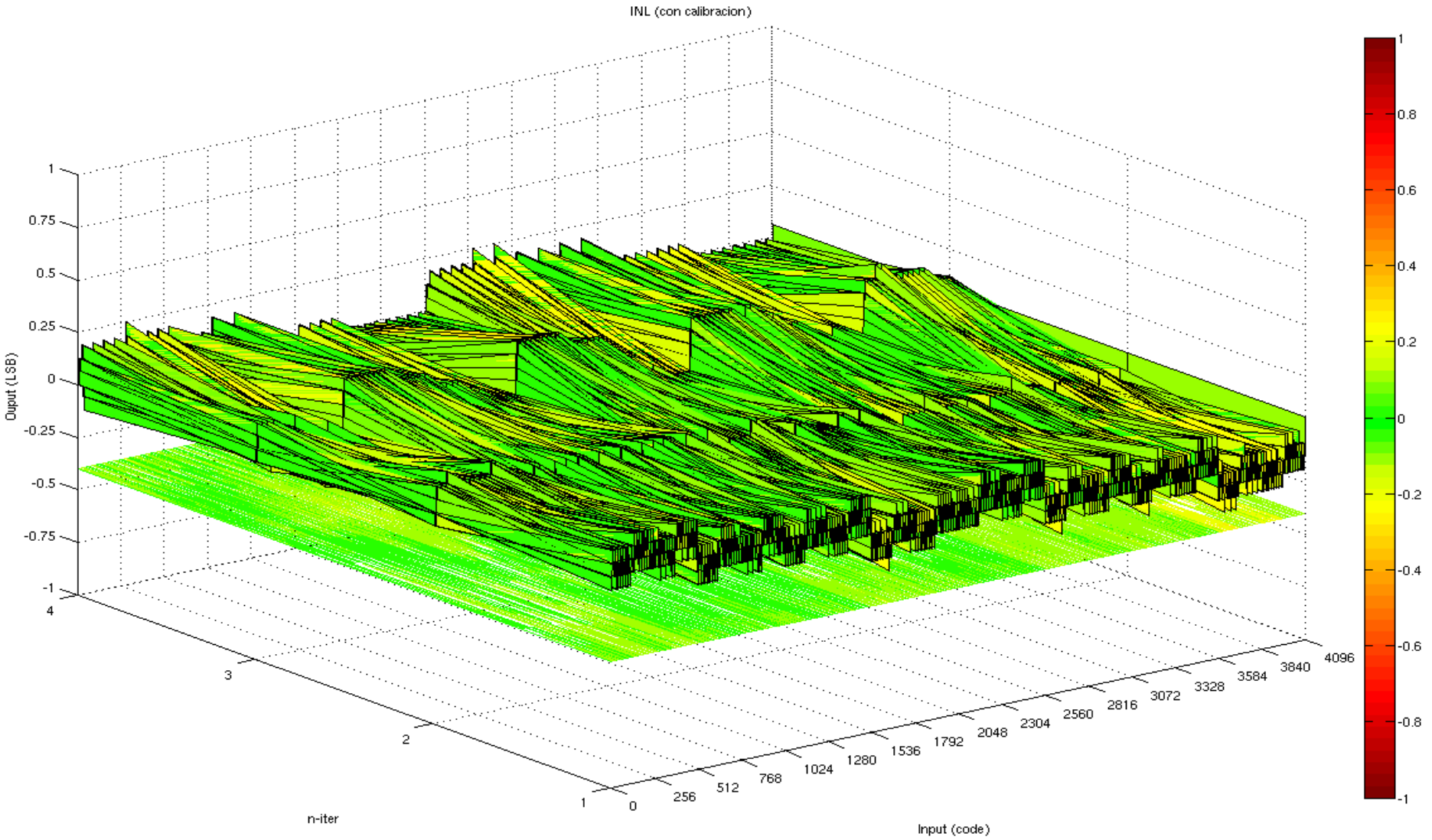


Figura 11.9: INL del ADC con calibración (para distintos errores de matching del DAC)

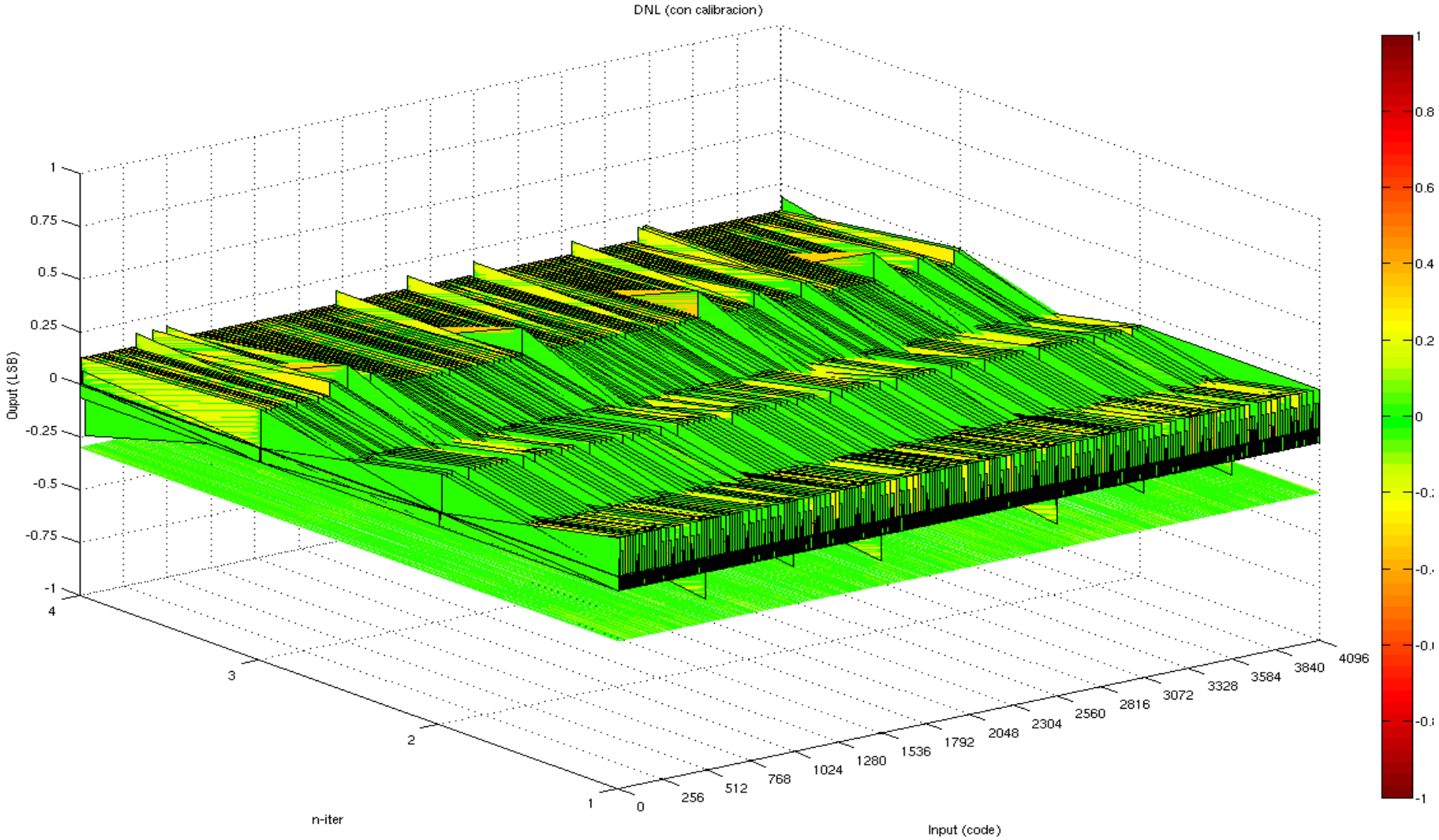


Figura 11.10: DNL del ADC con calibración (para distintos errores de matching del DAC)

CAPÍTULO 12

Conclusiones

Antes de concluir el informe sobre el Proyecto Fin de Carrera: *diseño de un convertidor analógico-digital de aproximaciones sucesivas de bajo consumo y área reducida*, en este último capítulo se recogen los aspectos más relevantes del mismo, así como los resultados finales logrados y las mayores dificultades encontradas a la hora de su realización.

En las siguientes líneas se hace un breve resumen de todo el trabajo realizado a lo largo del proyecto, partiendo de los objetivos iniciales definidos y llegando hasta cada una de las soluciones finales adoptadas. Además, se dejan planteados ciertos aspectos del diseño que han quedado abiertos, definiendo de esta forma posibles líneas de trabajo a seguir en un futuro.

12.1 RESULTADOS

Al término de este Proyecto Fin de Carrera el trabajo realizado ha sido el siguiente:

- Estudio teórico de los convertidores analógico-digital genéricos: base teórica de funcionamiento, características más relevantes, posibles arquitecturas para su implementación y parámetros de calidad que los definen.

- Estudio del estado del arte de los convertidores ADC de aproximaciones sucesivas: principio de funcionamiento, ventajas e inconvenientes y problemática asociada al *matching* del DAC.

- Implementación de una topología *fully-differential* de 12 bits para un SAR ADC. Simulaciones de alto nivel a partir de modelos *Verilog-A/Verilog-AMS*.

- Implementación del DAC capacitivo de 12 bits para la topología de SAR ADC diseñada: estudio de distintas estructuras y técnicas para optimizar el área sin comprometer en exceso el *matching* entre capacidades. Elaboración del *layout* del DAC y evaluación de su linealidad.

- Estudio, diseño e implementación de un sistema de calibración para compensar los errores de linealidad del DAC generados por los problemas de *matching* de sus capacidades.

- Descripción *Verilog* y síntesis del bloque digital de control para realizar las labores de calibración del DAC y ejecución del algoritmo de búsqueda SAR.

- Estudio, diseño e implementación del comparador requerido para el SAR ADC: implementación de un sistema de cancelación de *offset* y evaluación de sus prestaciones.

- Puesta en conjunto de todos los bloques diseñados, simulaciones *toplevel* y evaluación de prestaciones.

Tras el trabajo realizado, el resultado obtenido ha sido la consecución de un SAR ADC de 12 bits, con un *throughput* de 10.41kSPS, y un área y consumo estimados de 0.386 mm^2 y 28.79µA respectivamente.

Comparando los resultados logrados con las especificaciones que se imponían al iniciarse este trabajo, se puede afirmar que los objetivos han sido alcanzados. Véase la siguiente tabla comparativa que lo justifica:

Tipo de convertidor	SAR	SAR
Topología	*Fully-differential*	*Fully-differential*
Resolución	12 bits	12 bits
Throughput	10kSPS	10.41kSPS
Consumo	~30µA	28.79µA
Área	~0.4mm^2	0.386mm^2 *****

***** Área total incluyendo el rutado.

Tabla 12.1: Resultados obtenidos

Como se describe más arriba, al abordar este diseño se ha considerado necesario el uso de un DAC

calibrado para poder afrontar las especificaciones de área. El diseño del sistema de calibración junto con el de la estructura del DAC ha supuesto el mayor esfuerzo a la hora de realizar este proyecto; ha sido necesario llegar a un compromiso entre el área del DAC y los errores de linealidad cometidos por éste, que serán compensados por el sistema de calibración.

Además de todo el trabajo técnico realizado y los resultados objetivos obtenidos, este Proyecto Fin de Carrera a supuesto también, a nivel personal, una introducción a la forma de trabajo en el ámbito de la empresa privada, la familiarización con las herramientas de diseño y métodos empleados en las empresas del sector, y una primera toma de contacto en lo que puede ser un trabajo como ingeniero.

12.2 TRABAJO FUTURO

Una de las líneas de trabajo que quedan abiertas tras la finalización de este proyecto, recae sobre el sistema de calibración. Como se describió en el capítulo correspondiente, durante este proyecto se ha estudiado y puesto en práctica un sistema de calibración para compensar las no linealidades del DAC, lo que supone, como se ha visto, una gran mejoría en cuanto a área, pues ha sido posible reducir significativamente el área del DAC a utilizar. El inconveniente de este sistema es que requiere de una lógica adicional para el cálculo de los códigos de calibración, la cual, en este proyecto, se ha situado fuera del chip, pues se ha optado por implementar una calibración estática.
Un posible trabajo futuro a realizar es, el de migrar la calibración actual hacia una calibración dinámica, incluyendo la lógica de cálculo de los códigos de calibración dentro del chip; evaluar las ventajas e inconvenientes que ello conlleva y comparar ambos tipos de calibración.

Desde el punto de vista de la linealidad del propio DAC, sin tener en cuenta la calibración, también sería interesante experimentar con otro tipo de estructuras de DAC capacitivo que permitiesen conseguir mejoras, es el caso de las estructuras con segmentación. A grandes rasgos, la segmentación es una técnica que permite aumentar el *matching* entre las capacidades de mayor tamaño del DAC, utilizando para ello una distribución especial de las capacidades dentro del *array* junto a una codificación *thermometrica*.

Y para concluir, señalar que quedarían pendientes por realizar otras tareas auxiliares hasta poder llegar a la fabricación de este diseño en una MPW, y posterioremente testear su funcionamiento real en el laboratorio.

Referencias

[1] Franco Maloberti, *"Data Converters"*. Springer.

[2] Behzad Razavi. "*Principles of Data Conversion System Design"*. IEEE PRESS.

[3] Paul G. A Jespers. *"Integrated Converters: D to A and A to D Architectures, Analysis and Simulation*". Oxford.

[4] Phillip Allen, Douglas R. Holberg. *"CMOS Analog Circuit Design".* Oxford University Press.

[5] Dan Fitzpatrick, Ira Miller. *"Analog Behavioral Modeling with the Verilog-A Language"*. Springer.

[6] Ken Kundert, Olaf Zinke. *"The Designer's Guide to Verilog-AMS"*. Springer.

[7] Yasuhide Kuramochi, Akira Matsuzawa, Masayuki Kawabata. *"A 0.05-mm2 110uW 10b Self-Calibrating Successive Approximation ADC Core in 0.18um CMOS".* Paper TOKYOTECH.

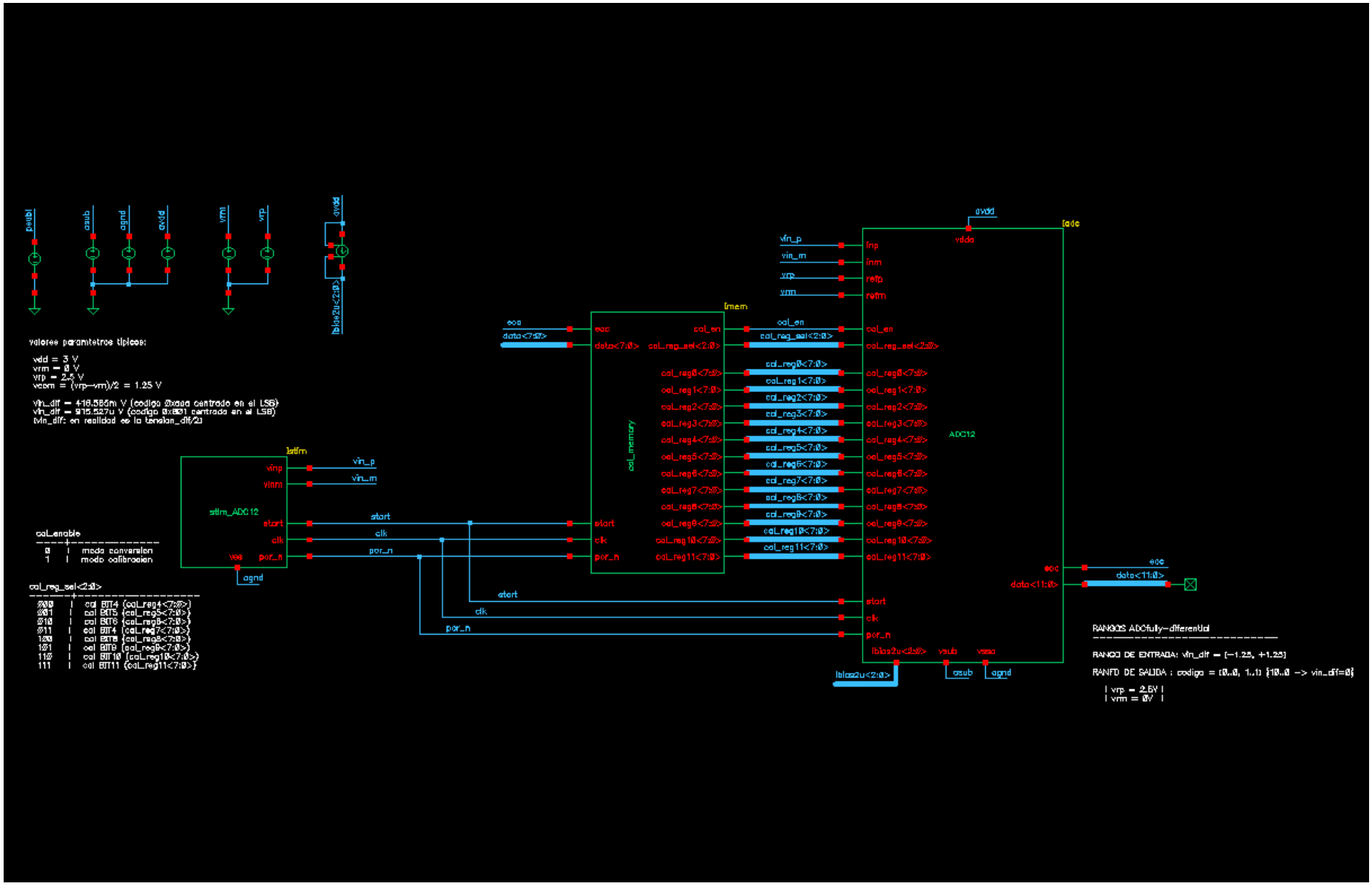
[8] Mahmoud Fawzy Wagday. *"Various Calibration Algorithms for Self-Calibrating A/D Converters"*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, VOL. *31*, NO. *I* , JULY 1990.

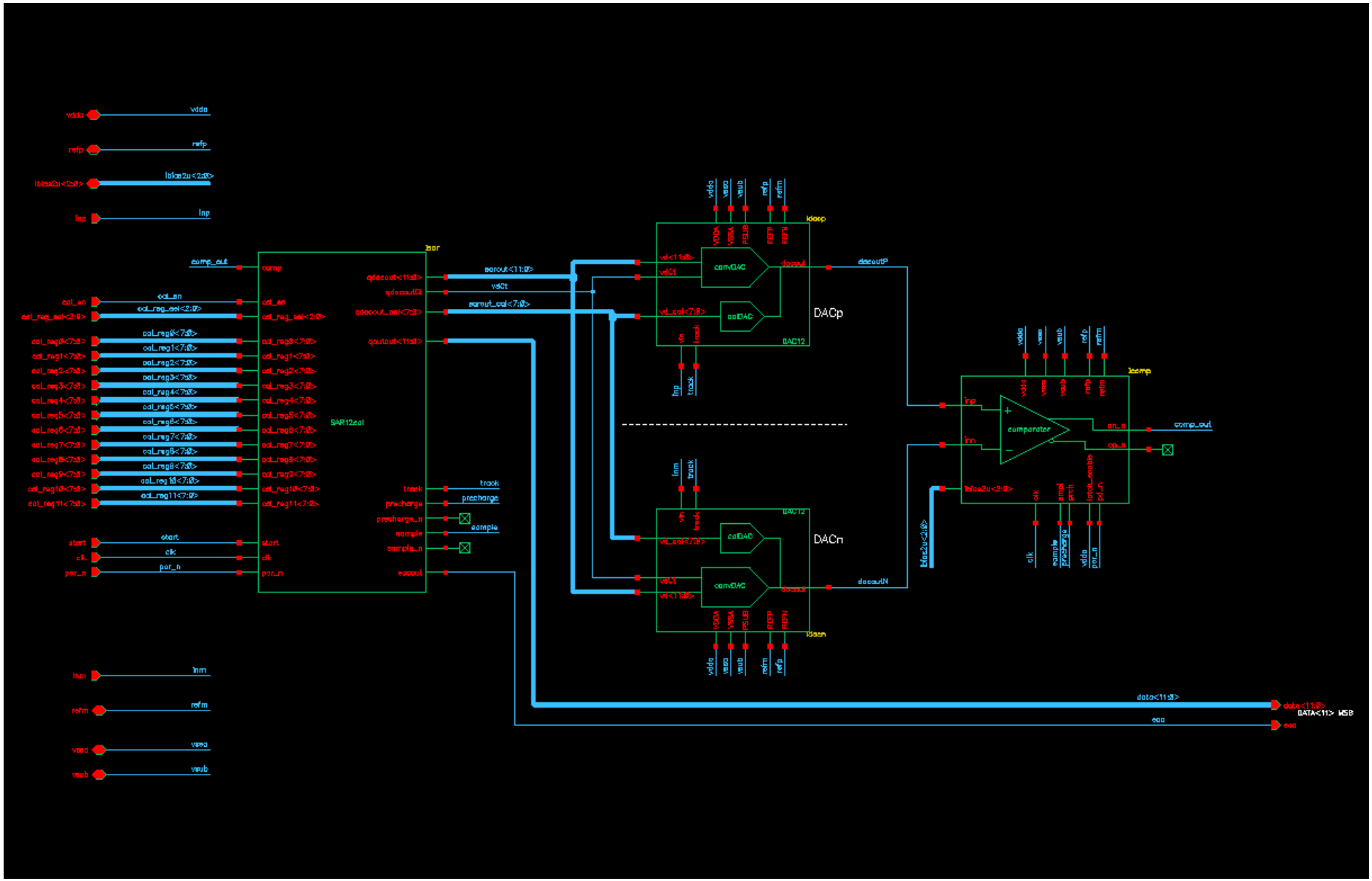
[9] Hae-Seung Lee, David A. Hodges, Paul R.Gray. *"A Self-Calibrating 15bit CMOS A/D Converter"*. IEEE JOURNAL OF SOLID STATE CIRCUITS, VOL SC-19, NO 6 DECEMBER 1984.

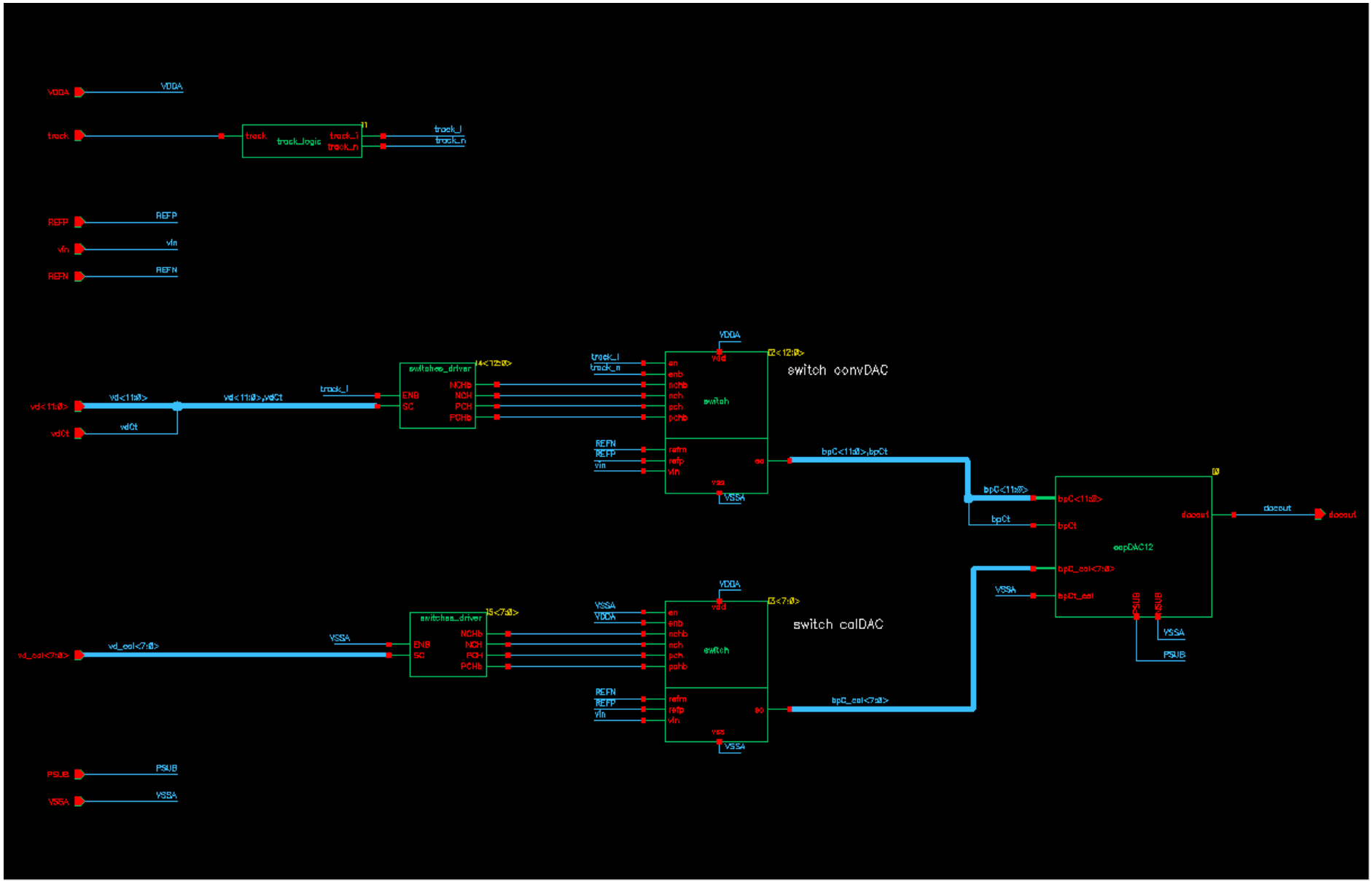
ANEXO A. Esquemáticos

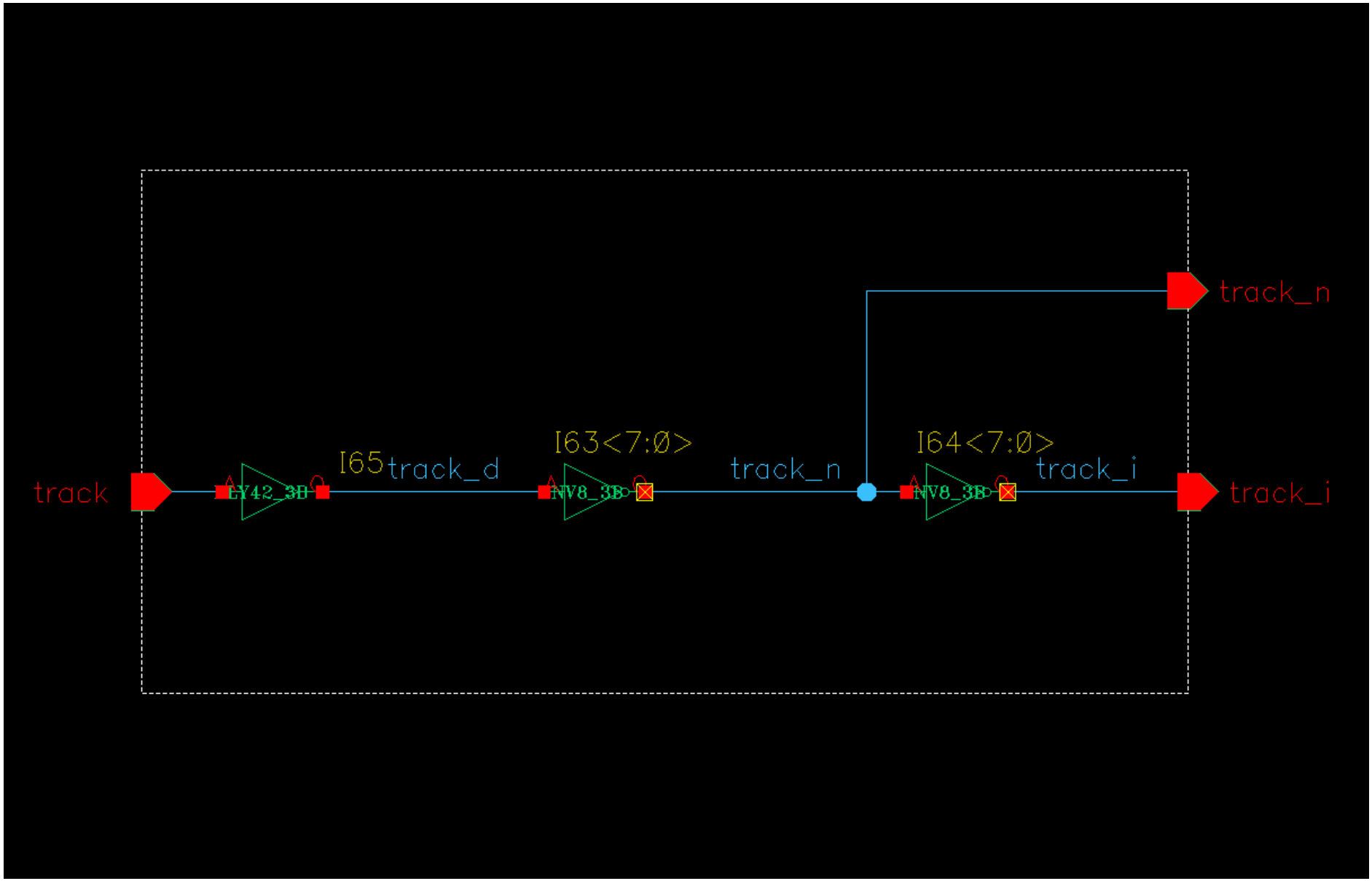
En el presente anexo se incluyen todos los esquemáticos elaborados para este proyecto, ordenados de mayor a menor nivel dentro de la jerarquía.

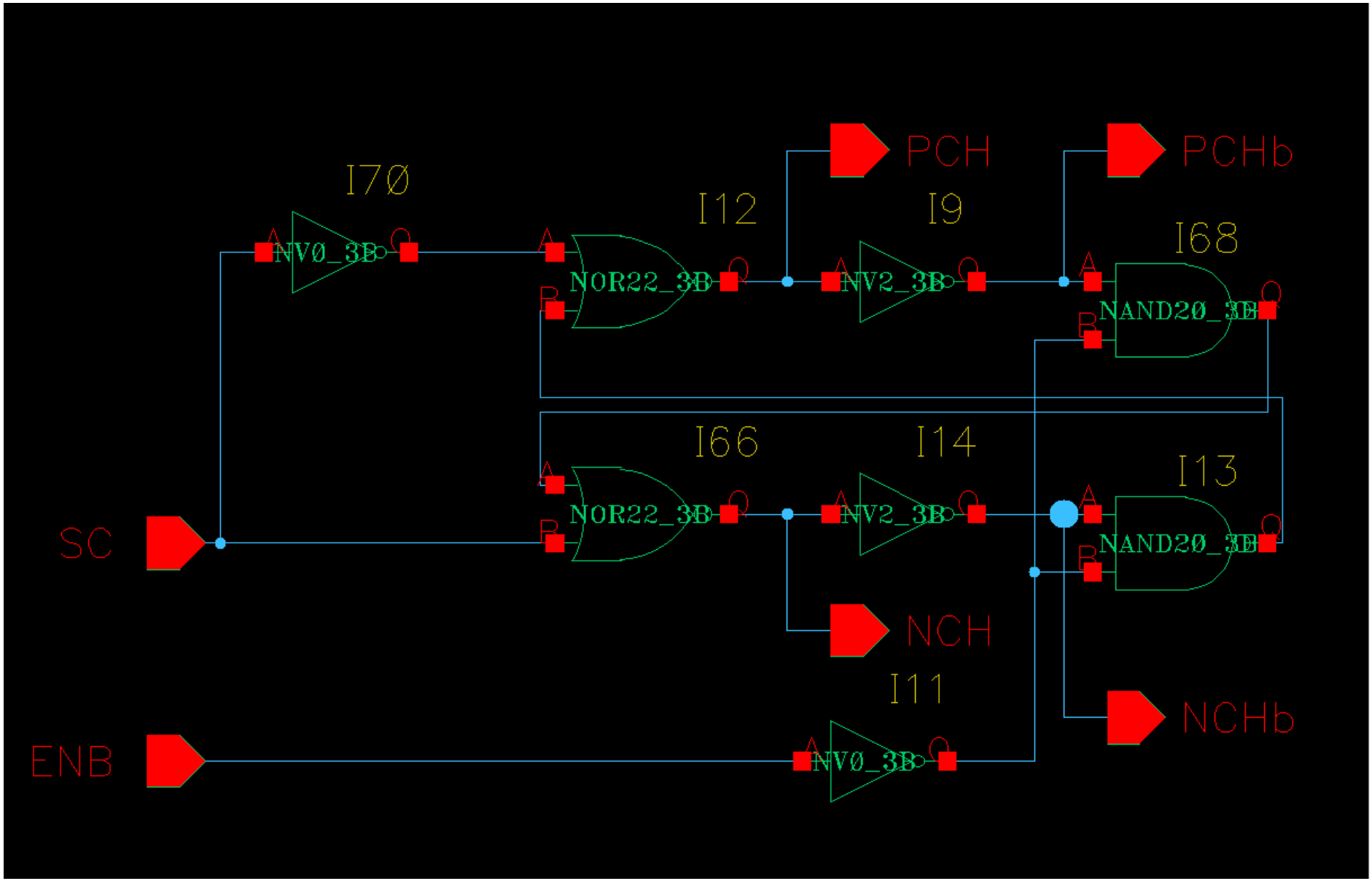
1. tb_ADC12.sch
2. ADC12.sch
3. DAC12.sch
4. track_logic.sch
5. switches_driver.sch
6. switch.sch
7. capDAC12_sim.sch
8. capDAC12_simp.sch
9. comparator.sch
10. switch_comp.sch
11. fd_opi.sch
12. comcap_control2.sch

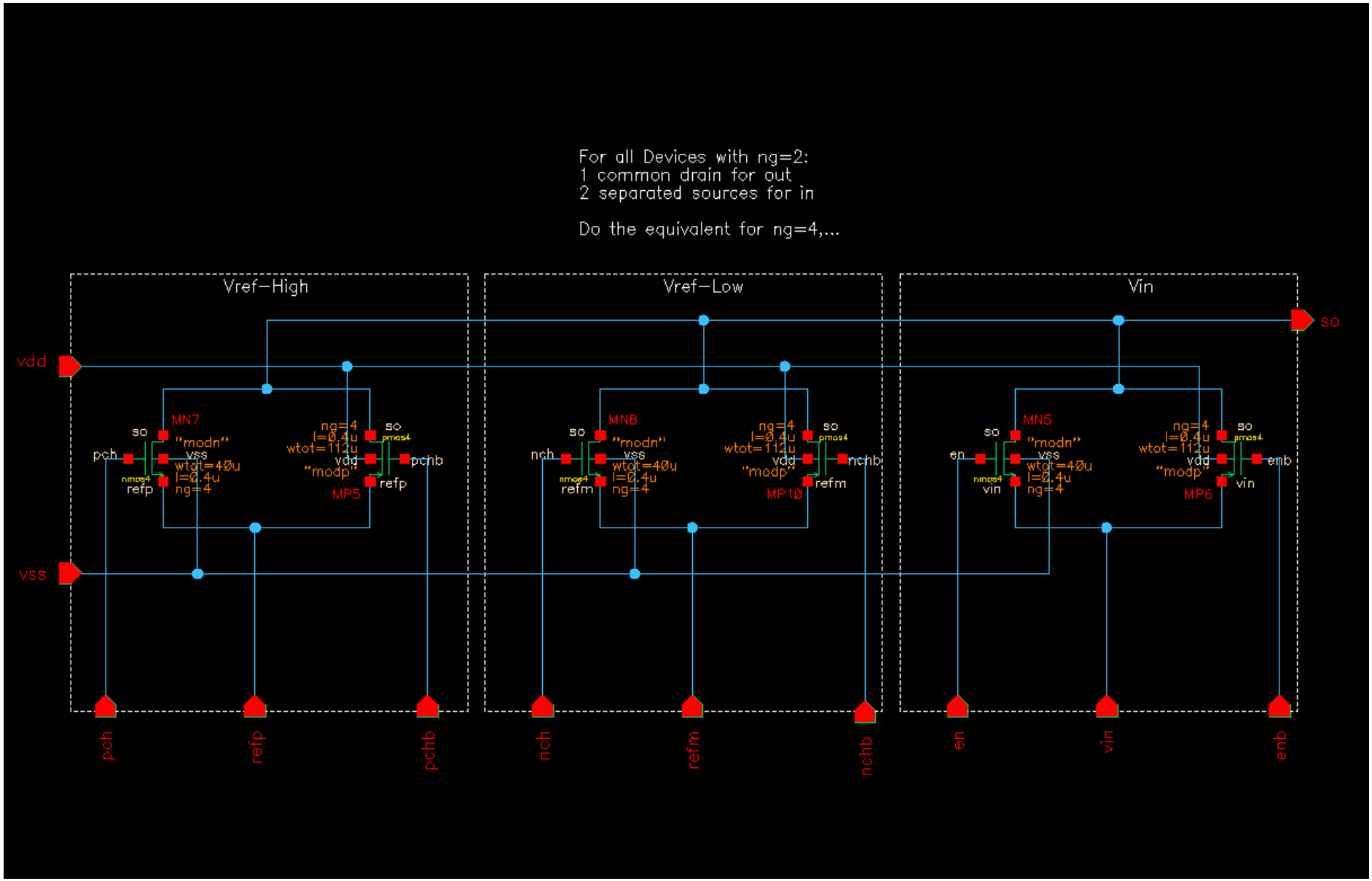

valores parametros tipicos:
vdd = 3 V
vrm = 0 V
vrp = 2.5 V
vcom = (vrp-vrm)/2 = 1.25 V
vin_dif = 416.585m V (codigo 0x666 centrado en el LSB)
vin_dif = 915.527u V (codigo 0x801 centrado en el LSB)
(vin_dif: en realidad es la tension_dif/2)
cal_enable
0 | modo conversion
1 | modo calibracion
cal_reg_sel<2:0>
000 | cal BIT4 (cal_reg4<7:0>)
001 | cal BIT5 (cal_reg5<7:0>)
010 | cal BIT6 (cal_reg6<7:0>)
011 | cal BIT7 (cal_reg7<7:0>)
100 | cal BIT8 (cal_reg8<7:0>)
101 | cal BIT9 (cal_reg9<7:0>)
110 | cal BIT10 (cal_reg10<7:0>)
111 | cal BIT11 (cal_reg11<7:0>)
stim_ADC12
cal_memory
ADC12
RANGOS ADC fully-diferential
RANGO DE ENTRADA: vin_dif = [-1.25, +1.25]
RANGO DE SALIDA : codigo = [0..0, 1..1] [10..0 -> vin_dif=0]
| vrp = 2.5V |
| vrm = 0V |

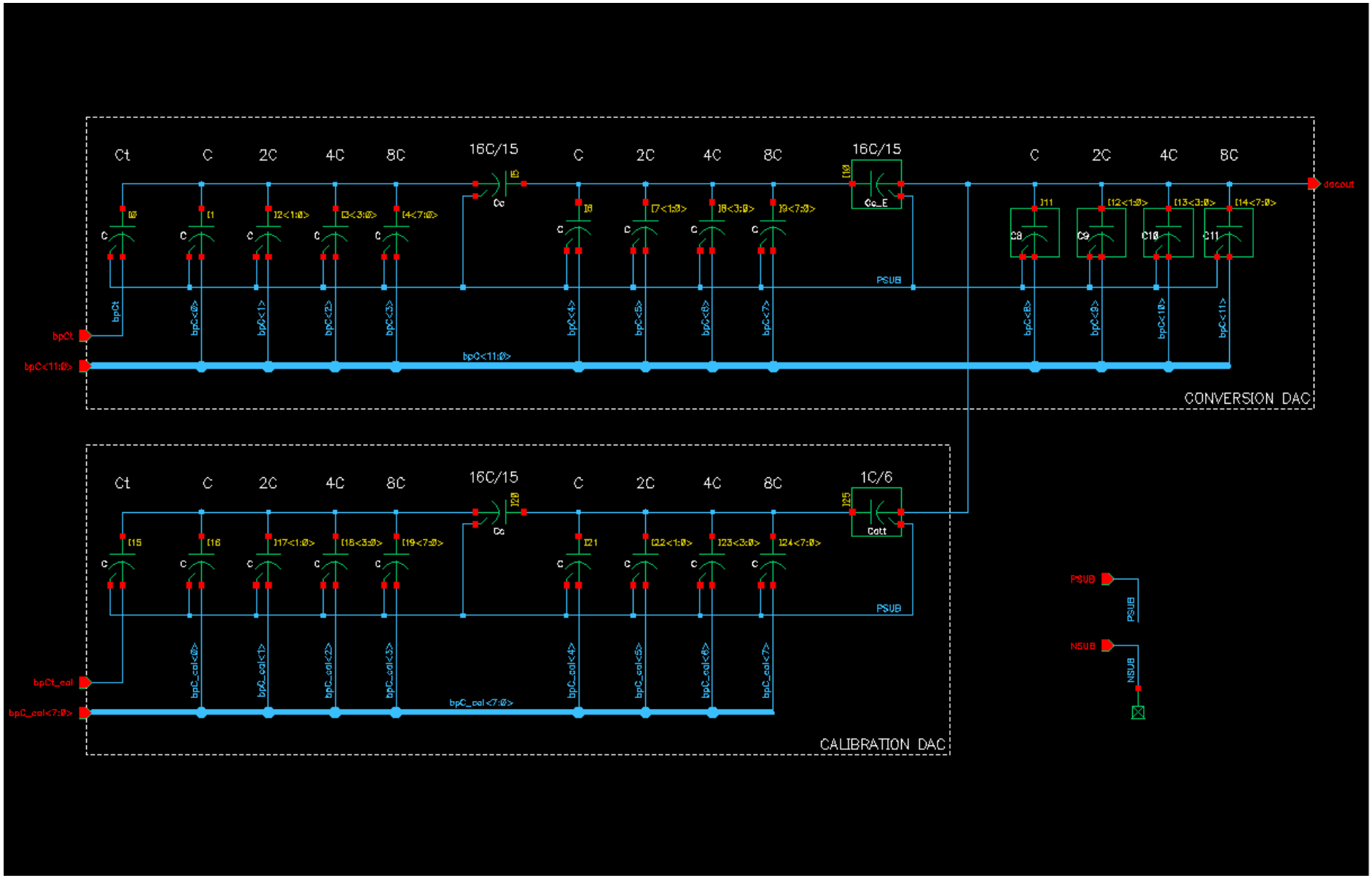

vddd
refp
Ibias2u<2:0>
Inp
comp_out
cal_en
cal_reg_sel<2:0>
cal_reg0<7:0>
cal_reg1<7:0>
cal_reg2<7:0>
cal_reg3<7:0>
cal_reg4<7:0>
cal_reg5<7:0>
cal_reg6<7:0>
cal_reg7<7:0>
cal_reg8<7:0>
cal_reg9<7:0>
cal_reg10<7:0>
cal_reg11<7:0>
start
clk
por_n
Inm
refm
vssa
vaub
SAR12cal
sar
sarout<11:0>
vdct
sarout_cal<7:0>
qdacout<11:0>
qdacout_cal<7:0>
qoutout<11:0>
track
precharge
precharge_n
sample
sample_n
eocout
vdda
vssa
vaub
refp
refm
VDDA
VSSA
PSUB
REFP
REFN
vd<11:8>
vdct
comDAC
dacoutP
vd_cal<7:0>
calDAC
DAC12
DACp
kdac
Inp
track
Inm
DACn
dacoutN
kdacn
kcomp
comparator
inp
inn
on_n
op_n
comp_out
Ibias2u<2:0>
clk
sample
precharge
track_enable
por_n
data<11:0>
DATA<11> MSB
eoc

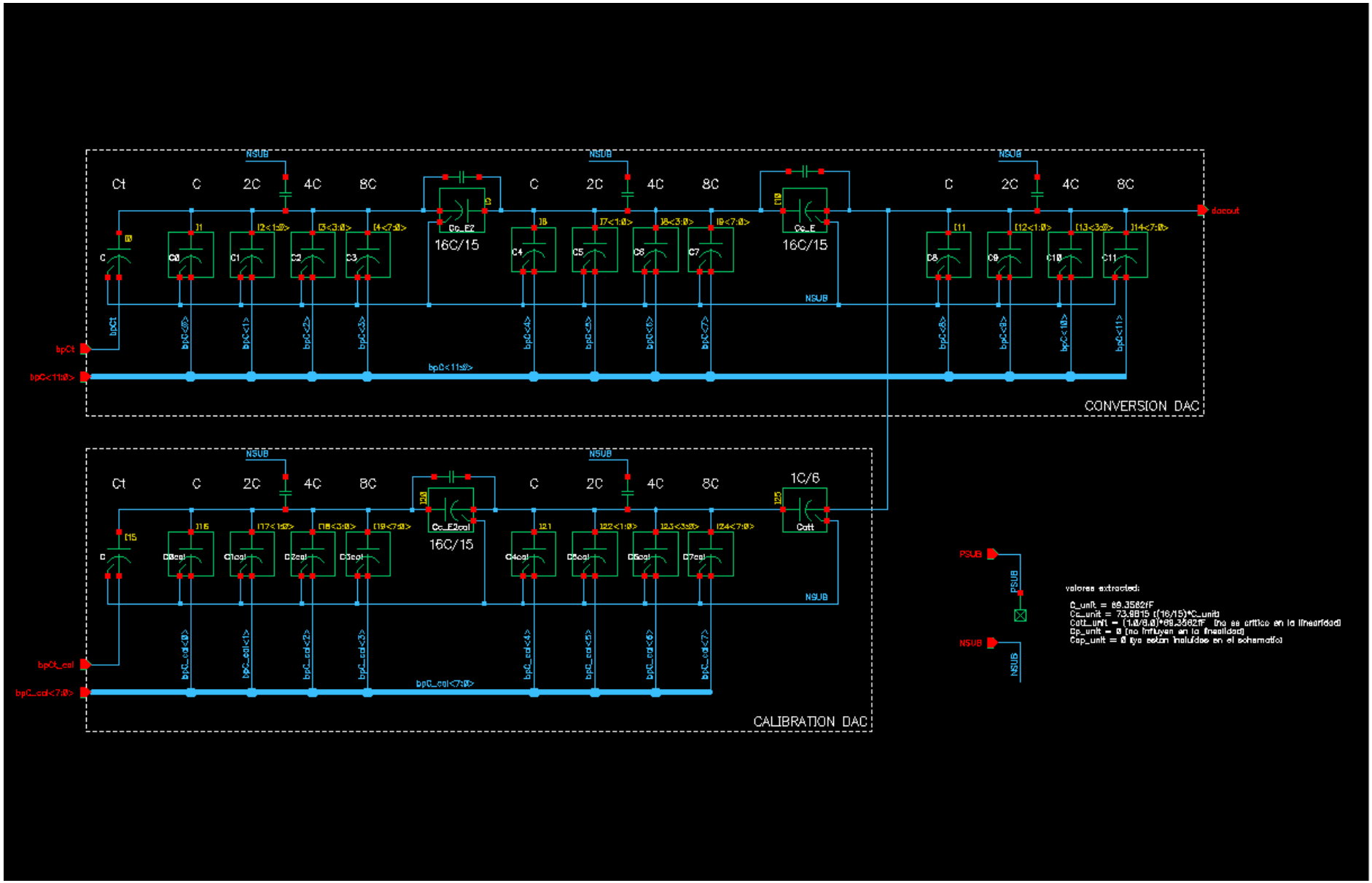

VDDA
track
track_logic
track_l
track_n
REFP
vin
REFN
switches_driver
ENB
SC
NCHb
NCH
PCH
PCHb
switch convDAC
switch calDAC
switch
vdd
en
enb
nchb
nch
pch
pchb
refm
refp
vin
vss
bpC<11:0>,bpCt
bpC<11:0>
bpCt
dacout
expDAC12
bpC_cal<7:0>
bpCt_cal
VSSA
PSUB

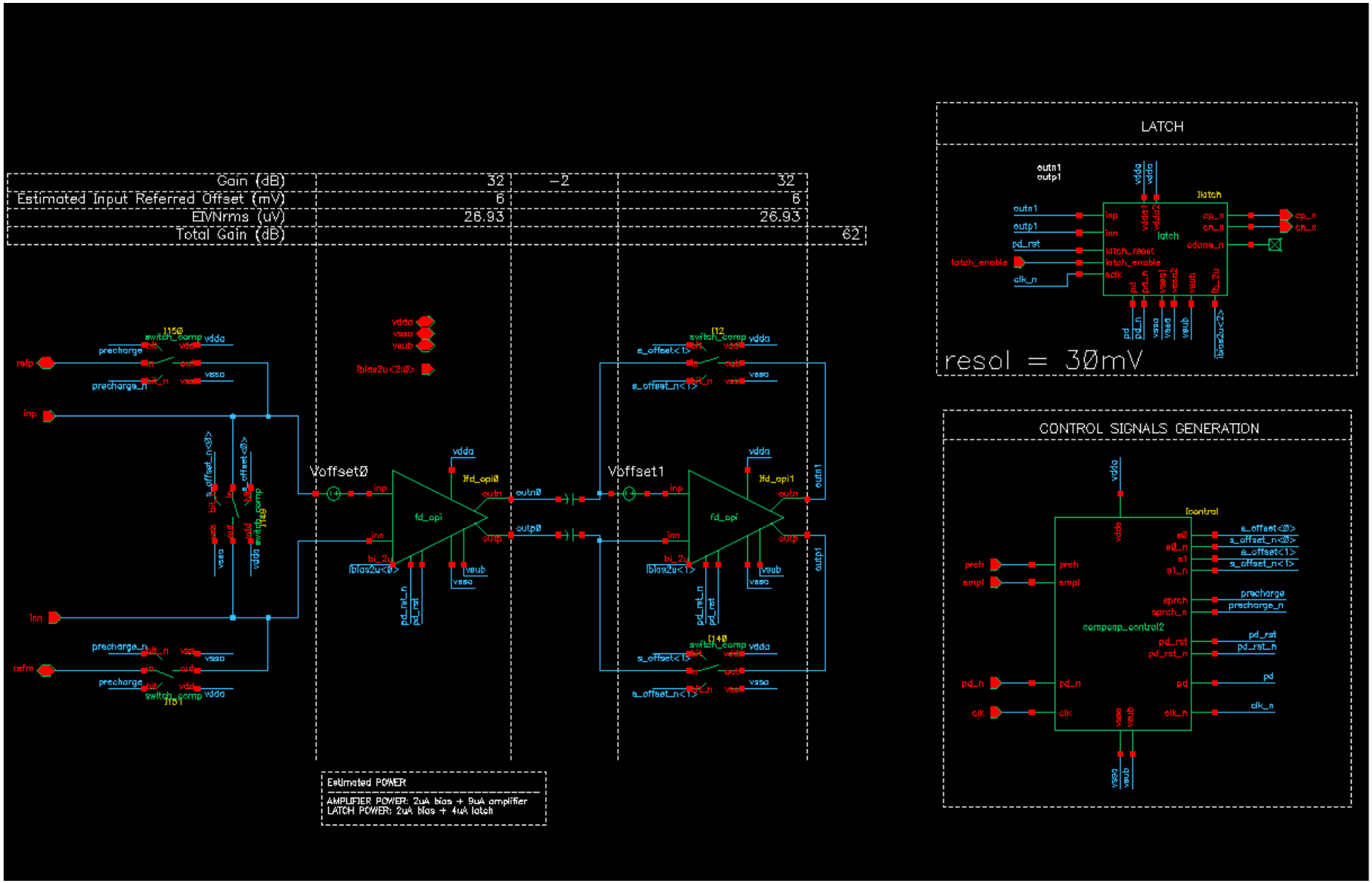

track_n
I63<7:0>
I64<7:0>
I65
track_d
track_n
track_i
track
LV42_3D
NV8_3D
NV8_3D
track_i

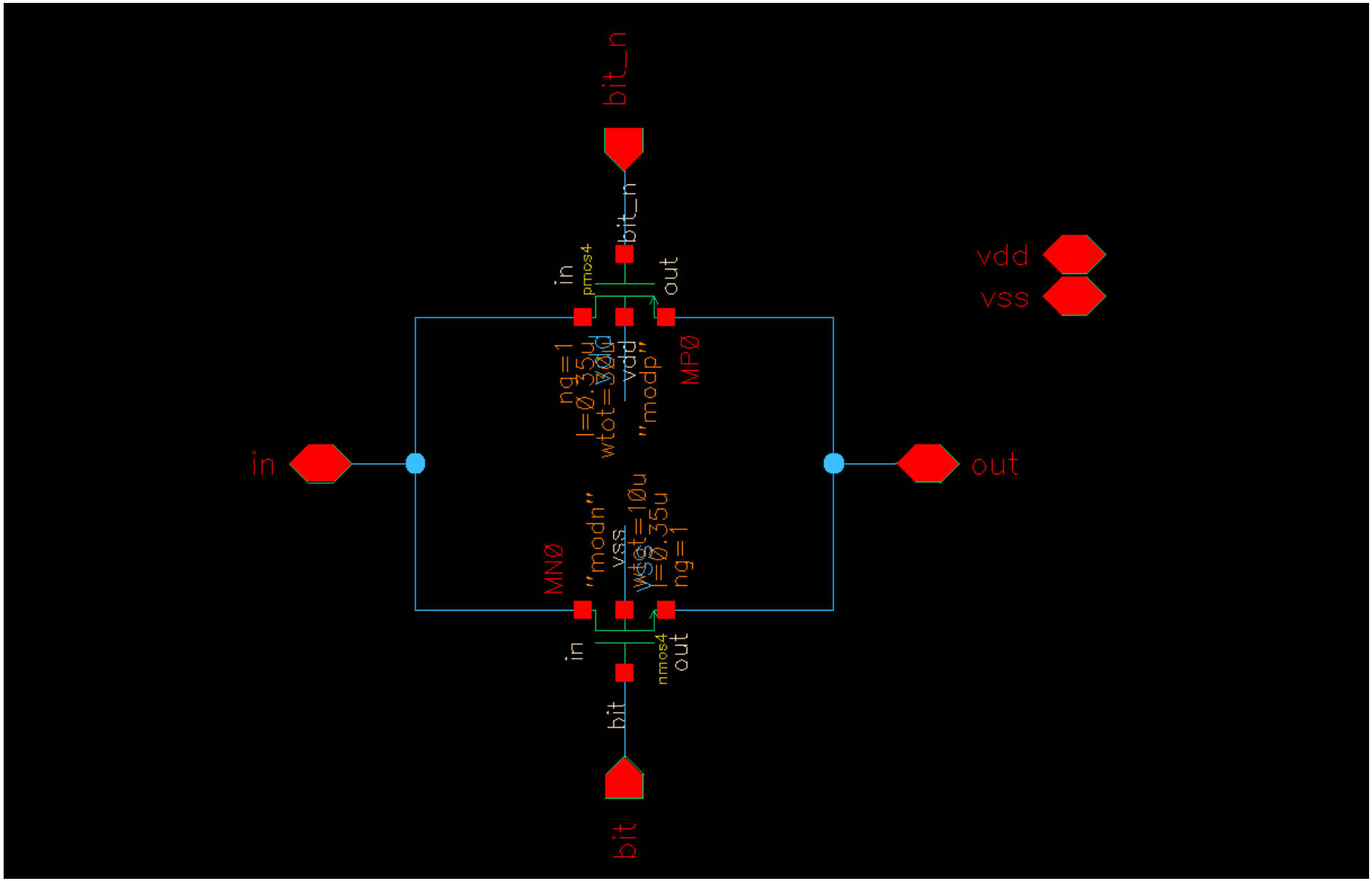

I70
NV0_3B
I12
NOR22_3B
PCH
I9
NV2_3B
PCHb
I68
NAND20_3B
I66
NOR22_3B
I14
NV2_3B
I13
NAND20_3B
SC
NCH
I11
NV0_3B
NCHb
ENB

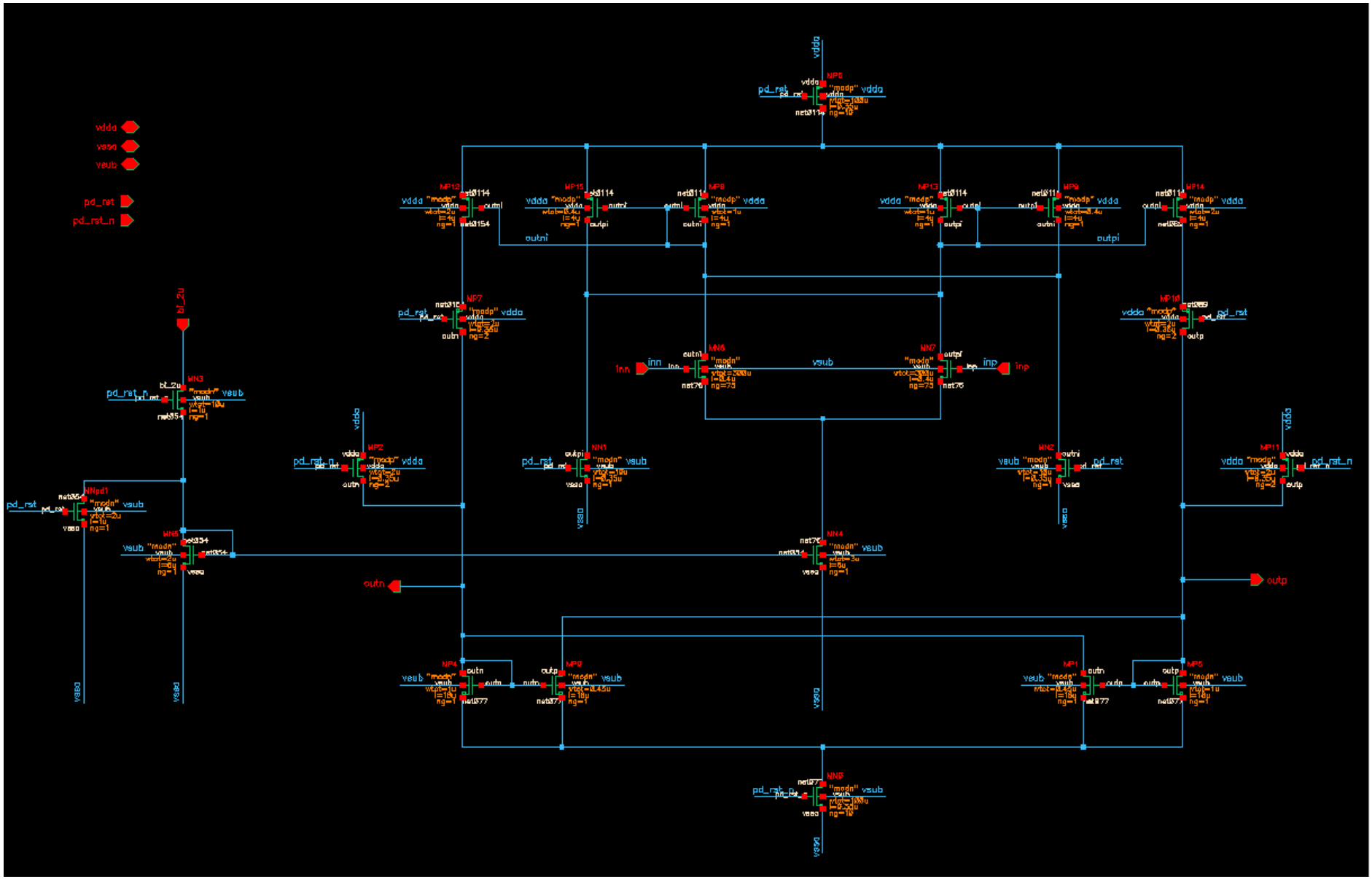

For all Devices with ng=2:
1 common drain for out
2 separated sources for in
Do the equivalent for ng=4,...
Vref-High
Vref-Low
Vin
vdd
vss
so
MN7
MP5
MN8
MP10
MN5
MP6
pch
refp
pchb
nch
refm
nchb
en
vin
enb

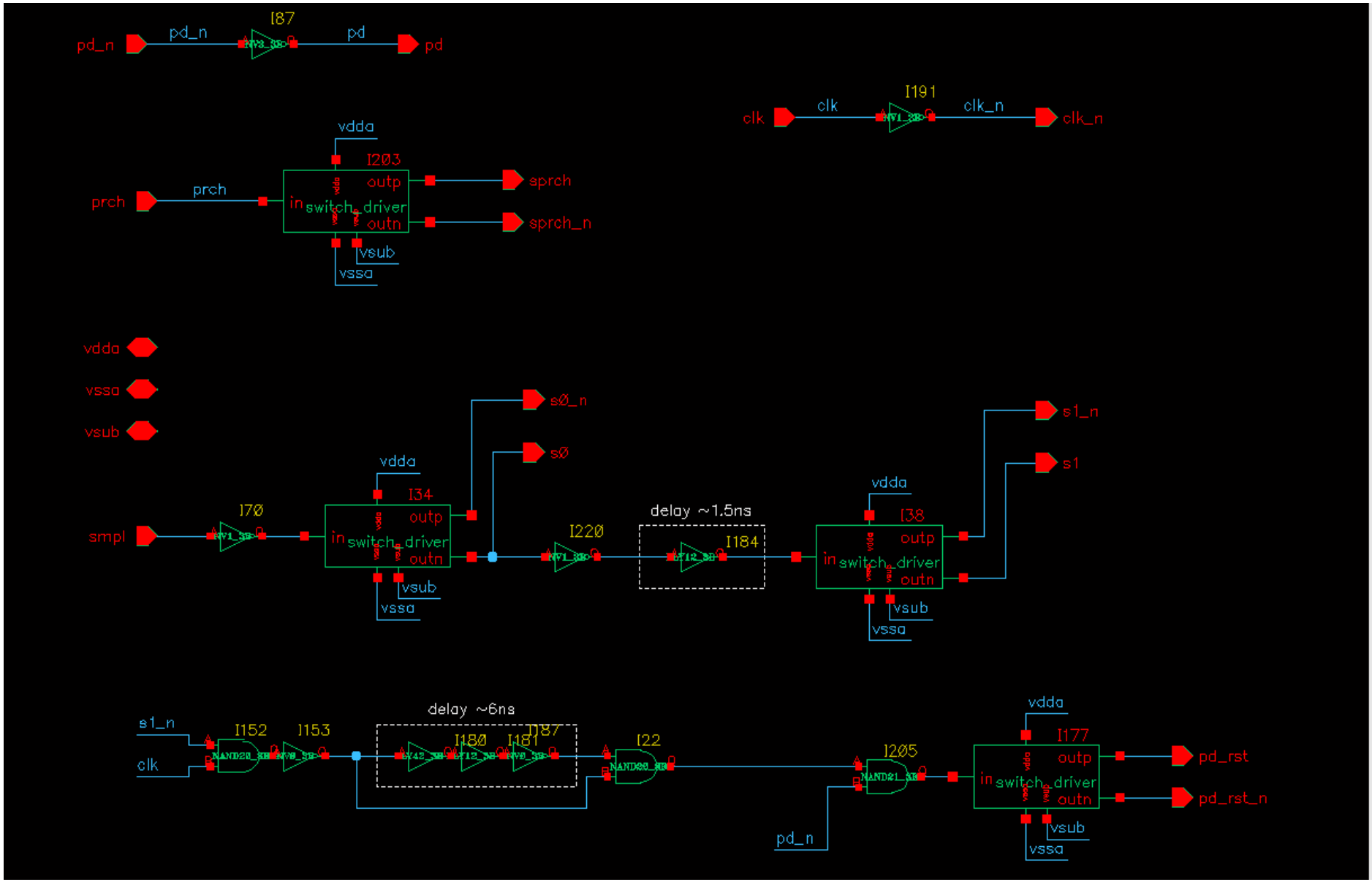

Ct
C
2C
4C
8C
16C/15
Cc
Cc_E
dacout
PSUB
bpCt
bpC<11:0>
CONVERSION DAC
1C/6
Catt
bpCt_cal
bpC_cal<7:0>
CALIBRATION DAC
NSUB


Ct
C
2C
4C
8C
16C/15
dacout
bpCt
bpC<11:0>
CONVERSION DAC
1C/6
Cc_E2col
Catt
bpCt_col
bpC_col<7:0>
CALIBRATION DAC
PSUB
NSUB
valores extracted:
C_unit = 89.3582fF
Cc_unit = 73.8815 ((16/15)*C_unit)
Catt_unit = (1.0/8.0)*89.3582fF (no es crítico en la linearidad)
Cp_unit = 0 (no influyen en la linealidad)
Csp_unit = 0 (ya están incluidos en el schematic)


Gain (dB) 32 -2 32
Estimated Input Referred Offset (mV) 6 6
EIVNrms (uV) 26.93 26.93
Total Gain (dB) 62
Voffset0
Voffset1
fd_api
LATCH
resol = 30mV
CONTROL SIGNALS GENERATION
Estimated POWER
AMPLIFIER POWER: 2uA bias + 9uA amplifier
LATCH POWER: 2uA bias + 4uA latch


bit_n
bit_n
in
pmos4
out
vdd
vss
ng=1
l=0.35u
wtot=100u
vdd
"modp"
MP0
in
out
MN0
"modn"
vss
wtot=10u
l=0.35u
ng=1
in
nmos4
out
bit
bit


vdda
vssa
vsub
pd_rst
pd_rst_n
bl_2u
inn
inp
outn
outp
outni
outpi
MP6
MP12
MP15
MP8
MP13
MP9
MP14
MP7
MP10
MN3
MN6
MN7
MP2
MN1
MP11
MN2
MNpd1
MN5
MN4
MP4
MP3
MP1
MP5
MN8

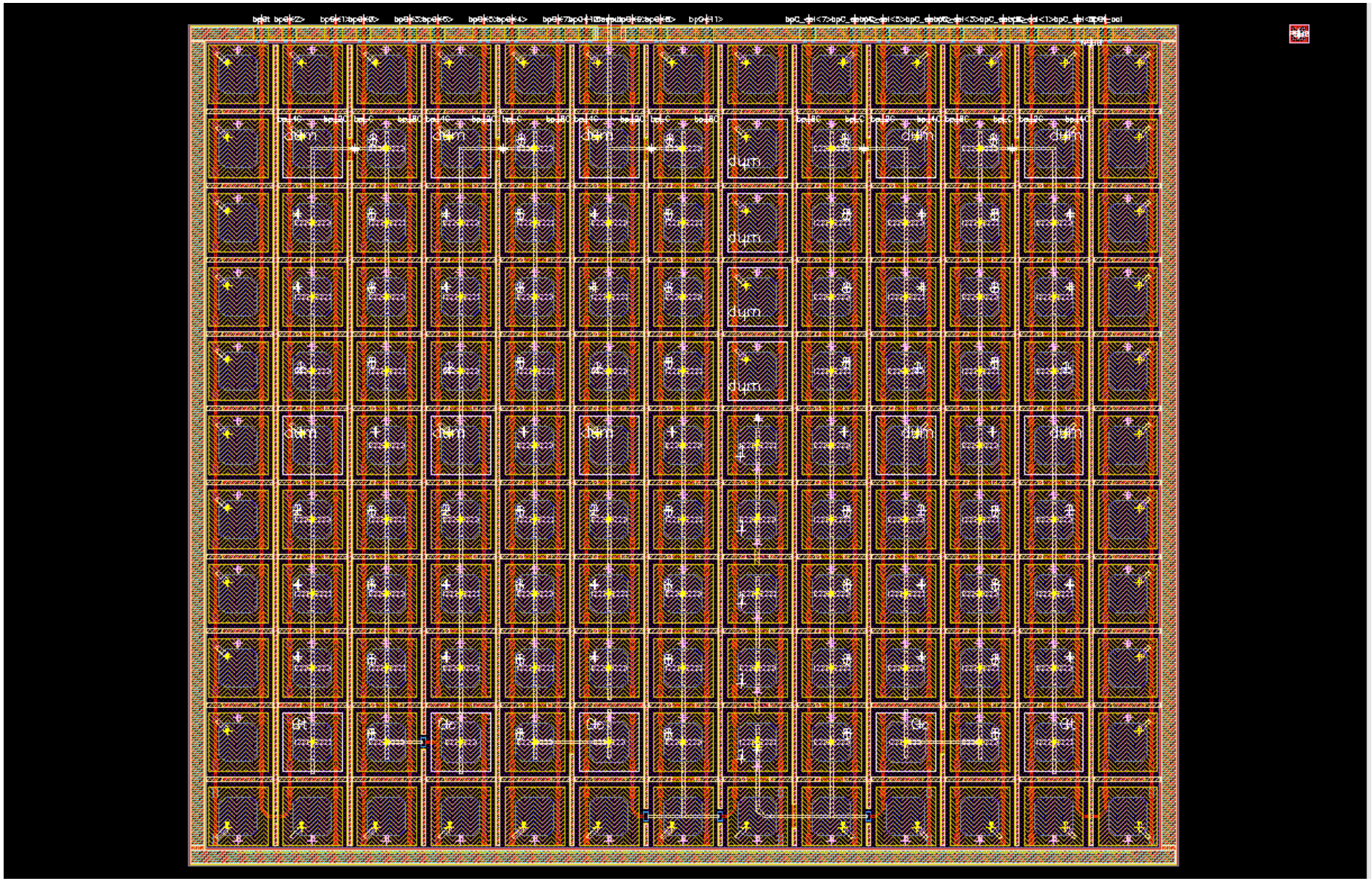

I87
pd_n
pd_n
pd
pd
I191
clk
clk
clk_n
clk_n
vdda
I203
prch
prch
in switch_driver
outp
outn
sprch
sprch_n
vsub
vssa
vdda
vssa
vsub
sØ_n
sØ
vdda
I34
I70
smpl
in switch_driver
outp
outn
I220
delay ~1.5ns
I184
vdda
I38
in switch_driver
outp
outn
s1_n
s1
vsub
vssa
vsub
vssa
delay ~6ns
s1_n
clk
I152
I153
I180
I181
I187
I22
I205
pd_n
vdda
I177
in switch_driver
outp
outn
pd_rst
pd_rst_n
vsub
vssa

ANEXO B. Layout del DAC

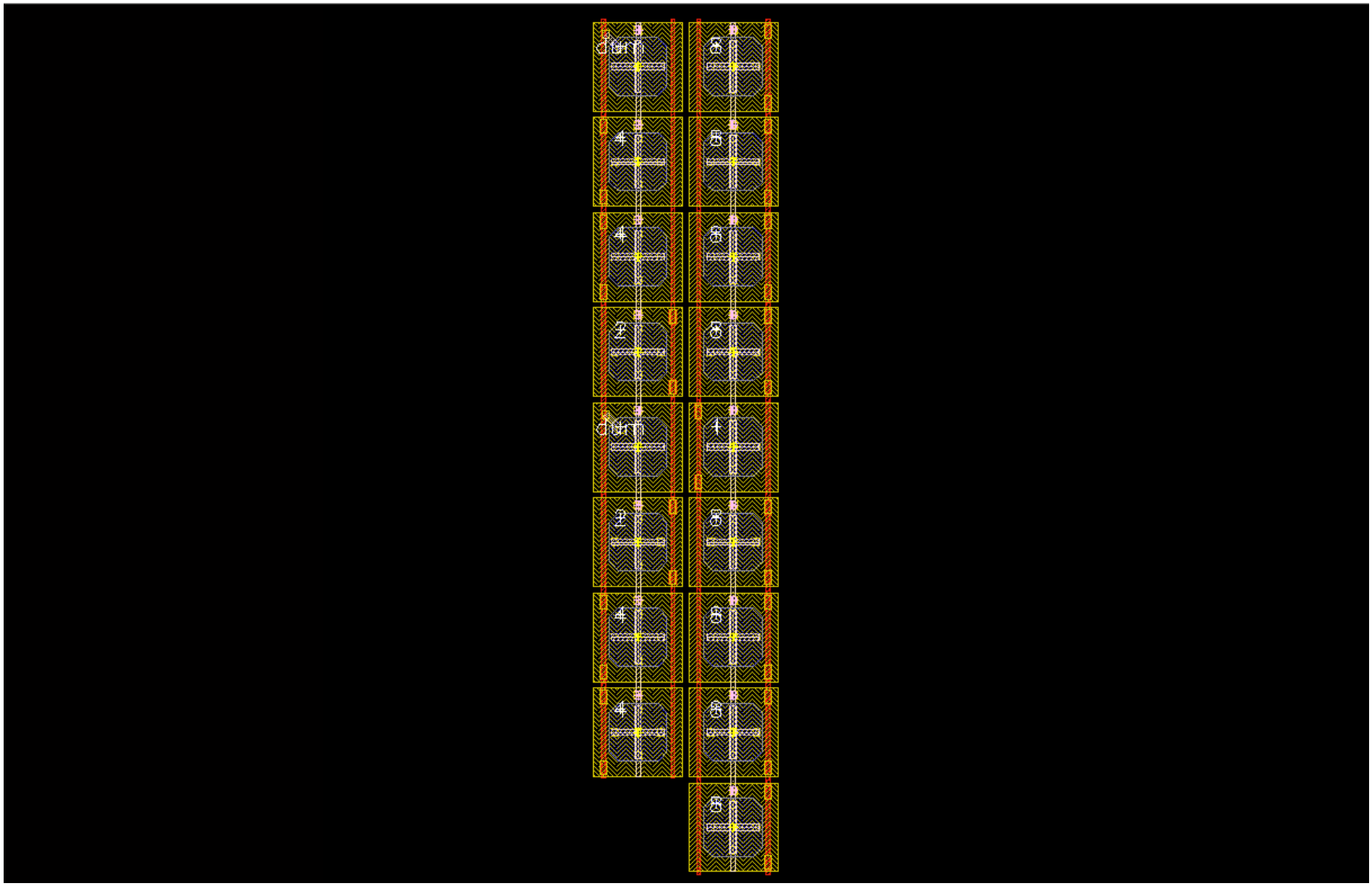
En el presente anexo se incluyen diversas capturas del *layou*t del DAC elaborado.

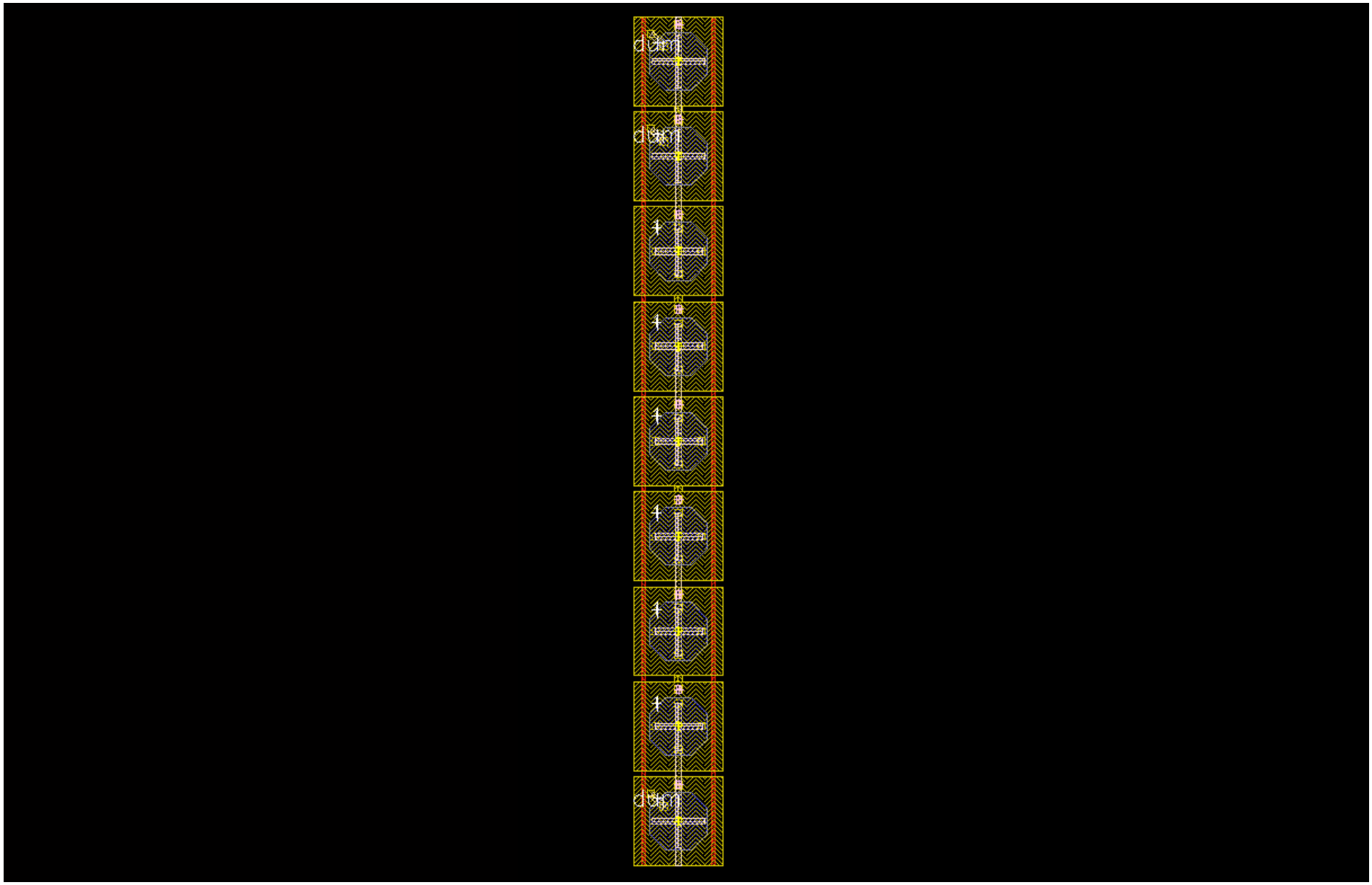
1. *Layout* del capDAC completo
2. Detalle del *layout* de un subDAC
3. Detalle del *layout* de la C_{att}
4. Detalle del *layout* de la capacidad unidad (C)

Layout del capDAC completo

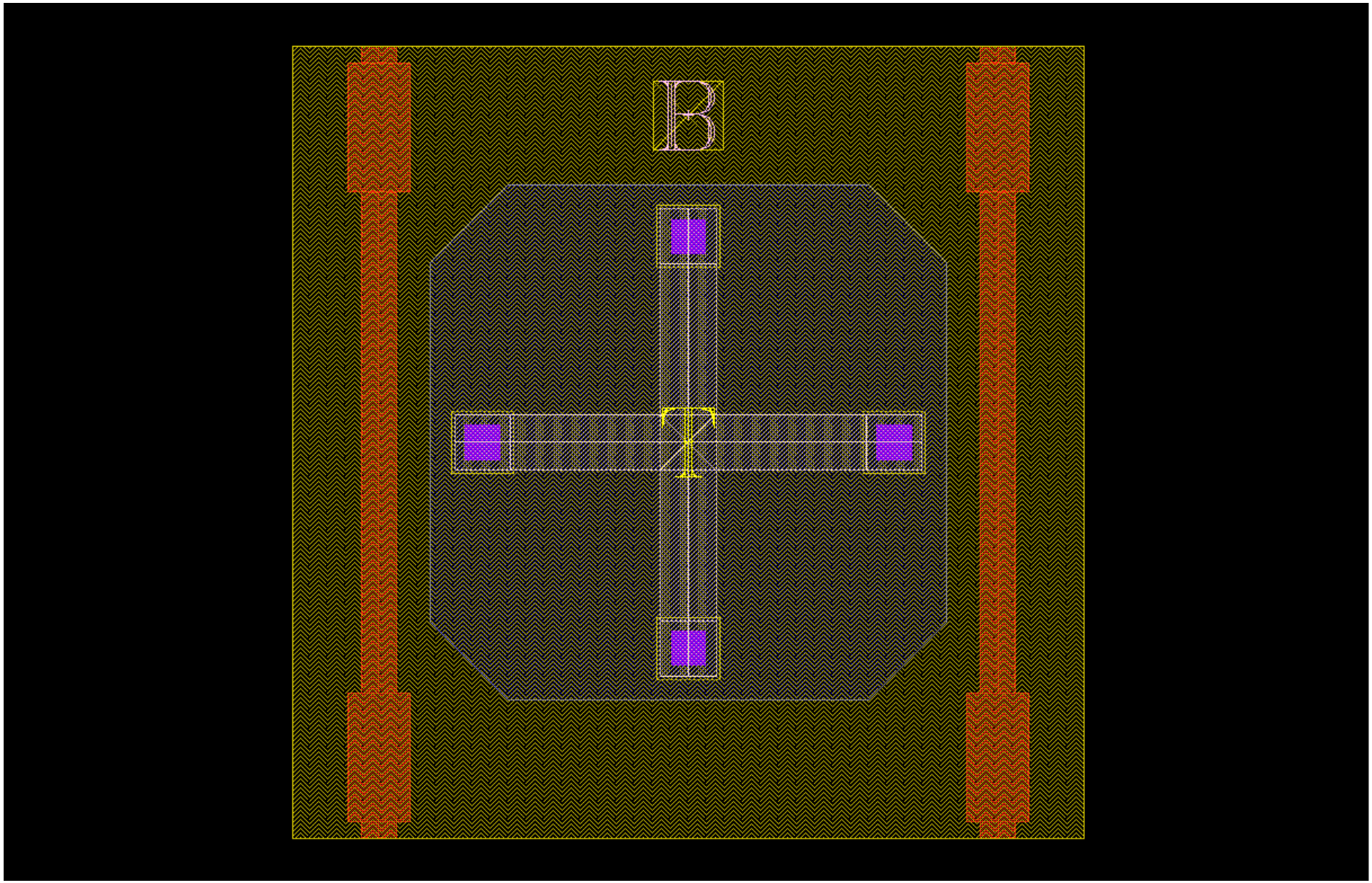


Detalle del *layout* de un subDAC





Detalle del *layout* de la capacidad unidad *C*



ANEXO C. Descripciones Verilog/Verilog-AMS

En el presente anexo se incluyen modelos *Verilog* y *Verilog-AMS* de distintos bloques implementados, que han sido descritos con la finalidad de acelerar las simulaciones realizadas. El listado de modelos que aquí se adjuntan es el siguiente.

5. stimADC12.vams
6. cal_memory.v
7. comparator2.vams
8. track_logic.v
9. switches_driver.v
10. switch.vams
11. switch_comp.vams
12. fd_opi.vams
13. latch.vams
14. compcap_control2.v

```
//Verilog-AMS HDL for "SIM_E_EBU", "stim_ADC12" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module stim_ADC12 ( vss, vinp, vinm, clk, start, por_n);

  //senyales analogicas y digitales
  input vss;
  electrical vss;
  output vinp;
  electrical vinp;
  output vinm;
  electrical vinm;
  output reg clk;
  logic clk;
  output reg start;
  logic start;
  output reg por_n;
  logic por_n;

  //parametros del modelo
  parameter real vcom = 1.25;
  parameter real vrm = 0;
  parameter real vrp = 2.5;
  parameter real tdel = 200p;
  parameter real trise = 200p;
  parameter real tfall = 200p;

  //variables
  integer logfile;
  real vin_dif;
  real display_vin;


  initial begin

$sdf_annotate("/fsvl01/uxvlp/Diploma_thesis/saradc_cal/synthesis/timing/SAR
12cal.sdf", Iadc.Isar);    //asocia el timing de la sintesis al bloque
sintetizado

    clk = 0;
    start = 0;
    por_n = 0;
    vin_dif = -1.25;

    //0. sistema ON (idle)
    @(negedge clk) por_n = 1;

    //1. sistema ON (run)
    @(negedge clk)  start = 1;

    //2. busqueda de los valores de calibracion (busco 8 valores de
calibracion!)
    @(negedge tb_ADC12.Iadc.eoc);
```

```
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc);
    @(negedge tb_ADC12.Iadc.eoc) begin
      $display("\n__CAL: valor de calibracion medio BIT(11):
%d\n",tb_ADC12.Imem.cal_reg11);
      $display("__CAL: valor de calibracion medio BIT(10):
%d\n",tb_ADC12.Imem.cal_reg10);
      $display("__CAL: valor de calibracion medio BIT(9):
%d\n",tb_ADC12.Imem.cal_reg9);
      $display("__CAL: valor de calibracion medio BIT(8):
%d\n",tb_ADC12.Imem.cal_reg8);
      $display("__CAL: valor de calibracion medio BIT(7):
%d\n",tb_ADC12.Imem.cal_reg7);
      $display("__CAL: valor de calibracion medio BIT(6):
%d\n",tb_ADC12.Imem.cal_reg6);
      $display("__CAL: valor de calibracion medio BIT(5):
%d\n",tb_ADC12.Imem.cal_reg5);
      $display("__CAL: valor de calibracion medio BIT(4):
%d\n",tb_ADC12.Imem.cal_reg4);
      $display("__CAL: valor de calibracion medio BIT(3):
%d\n",tb_ADC12.Imem.cal_reg3);
      $display("__CAL: valor de calibracion medio BIT(2):
%d\n",tb_ADC12.Imem.cal_reg2);
      $display("__CAL: valor de calibracion medio BIT(1):
%d\n",tb_ADC12.Imem.cal_reg1);
      $display("__CAL: valor de calibracion medio BIT(0):
%d\n",tb_ADC12.Imem.cal_reg0);

    end


   //3. conversiones
   while(vin_dif<= 1.25) begin

    //Fin de conversion
    @(posedge tb_ADC12.Iadc.eoc) begin
       $display("\nADC_result: %d %e\n", tb_ADC12.Iadc.Isar.qoutout,
display_vin);  //guarda los resultados en el logFILE:
/tmp/e_ebu/sim/tb_ADC12/ams/config/[param_folder]/psf/ncsim.log. Para
extraerlos usar el script:
/fsvl01/uxvlp/Diploma_thesis/saradc_cal/bin/extract_ADC_results*

       $display("\n==================================");
       $display("\n __RESULT SAR ADC: 0x%h (%d) \n",
tb_ADC12.Iadc.Isar.qoutout, tb_ADC12.Iadc.Isar.qoutout);
       $display("==================================\n");

       //nueva tension a convertir
       vin_dif = vin_dif + ((vrp - vrm)/(2**12))/8;  // Input voltage
change by LSB/8 !!!
       //vin_dif = vin_dif + ((vrp - vrm)/(2**12))/16;  // Input voltage
change by LSB/16 !!!

    end
```

```
   end

  end


  always #3000 clk = !clk;  //clk de 166kHz (10.41kSPS)



  analog begin
    V(vinp, vss) <+ transition(vcom + vin_dif ,tdel,trise,tfall);
    V(vinm, vss) <+ transition(vcom - vin_dif ,tdel,trise,tfall);

    display_vin = V(vinp, vinm); // Display
  end


endmodule
```

```
//Verilog HDL for "SIM_E_EBU", "cal_memory" "functional"

`define NBITS_SAR 12  //numero de bits del SAR
`define NBITS_CAL 8   //numero de bits de calibracion
`define NBITS_toCAL 8   //numero de bits a calibrar (8 MSB)
//`define N_MEAS  1  //numero de medidas para hacer la media

module cal_memory ( clk, por_n, start, eoc, data, cal_en, cal_reg_sel,
cal_reg0, cal_reg1, cal_reg2, cal_reg3, cal_reg4, cal_reg5, cal_reg6,
cal_reg7, cal_reg8, cal_reg9, cal_reg10, cal_reg11 );

  //senyales digitales
  input clk;
  input por_n;
  input start;
  input eoc;
  input [`NBITS_CAL-1:0] data;
  output reg cal_en;
  output reg [2:0] cal_reg_sel;
  output reg signed [`NBITS_CAL-1:0] cal_reg0, cal_reg1, cal_reg2,
cal_reg3, cal_reg4, cal_reg5, cal_reg6, cal_reg7, cal_reg8, cal_reg9,
cal_reg10, cal_reg11;

  //parametros
  parameter sWait=0, sSearch=1, sConv=2;

  //variables
  reg [1:0] state;  //almacena el estado actual
  reg [7:0] error_value[7:0];   //almacena las tensiones de error medidas
(precision 8bits)
  reg signed [15:0] cal_value;  //almacena el valor de calibracion
calculado en cada momento (precision 16bits)
  reg [15:0] mask;  //mascara de 16bits para extraer los 8bits LSB y
aplicar el redondeo a 8bits

  //integer i_search; //se utiliza para contar el numero de busquedas
  //real buffer_input;   //acumulador para calcular la media del valor de
calibracion



  //procesado de variables (MAQUINA DE ESTADOS)

  //inicializacion de variables
  initial begin
    state = sWait; //estado inicial 'sWait'

    cal_en = 1;
    cal_reg_sel = 7;

    cal_reg11 = 0;
    cal_reg10 = 0;
    cal_reg9 = 0;
    cal_reg8 = 0;
    cal_reg7 = 0;
```

```
    cal_reg6 = 0;
    cal_reg5 = 0;
    cal_reg4 = 0;
    cal_reg3 = 0;
    cal_reg2 = 0;
    cal_reg1 = 0;
    cal_reg0 = 0;

    error_value[7] = 8'h00;
    error_value[6] = 8'h00;
    error_value[5] = 8'h00;
    error_value[4] = 8'h00;
    error_value[3] = 8'h00;
    error_value[2] = 8'h00;
    error_value[1] = 8'h00;
    error_value[0] = 8'h00;
    cal_value = 16'h0000;   //precision de 16bits para los calculos
intermedios de los valores de calibracion
    mask = 16'h00ff;

  end


  //MAQUINA DE ESTADOS
  always @(posedge clk or negedge por_n) begin

    if (!por_n) begin
          state = sWait;

          //inicializo variables
          //i_search = 0;
          //buffer_input = 0;
          cal_en = 1;
          cal_reg_sel = 7;

    end

    else  case (state)    //paso al siguiente estado

      ///Estado de espera
      sWait :
        begin
          //inicializo variables
          //i_search = 0;
          //buffer_input = 0;
          cal_en = 1;
          cal_reg_sel = 7;

          //pasa al siguiente estado
          if(start) state = sSearch;
          else  state = sWait;

        end


      ///Estado de busqueda
      sSearch :
        begin
```

```
          if ( Iadc.eoc == 0 ) begin   //espera al EOC
            state = sSearch;

            if ( Iadc.Isar.state == 4'b1010 ) begin    //Isar en el estado
de sConv_Cal: actualizo cal_reg_sel y cal_en para tener las senyales
cal_reg_sel y cal_en estables para la siguiente busqueda
              if ( cal_reg_sel > 0 ) begin
                cal_reg_sel = cal_reg_sel - 1;
                cal_en = 1;
              end
              else  cal_en = 0;

            end

          end

          else begin

            //almacena las tensiones de error medidas (precision 8bits)
            if (cal_en == 1) begin
              error_value[cal_reg_sel + 1] = data;
              $display("__Error_value(bit%d)= %d", cal_reg_sel+5,
error_value[cal_reg_sel+1]);

            end

            if ( cal_en == 0 ) begin

              //almacena la ultima tension de error medida (precision
            8bits)
              error_value[0] = data;
              $display("__Error_value(bit         4)= %d", error_value[0]);

              //BIT11
              cal_value = (error_value[7]-128)*256 / 2;

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg11 = cal_value/256
+ 1;
                else cal_reg11 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg11 = cal_value/256 - 1;
                else cal_reg11 = cal_value/256;
              end



              //BIT10
              cal_value = ( (error_value[6]-128)*256 / 2 ) - (
(error_value[7]-128)*256 / 4 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg10 = cal_value/256
+ 1;
                else cal_reg10 = cal_value/256;
              end
              else begin
```

```
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg10 = cal_value/256 - 1;
                else cal_reg10 = cal_value/256;
              end


              //BIT9
              cal_value = ( (error_value[5]-128)*256 / 2 ) - (
(error_value[6]-128)*256 / 4 ) - ( (error_value[7]-128)*256 / 8 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg9 = cal_value/256 +
1;
                else cal_reg9 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg9 = cal_value/256 - 1;
                else cal_reg9 = cal_value/256;
              end


              //BIT8
              cal_value = ( (error_value[4]-128)*256 / 2 ) - (
(error_value[5]-128)*256 / 4 ) - ( (error_value[6]-128)*256 / 8 ) - (
(error_value[7]-128)*256 / 16 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg8 = cal_value/256 +
1;
                else cal_reg8 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg8 = cal_value/256 - 1;
                else cal_reg8 = cal_value/256;
              end


               //BIT7
              cal_value = ( (error_value[3]-128)*256 / 2 ) - (
(error_value[4]-128)*256 / 4 ) - ( (error_value[5]-128)*256 / 8 ) - (
(error_value[6]-128)*256 / 16 ) - ( (error_value[7]-128)*256 / 32 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg7 = cal_value/256 +
1;
                else cal_reg7 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg7 = cal_value/256 - 1;
                else cal_reg7 = cal_value/256;
              end
```

```
                //BIT6
                cal_value = ( (error_value[2]-128)*256 / 2 ) - (
(error_value[3]-128)*256 / 4 )- ( (error_value[4]-128)*256 / 8 ) - (
(error_value[5]-128)*256 / 16 ) - ( (error_value[6]-128)*256 / 32 ) - (
(error_value[7]-128)*256 / 64 );

                if(cal_value >= 0) begin  //round(cal_value)
                  if ( (cal_value & mask) >= 128 ) cal_reg6 = cal_value/256 +
1;
                  else cal_reg6 = cal_value/256;
                end
                else begin
                  if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg6 = cal_value/256 - 1;
                  else cal_reg6 = cal_value/256;
                end


                //BIT5
                cal_value = ( (error_value[1]-128)*256 / 2 ) - (
(error_value[2]-128)*256 / 4 ) - ( (error_value[3]-128)*256 / 8 ) - (
(error_value[4]-128)*256 / 16 ) - ( (error_value[5]-128)*256 / 32 ) - (
(error_value[6]-128)*256 / 64 ) - ( (error_value[7]-128)*256 / 128 );

                if(cal_value >= 0) begin  //round(cal_value)
                  if ( (cal_value & mask) >= 128 ) cal_reg5 = cal_value/256 +
1;
                  else cal_reg5 = cal_value/256;
                end
                else begin
                  if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg5 = cal_value/256 - 1;
                  else cal_reg5 = cal_value/256;
                end


                //BIT4
                cal_value = ( (error_value[0]-128)*256 / 2 ) - (
(error_value[1]-128)*256 / 4 ) - ( (error_value[2]-128)*256 / 8 ) - (
(error_value[3]-128)*256 / 16 ) - ( (error_value[4]-128)*256 / 32 ) - (
(error_value[5]-128)*256 / 64 ) - ( (error_value[6]-128)*256 / 128 ) - (
(error_value[7]-128)*256 / 256 );

                if(cal_value >= 0) begin  //round(cal_value)
                  if ( (cal_value & mask) >= 128 ) cal_reg4 = cal_value/256 +
1;
                  else cal_reg4 = cal_value/256;
                end
                else begin
                  if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg4 = cal_value/256 - 1;
                  else cal_reg4 = cal_value/256;
                end
```

```
              //BIT3
              cal_value = - ( (error_value[0]-128)*256 / 4 ) - (
(error_value[1]-128)*256 / 8 ) - ( (error_value[2]-128)*256 / 16 ) - (
(error_value[3]-128)*256 / 32 ) - ( (error_value[4]-128)*256 / 64 ) - (
(error_value[5]-128)*256 / 128 ) - ( (error_value[6]-128)*256 / 256 ) - (
(error_value[7]-128)*256 / 512 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg3 = cal_value/256 +
1;
                else cal_reg3 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg3 = cal_value/256 - 1;
                else cal_reg3 = cal_value/256;
              end


              //BIT2
              cal_value = - ( (error_value[0]-128)*256 / 8 ) - (
(error_value[1]-128)*256 / 16 ) - ( (error_value[2]-128)*256 / 32 ) - (
(error_value[3]-128)*256 / 64 ) - ( (error_value[4]-128)*256 / 128 ) - (
(error_value[5]-128)*256 / 256 ) - ( (error_value[6]-128)*256 / 512 ) - (
(error_value[7]-128)*256 / 1024 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg2 = cal_value/256 +
1;
                else cal_reg2 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg2 = cal_value/256 - 1;
                else cal_reg2 = cal_value/256;
              end


              //BIT1
              cal_value = - ( (error_value[0]-128)*256 / 16 ) - (
(error_value[1]-128)*256 / 32 ) - ( (error_value[2]-128)*256 / 64 ) - (
(error_value[3]-128)*256 / 128 ) - ( (error_value[4]-128)*256 / 256 ) - (
(error_value[5]-128)*256 / 512 ) - ( (error_value[6]-128)*256 / 1024 ) - (
(error_value[7]-128)*256 / 2048 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg1 = cal_value/256 +
1;
                else cal_reg1 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg1 = cal_value/256 - 1;
                else cal_reg1 = cal_value/256;
              end
```

```
              //BIT0
              cal_value = - ( (error_value[0]-128)*256 / 32 ) - (
(error_value[1]-128)*256 / 64 ) - ( (error_value[2]-128)*256 / 128 ) - (
(error_value[3]-128)*256 / 256 ) - ( (error_value[4]-128)*256 / 512 ) - (
(error_value[5]-128)*256 / 1024 ) - ( (error_value[6]-128)*256 / 2048 ) - (
(error_value[7]-128)*256 / 4096 );

              if(cal_value >= 0) begin  //round(cal_value)
                if ( (cal_value & mask) >= 128 ) cal_reg0 = cal_value/256 +
1;
                else cal_reg0 = cal_value/256;
              end
              else begin
                if( ((cal_value & mask) != 0) & ((cal_value & mask) <= 128)
) cal_reg0 = cal_value/256 - 1;
                else cal_reg0 = cal_value/256;
              end



              $display("__Cal_value(bit11)= %d", cal_reg11);
              $display("__Cal_value(bit10)= %d", cal_reg10);
              $display("__Cal_value(bit9)= %d", cal_reg9);
              $display("__Cal_value(bit8)= %d", cal_reg8);
              $display("__Cal_value(bit7)= %d", cal_reg7);
              $display("__Cal_value(bit6)= %d", cal_reg6);
              $display("__Cal_value(bit5)= %d", cal_reg5);
              $display("__Cal_value(bit4)= %d", cal_reg4);
              $display("__Cal_value(bit3)= %d", cal_reg3);
              $display("__Cal_value(bit2)= %d", cal_reg2);
              $display("__Cal_value(bit1)= %d", cal_reg1);
              $display("__Cal_value(bit0)= %d", cal_reg0);


              //pasa al siguiente estado
              state <= sConv;
            end

          end

        end


      ///Estado de conversion
      sConv:
        begin
        cal_en = 0;

        if( Iadc.eoc == 1 ) begin   //fin de conversion
          $display("ADC RESULT: %h (%d)",Iadc.data, Iadc.data);
        end

      end

    endcase
  end

endmodule
```

```
//Verilog-AMS HDL for "COMPARATOR_e_ebu", "comparator2" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module comparator2 ( cn_n, cp_n, refm, refp, vdda, vssa, vsub, clk,
ibias2u, inn, inp, latch_enable, pd_n, prch, smpl );

  //senyales analogicas y digitales
  input pd_n;
  logic pd_n;
  input inn;
  electrical inn;
  input  [2:0] ibias2u;
  electrical [2:0] ibias2u;
  inout vdda;
  electrical vdda;
  output reg cn_n;
  logic cn_n;
  input inp;
  electrical inp;
  input clk;
  logic clk;
  inout vsub;
  electrical vsub;
  inout vssa;
  electrical vssa;
  input prch;
  logic prch;
  output reg cp_n;
  logic cp_n;
  input smpl;
  logic smpl;
  input latch_enable;
  logic latch_enable;
  inout refp;
  electrical refp;
  inout refm;
  electrical refm;

  //parametros del modelo
  parameter real resol = 76.3u;  //resolucion de 15bits (2.5/2**15)
  parameter real res_switch = 10;  //resistencia switch-on



analog begin
   //conecto las ftes de corriente a una resistencia (cualquiera) porque si
no, tenemos fuentes de corriente sin conectar y el simulador
   // funciona mal: una fte de corriene fuerza una corriente por ella, pero
si esta sin conectar.. no pasa corriente por ella!!! (inconcluencia para el
simulador)
   V(ibias2u[2], vssa) <+ I(ibias2u[2], vssa)/2e-6;
   V(ibias2u[1], vssa) <+ I(ibias2u[1], vssa)/2e-6;
   V(ibias2u[0], vssa) <+ I(ibias2u[0], vssa)/2e-6;
```

```
   //switches de pregarga y muestreo
   if (prch == 1 && smpl == 0) begin   //If control is 1-0, close the
switch (precharge)
      V(refp, inp) <+ res_switch * I(refp, inp);
      V(refm, inn) <+ res_switch * I(refm, inn);
    end
    else if (prch == 0 && smpl == 1) begin   //If control is 0-1, close the
switch (sample)
      V(inp, inn) <+ res_switch * I(inp, inn);
    end
    else if (prch == 0 && smpl == 0) begin //If control is 0-0, open
switches (precharge & sample)
      I(refp, inp) <+ 0;
      I(refm, inn) <+ 0;
      I(inp, inn) <+ 0;
    end
    else begin   //Default: open switches (precharge & sample)
      I(refp, inp) <+ 0;
      I(refm, inn) <+ 0;
      I(inp, inn) <+ 0;
    end

 end



 initial begin
    cn_n = (V(inp, inn) > resol) ? 1 : 0;
    cp_n = !cn_n;
 end


 always @(negedge clk) begin
   cn_n = (V(inp, inn) > resol) ? 1 : 0;
   cp_n = !cn_n;
 end

endmodule
```

```
//Verilog HDL for "SIM_E_EBU", "track_logic" "functional"

module track_logic ( track_i, track_n, track );

  //senyales digitales
  input track;
  output reg track_i;
  output track_n;


  initial begin
    track_i = 0;
  end

  always @(posedge track or negedge track) begin
    #5 track_i = track
  end

  assign track_n = !track_i;

endmodule
```

```
//Verilog HDL for "SIM_E_EBU", "switches_driver" "functional"

module switches_driver ( NCH, PCH, PCHb, ENB, SC, NCHb );

  //senyales digitales
  input SC;
  output PCH;
  output NCHb;
  output PCHb;
  input ENB;
  output NCH;


  assign PCH = !ENB && SC;
  assign PCHb = !(!ENB && SC);
  assign NCH = !ENB && !SC;
  assign NCHb = !(!ENB && !SC);

endmodule
```

```
//Verilog-AMS HDL for "SIM_E_EBU", "switch" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module switch ( so, en, enb, nch, pch, pchb, refm, refp, vin, vdd, vss,
nchb);

  //senyales analogicas y digitales
  input vdd;
  electrical vdd;
  input vss;
  electrical vss;
  input nch;
  logic nch;
  input nchb;
  logic nchb;
  input pch;
  logic pch;
  input pchb;
  logic pchb;
  input en;
  logic en;
  input enb;
  logic enb;
  input vin;
  electrical vin;
  input refp;
  electrical refp;
  input refm;
  electrical refm;
  output so;
  electrical so;

  //parametros del modelo
  parameter real res_switch = 10 ;

  analog begin
    if (pch == 1 && nch == 0 && en == 0) begin //If control is 1-0-0, close
the switch (refp)
      V(refp, so) <+ res_switch * I(refp, so);
    end
    else if (pch == 0 && nch == 1 && en == 0) begin //If control is 0-1-0,
close the switch (refm)
      V(refm, so) <+ res_switch * I(refm, so);
    end
    else if (pch == 0 && nch == 0 && en == 1) begin //If control is 0-0-1,
close the switch (track)
      V(vin, so) <+ res_switch * I(vin, so);
    end
    else begin  //Default: close the switch (refm)
      V(refm, so) <+ res_switch * I(refm, so);
    end
  end

endmodule
```

```
//Verilog-AMS HDL for "COMPARATOR_e_ebu", "switch_comp" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module switch_comp ( bit, vss, vdd, bit_n, in, out );

  inout vdd;
  electrical vdd;
  inout vss;
  electrical vss;
  input bit_n;
  logic bit_n;
  input bit;
  logic bit;
  inout in;
  electrical in;
  inout out;
  electrical out;

  //parametros del modelo
  parameter real res_switch_on = 10;
  parameter real res_switch_off = 1T;


  analog begin

  if(bit == 1) begin
    V(in,out) <+ res_switch_on * I(in,out);
  end
  else begin
    V(in,out) <+ res_switch_off * I(in,out);
  end

  end

endmodule
```

```
//Verilog-AMS HDL for "COMPARATOR_e_ebu", "fd_opi" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module fd_opi ( outn, outp, vdda, vssa, vsub, bi_2u, inn, inp, pd_rst_n,
pd_rst );

  //senyales analogicas y digitales
  input pd_rst;
  logic pd_rst;
  output outp;
  electrical outp;
  input inn;
  electrical inn;
  inout vdda;
  electrical vdda;
  input inp;
  electrical inp;
  inout vsub;
  electrical vsub;
  inout vssa;
  electrical vssa;
  input pd_rst_n;
  logic pd_rst_n;
  input bi_2u;
  electrical bi_2u;
  output outn;
  electrical outn;


  //parametros del modelo
  parameter real vcom = 1.35;  //modo comun de la salida
  parameter real Ad = 39.81;  //ganancia diferencial (en lineal)
  parameter real Cin = 100f;  //capacidad de entrada
  parameter real Vsat = 2;  //tension de saturacion a la salida

  analog begin
   //conecto las ftes de corriente a una resistencia (cualquiera) porque si
no, tenemos fuentes de corriente sin conectar y el simulador
   // funciona mal: una fte de corriene fuerza una corriente por ella, pero
si esta sin conectar.. no pasa corriente por ella!!! (inconcluencia para el
simulador)
   V(bi_2u, vssa) <+ I(bi_2u, vssa)/2e-6;

   //Capacidad de entrada
   I(inp, vssa) <+ Cin*ddt(V(inp, vssa));
   I(inn, vssa) <+ Cin*ddt(V(inn, vssa));



   //AMPLIFICADOR

    if (Ad*V(inp,inn) > Vsat) begin
        V(outp, vssa) <+ vcom + Vsat/2;
```

```
        V(outn, vssa) <+ vcom - Vsat/2;
    end
    else if(Ad*V(inp,inn) < -Vsat) begin
        V(outp, vssa) <+ vcom - Vsat/2;
        V(outn, vssa) <+ vcom + Vsat/2;
    end

    else begin
        V(outp, vssa) <+ vcom + Ad*V(inp,inn)/2;
        V(outn, vssa) <+ vcom - Ad*V(inp,inn)/2;
    end

 end

endmodule
```

latch.vams

```
//Verilog-AMS HDL for "COMPARATOR_e_ebu", "latch" "verilogams"

`include "constants.vams"
`include "disciplines.vams"

module latch ( cdone_n, cn_n, cp_n, ib_2u, inn, inp, latch_enable,
latch_reset, pd, pd_n, sclk, vdda1, vdda2, vssa1, vssa2, vsub );

  //senyales analogicas y digitales
  input pd_n;
  logic pd_n;
  input inn;
  electrical inn;
  output reg cn_n;
  logic cn_n;
  input pd;
  logic pd;
  input inp;
  electrical inp;
  input ib_2u;
  electrical ib_2u;
  output cdone_n;
  logic cdone_n;
  input vssa2;
  electrical vssa2;
  input vssa1;
  electrical vssa1;
  output reg cp_n;
  logic cp_n;
  input vdda2;
  electrical vdda2;
  input vdda1;
  electrical vdda1;
  input vsub;
  electrical vsub;
  input sclk;
  logic sclk;
  input latch_reset;
  logic latch_reset;
  input latch_enable;
  logic latch_enable;

  //parametros del modelo
  parameter real resol = 30m; //resolucion de 30mV


 //conecto las ftes de corriente a una resistencia (cualquiera) porque si
no, tenemos fuentes de corriente sin conectar y el simulador
 // funciona mal: una fte de corriene fuerza una corriente por ella, pero
si esta sin conectar.. no pasa corriente por ella!!! (inconcluencia para el
simulador)
 analog begin
   V(ib_2u, vssa1) <+ I(ib_2u, vssa1)/4e-6;
 end
```

```
 initial begin
   cn_n = 1;
   cp_n = 1;
 end

 always @(posedge sclk or posedge latch_reset) begin

   if(latch_reset == 1) begin
    #1 cn_n = 1;
    #1 cp_n = 1;
   end

   else begin
    #4 cn_n = (V(inp, inn) > resol) ? 1 : 0;
    #4 cp_n = !cn_n;
   end

 end

endmodule
```

```
//Verilog HDL for "COMPARATOR_e_ebu", "compcap_control2" "functional"

module compcap_control2 ( s1, s1_n, smpl, vdda, vsub, vssa, s0, s0_n,
pd_rst, clk, pd_rst_n, clk_n, prch, pd_n, sprch_n, sprch, pd );

  //senyales digitales
  output s1_n;
  output pd_rst;
  input pd_n;
  inout vdda;
  output s1;
  input clk;
  inout vsub;
  inout vssa;
  output sprch;
  output s0;
  output pd;
  input prch;
  output pd_rst_n;
  output sprch_n;
  input smpl;
  output clk_n;
  output s0_n;


  //ciurcuito combinacional

  assign clk_n = !clk;

  assign pd = !pd_n;

  assign #1 sprch = prch;
  assign #1 sprch_n = !prch;

  assign #2 s0 = smpl;
  assign #2 s0_n = !smpl;

  assign #2 s1 = s0;
  assign #2 s1_n = s0_n;

  assign #7 delay = !(s1_n & clk);
  assign #1 Pulsos = (s1_n & clk) & delay;

  assign #1 pd_rst = !pd_n | Pulsos;
  assign #1 pd_rst_n = !(!pd_n | Pulsos);


endmodule
```

ANEXO D. Modelo teórico MATLAB del sistema de calibración

En el presente anexo se incluyen los distintos *scripts* de MATLAB que se han utilizado para modelar el comportamiento del ADC y ajustar la topología del DAC y el sistema de calibración.

1. ADC12_cal.m
2. DAC.m
3. calSAR.m

```
function [INL, INL_calib, DNL, DNL_calib] = ADC12_cal(err_vector_convDAC,
err_vector_calDAC)

%-------------------------------------------------------------------------
% - function ADC12_cal(): modelo teorico del ADC12. Calcula su fdt, DNL e
%                         INL.
%   Los resultados coinciden con los de la simulaciones Cadence del ADC12
%   (la linearidad del DAC12 es la que da la linearidad del ADC12)
%
%   UTILIZA EL METODO DE CALIBRACION: 8MSB calibrados + 4LSB
%   pseudocalibrados
%
%   .INL, INL_calib: INL del DAC sin calibrar y con calibracion
%   .DNL, DNL_calib: DNL del DAC sin calibrar y con calibracion
%
%   .err_vector_convDAC: vector con los errores del convDAC (errores + y -
%                        en %)
%
%      err_vector_convDAC = [errC_conv(1) . . . errC_conv(12)
%                            errCc_sub1_conv]
%
%   .err_vector_calDAC: vector con los errorees del calDAC (errores + y -
%                       en %)
%
%       err_vector_calDAC = [errC_cal(1) . . . errC_conv(8)]
%-------------------------------------------------------------------------

clf;

% Variables ---------------------------------------------------------------
Nbits_conv = 12;    % num de bits del convDAC
Nbits_cal = 8;  % num de bits del calDAC

%para simplificar escojemos: Vcm=0 y Vref=1


% Calculo de los valores de calibracion -----------------------------------
%(en VOLTIOS son los ideales, en CODE son los reales[redondeo calDAC])

% 1. Tension error (Verror, Verror_code)
%codigo del convDAC
code_matrix_conv = [0 1 1 1 1 1 1 1 ; 0 0 1 1 1 1 1 1 ; 0 0 0 1 1 1 1 1 ; 0
0 0 0 1 1 1 1 ; 0 0 0 0 0 1 1 1 ; 0 0 0 0 0 0 1 1 ; 0 0 0 0 0 0 0 1 ; 0 0 0
0 0 0 0 0];
code_matrix_conv = [code_matrix_conv  ones(8,4)];

%conversion
[Vconv,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
code_matrix_conv, [1 zeros(1,7)]);
[Vconv1,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
[zeros(1,11) 1], [1 zeros(1,7)]); % el Ct tambien a 1!
VdacA = Vconv' + Vconv1';

%codigo del convDAC (bit-reverse)
code_matrix_conv = [eye(8) zeros(8,4)];
```

```
%conversion
[Vconv,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
code_matrix_conv, [1 zeros(1,7)]);
VdacB = Vconv';

%error de voltaje (VOLTIOS) %resol inf
Verror =  - (VdacB - VdacA);

%error de voltaje (CODE)    %resol 8 bits
Verror_code = calSAR(Verror, err_vector_convDAC, err_vector_calDAC)


% 2. Tension de calibracion (Vcal,Vcal_code)
%tension de calibracion (CODE)  %resol 8 bits
Verror_code_dec = Verror_code * [128 64 32 16 8 4 2 1]';
Verror_code_dec = Verror_code_dec - 128;

%%%%% calibracion del main %%%%%
Vcal_code_dec(1) = Verror_code_dec(1)/2;
Vcal_code_dec(2) = Verror_code_dec(2)/2 - Verror_code_dec(1)/4;
Vcal_code_dec(3)   =   Verror_code_dec(3)/2   -   Verror_code_dec(2)/4   -
Verror_code_dec(1)/8;
Vcal_code_dec(4)   =   Verror_code_dec(4)/2   -   Verror_code_dec(3)/4   -
Verror_code_dec(2)/8 - Verror_code_dec(1)/16;

%%%%% calibracion del sub1 %%%%%
Vcal_code_dec(5)   =   Verror_code_dec(5)/2   -   Verror_code_dec(4)/4   -
Verror_code_dec(3)/8 - Verror_code_dec(2)/16 - Verror_code_dec(1)/32;
Vcal_code_dec(6)   =   Verror_code_dec(6)/2   -   Verror_code_dec(5)/4   -
Verror_code_dec(4)/8  -  Verror_code_dec(3)/16  -  Verror_code_dec(2)/32  -
Verror_code_dec(1)/64;
Vcal_code_dec(7)   =   Verror_code_dec(7)/2   -   Verror_code_dec(6)/4   -
Verror_code_dec(5)/8  -  Verror_code_dec(4)/16  -  Verror_code_dec(3)/32  -
Verror_code_dec(2)/64 - Verror_code_dec(1)/128;
Vcal_code_dec(8)   =   Verror_code_dec(8)/2   -   Verror_code_dec(7)/4   -
Verror_code_dec(6)/8  -  Verror_code_dec(5)/16  -  Verror_code_dec(4)/32  -
Verror_code_dec(3)/64 - Verror_code_dec(2)/128 - Verror_code_dec(1)/256;

%%%%% pseudocalibracion del sub2 %%%%%
Vcal_code_dec(9)   =   -   Verror_code_dec(8)/4   -   Verror_code_dec(7)/8   -
Verror_code_dec(6)/16  -  Verror_code_dec(5)/32  -  Verror_code_dec(4)/64  -
Verror_code_dec(3)/128 - Verror_code_dec(2)/256 - Verror_code_dec(1)/512;
Vcal_code_dec(10)  =  -  Verror_code_dec(8)/8  -  Verror_code_dec(7)/16  -
Verror_code_dec(6)/32  -  Verror_code_dec(5)/64  -  Verror_code_dec(4)/128  -
Verror_code_dec(3)/256 - Verror_code_dec(2)/512 - Verror_code_dec(1)/1024;
Vcal_code_dec(11)  =  -  Verror_code_dec(8)/16  -  Verror_code_dec(7)/32  -
Verror_code_dec(6)/64  -  Verror_code_dec(5)/128  -  Verror_code_dec(4)/256  -
Verror_code_dec(3)/512 - Verror_code_dec(2)/1024 - Verror_code_dec(1)/2048;
Vcal_code_dec(12)  =  -  Verror_code_dec(8)/32  -  Verror_code_dec(7)/64  -
Verror_code_dec(6)/128  -  Verror_code_dec(5)/256  -  Verror_code_dec(4)/512  -
Verror_code_dec(3)/1024          -          Verror_code_dec(2)/2048          -
Verror_code_dec(1)/4096;

Vcal_code_dec = round(Vcal_code_dec)   %redondeo a 8bits
```

```
% Aplicacion de la calibracion ------------------------------------------

%codigos del convDAC
%(codigos de 1/8 de LSB)
codein_conv = 0:(1/8):(2^Nbits_conv) - 1; % en decimal (exactos)
code_matrix_conv  =  dec2bin(floor(codein_conv),12)-48;      %  en  binario
(redondeados a 12bits)

%codigos del calDAC (CODE)
code_matrix_cal_dec = code_matrix_conv(:,1:end) * (Vcal_code_dec(:,1:end))'
+128;
code_matrix_cal = dec2bin(code_matrix_cal_dec,8)-48;

%conversion (CODE)
[Vconv,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
code_matrix_conv,code_matrix_cal);
Vdac_calib = Vout';



% CALCULOS DE LINEARIDAD y GRAFICAS --------------------------------------

%calculo de las Vlsb de referencia
Vlsb_ideal = 1/2^Nbits_conv;    %Vlsb del DAC ideal
[Vconv,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
[zeros(1,11) 1], [1 zeros(1,7)]);       %Vlsb del DAC real (tiene error de
ganancia) utilizamos esta, no la Vlsb ideal, porque lo que queremos ver son
solo los errores de linearidad, no el error de ganancia

Vlsb_conv = Vconv;

% FDT del DACideal y del DACreal(sin calibrar) - - -
%salida del DAC ideal de 12bits
Videal = floor(codein_conv)*Vlsb_conv;     %12 bits

%tension de entrada (rampa)
Vin = codein_conv*Vlsb_conv;

%salida del DAC real (sin calibrar)
[Vconv,   Vcal,   Vout]   =   DAC(err_vector_convDAC,   err_vector_calDAC,
code_matrix_conv, [1 zeros(1,7)]);
Vdac = Vconv';

%representacion grafica
subplot(3,2,5);
hold on;
plot(codein_conv, Vin/Vlsb_conv, 'b');
plot(codein_conv, Videal/Vlsb_conv, 'g');
plot(codein_conv, Vdac/Vlsb_conv, 'r');
hold off;
title('Vdac-ideal vs. Vdac-real(sin calibrar)');
xlabel('Input (code)'); ylabel('Output (LSB)');


% FDT del DACideal y del DACreal(con calibracion) - - -
%representacion grafica
subplot(3,2,6);
hold on;
plot(codein_conv, Vin/Vlsb_conv, 'b');
```

```
plot(codein_conv, Videal/Vlsb_conv, 'g');
plot(codein_conv, Vdac_calib/Vlsb_conv, 'r');
hold off;
title('Vdac-ideal vs. Vdac-real(con calibracion)');
xlabel('Input (code)'); ylabel('Ouput (LSB)');


% INL / DNL sin calibracion - - -
%calculo del DNL sin calibrar
DNL = Vdac(1:8:end);
DNL = ( diff(DNL) - Vlsb_conv ) / Vlsb_conv;

%valores maximo y minimo
DNL_limits = [min(DNL) max(DNL)]

%representacion grafica
subplot(3,2,3);
plot(DNL, 'b');
title('DNL (sin calibracion)');
xlabel('Input (code)'); ylabel('Ouput (LSB)');

%calculo del INL sin calibrar
INL = (Vdac - Videal) / Vlsb_conv;

%valores maximo y minimo
INL_limits = [min(INL) max(INL)]

%representacion grafica
subplot(3,2,1);
plot(codein_conv, INL, 'b');
title('INL (sin calibracion)');
xlabel('Input (code)'); ylabel('Ouput (LSB)');


% INL / DNL con calibracion - - -
%calculo del DNL con calibracion
DNL_calib = Vdac_calib(1:8:end);
DNL_calib = ( diff(DNL_calib) - Vlsb_conv ) / Vlsb_conv;

%valores maximo y minimo
DNLcalib_limits = [min(DNL_calib) max(DNL_calib)]

%representacion grafica
subplot(3,2,4);
plot(DNL_calib, 'b');
title('DNL (con calibracion)');
xlabel('Input (code)'); ylabel('Output (LSB)');

%calculo del INL con calibracion
INL_calib = (Vdac_calib - Videal) / Vlsb_conv;

%valores maximo y minimo
INLcalib_limits = [min(INL_calib) max(INL_calib)]

%representacion grafica
subplot(3,2,2);
plot(codein_conv, INL_calib, 'b');
title('INL (con calibracion)');
xlabel('Input (code)'); ylabel('Ouput (LSB)');

end
```

```
function [Vconv, Vcal, Vout] = DAC(err_vector_convDAC, err_vector_calDAC,
code_matrix_convDAC, code_matrix_calDAC)

%-------------------------------------------------------------------------
% - funcion DAC(): calcula la tension de salida del DAC compuesto por
% el convDAC y el calDAC. La tension de salida sera la contribucion de
% ambos (convDAC y calDAC)
%
%   .Vconv: tension de salida del convDAC solo
%   .Vcal: tension de salida del calDAC solo
%   .Vout: tension de salida del DAC (Vconv + vcal)
%
%   .code_matrix_convDAC: codigo para el convDAC (12bits). [ *(1) -> BIT
%                          MSB ]
%   .code_matrix_calDAC: codigo pra el calDAC (8 bits). [ *(1) -> BIT MSB ]
%   .err_vector_convDAC: vector con los errores del convDAC (errores + y -
%                        en %)
%
%      err_vector_convDAC = [errC_conv(1) . . . errC_conv(12)
%                            errCc_sub1_conv]
%
%   .err_vector_calDAC: vector con los errorees del calDAC (errores + y -
%                       en %)
%
%       err_vector_calDAC = [errC_cal(1) . . . errC_conv(8)]
%
%-------------------------------------------------------------------------

%% convDAC ----------------------------------------------------------------
%definicion del convDAC -------------------------
 C_conv = [8 ; 4 ; 2 ; 1 ; 8 ; 4 ; 2 ; 1 ; 8 ; 4 ; 2 ; 1];     % --- %
//C_conv(1) -> MSB

Ct_conv = 1;              % --- %
Cc_sub2_conv = (16/15) + 0.016;    % --- % + parasitic (0.016)
Cc_sub1_conv = (16/15) + 0.014;    % --- % + parasitic (0.014)

Cps2_conv = 0.3613;        % --- % parasitic en subDAC (0.3613)
Cps1_conv = 0.2279;        % --- % parasitic en midDAC (0.2279)
Cpm_conv = 0;          % --- % parasitic en mainDAC (no influye: se compensa
en la precarga)

Cpcal_conv = 0;       %carga del convDAC, en este caso es la capacidad el
                      %calDAC
                      %        - actua como un parasitic en el mainDAC!
                      %        - genera un error de ganancia, pero este se
                      %          compensa en el muestreo

err_vector_convDAC = err_vector_convDAC(:); %aplicacion de los errores
C_conv = C_conv .* ( err_vector_convDAC(1:end-1)/100 + 1 );
Cc_sub1_conv = Cc_sub1_conv .* ( err_vector_convDAC(end)/100 + 1 );

C_conv_sub2 = C_conv(9:12);
C_conv_sub1 = C_conv(5:8);
C_conv_main = C_conv(1:4);
```

```
CB_conv = ((sum(C_conv_main) + Cpm_conv + Cpcal_conv) * Cc_sub1_conv) /
(sum(C_conv_main) + Cpm_conv + Cpcal_conv + Cc_sub1_conv);
CC_conv = ((sum(C_conv_sub2) + Ct_conv + Cps2_conv) * Cc_sub2_conv) /
(sum(C_conv_sub2) + Ct_conv + Cps2_conv + Cc_sub2_conv);
CA_conv = ((CB_conv + sum(C_conv_sub1) + Cps1_conv) * Cc_sub2_conv) /
(CB_conv + sum(C_conv_sub1) + Cps1_conv + Cc_sub2_conv);
CD_conv = ((CC_conv + sum(C_conv_sub1) + Cps1_conv) * Cc_sub1_conv) /
(CC_conv + sum(C_conv_sub1) + Cps1_conv + Cc_sub1_conv);

C_conv_tot_sub2 = CA_conv + Cps2_conv + sum(C_conv_sub2) + Ct_conv;
C_conv_tot_sub1 = CB_conv + sum(C_conv_sub1) + Cps1_conv + CC_conv;
C_conv_tot_main = sum(C_conv_main) + Cpcal_conv + Cpm_conv + CD_conv;

Fsub1_conv = Cc_sub1_conv / (Cc_sub1_conv + Cpcal_conv + Cpm_conv +
sum(C_conv_main));
Fsub2_conv = Fsub1_conv * ( Cc_sub2_conv / (Cc_sub2_conv + sum(C_conv_sub1)
+ Cps1_conv + CB_conv) );

%proceso de conversion ----------------------------
code_matrix_conv_sub2 = code_matrix_convDAC(:,9:12);
code_matrix_conv_sub1 = code_matrix_convDAC(:,5:8);
code_matrix_conv_main = code_matrix_convDAC(:,1:4);

Vdac_conv_sub2 = (1/C_conv_tot_sub2) * code_matrix_conv_sub2 * C_conv_sub2;
Vdac_conv_sub1 = (1/C_conv_tot_sub1) * code_matrix_conv_sub1 * C_conv_sub1;
Vdac_conv_main = (1/C_conv_tot_main) * code_matrix_conv_main * C_conv_main;

Vconv = Vdac_conv_sub2 * Fsub2_conv + Vdac_conv_sub1 * Fsub1_conv +
Vdac_conv_main;


%%  calDAC  ------------------------------------------------------------Z
%definicion del calDAC -------------------------
C_cal = [8 ; 4 ; 2 ; 1 ; 8 ; 4 ; 2 ; 1];    % --- %  //C_cal(1) -> MSB

Ct_cal = 1;           % --- %
Cc_cal = (16/15) + 0.014; % --- % + parasitic (0.014)
Catt_cal = 1/6;       % --- %

Cps2_cal = 0.2294;  % --- % parasitc en subDAC_cal (0.2294)
Cpm_cal = 0;        % --- %
Cpconv_cal = CD_conv + Cpm_conv + sum(C_conv_main);      %carga del calDAC,
en este caso es la capacidad del convDAC

err_vector_calDAC = err_vector_calDAC(:); %aplicacion de los errores
C_cal = C_cal .* (err_vector_calDAC(1:end)/100 + 1);

C_cal_sub2 = C_cal(5:8);
C_cal_sub1 = C_cal(1:4);

CB_cal = ((Cpm_cal + Cpconv_cal) * Catt_cal) / (Cpm_cal + Cpconv_cal +
Catt_cal);
CC_cal = ((Cps2_cal + sum(C_cal_sub2) + Ct_cal) * Cc_cal) / (Cps2_cal +
sum(C_cal_sub2) + Ct_cal + Cc_cal);
CA_cal = ((CB_cal + sum(C_cal_sub1)) * Cc_cal) / (CB_cal + sum(C_cal_sub1)
+ Cc_cal);
CD_cal = ((CC_cal + sum(C_cal_sub1)) * Catt_cal) / (CC_cal +
sum(C_cal_sub1) + Catt_cal);
```

```
C_cal_tot_sub2 = CA_cal + Cps2_cal + sum(C_cal_sub2) + Ct_cal;
C_cal_tot_sub1 = CB_cal + sum(C_cal_sub1) + CC_cal;

Fsub1_cal = Catt_cal / (Catt_cal + Cpm_cal + Cpconv_cal);
Fsub2_cal = Fsub1_cal * ( Cc_cal / (Cc_cal + sum(C_cal_sub1) + CB_cal) );

%calulo del offset --------------------------------
%(para: code_matrix_in = 10...0 -> Vout = 0V)
code_matrix_cal_sub2 = zeros(1,4);
code_matrix_cal_sub1 = [1 zeros(1,3)];

Vdac_cal_sub2 = (1/C_cal_tot_sub2) * code_matrix_cal_sub2 * C_cal_sub2;
Vdac_cal_sub1 = (1/C_cal_tot_sub1) * code_matrix_cal_sub1 * C_cal_sub1;

offset = Vdac_cal_sub2 * Fsub2_cal + Vdac_cal_sub1 * Fsub1_cal;

%proceso de conversion ----------------------------
code_matrix_cal_sub2 = code_matrix_calDAC(:,5:8);
code_matrix_cal_sub1 = code_matrix_calDAC(:,1:4);

Vdac_cal_sub2 = (1/C_cal_tot_sub2) * code_matrix_cal_sub2 * C_cal_sub2;
Vdac_cal_sub1 = (1/C_cal_tot_sub1) * code_matrix_cal_sub1 * C_cal_sub1;

Vcal = Vdac_cal_sub2 * Fsub2_cal + Vdac_cal_sub1 * Fsub1_cal - offset;


%% tension total ------------------------------------------------------
Vout = Vconv + Vcal;


end
```

```
function code_out = calSAR(Vin, err_vector_convDAC, err_vector_calDAC)

%-------------------------------------------------------------------------
% - funcion calSAR(): implementa una busqueda SAR en el calDAC. A partir de
% la tension Vin, obtiene el codigo del calDAC que la genera (con la
% resolucion de ~LSB/6 que tiene el calDAC)
%
%   .code_out: codigo encontrado por el calDAC
%
%   .Vin: tension a buscar por el calDAC
%   .err_vector_convDAC: vector con los errores del convDAC (errores + y -
%                        en %)
%
%      err_vector_convDAC = [errC_conv(1) . . . errC_conv(12)
%                            errCc_sub1_conv]
%
%   .err_vector_calDAC: vector con los erroees del calDAC (errores + y -
%                       en %)
%
%       err_vector_calDAC = [errC_cal(1) . . . errC_conv(8)]
%-------------------------------------------------------------------------


resol = (1/2^15) / 4; %resolucion del comparador (15bits). Dividida por 4
porque en la realidad trabajo con dos DACS: diferencial
code_out = zeros(length(Vin),8);

for j=1:1:length(Vin)

    for i=1:1:8

        code_out(j,i) = 1;    %BITi activo

        [Vconv, Vcal, Vout] = DAC(err_vector_convDAC, err_vector_calDAC,
                              zeros(1,12), code_out(j,:));

        if (Vcal - Vin(j) > resol)
            code_out(j,i) = 0;
        end

    end
end
```