

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería Electrónica

Realidad Aumentada como una Herramienta para el Análisis y Diseño de Circuitos Electrónicos Analógicos con Tecnología MOS y Circuitos Digitales Secuenciales

Proyecto Tecnológico

Fernando Daniel Ramírez Cruz

Matrícula: 2163033923

Correo electrónico: al2163033923@azc.uam.mx

Asesor: M. en C. Fernando José de Jesús Ramírez
Rojas

Profesor Titular

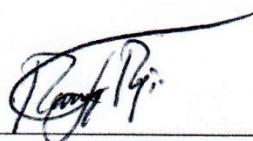
Departamento de Electrónica

Correo electrónico: frr@azc.uam.mx

Trimestre Lectivo: 21-P

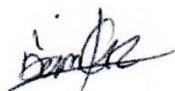
Fecha: 21 de octubre de 2021

Yo, M. en C. Fernando José de Jesús Ramírez Rojas, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



M. en C. Fernando José de Jesús Ramírez Rojas

Yo, Fernando Daniel Ramírez Cruz, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Fernando Daniel Ramírez Cruz

Dedicatoria

A mis padres:

Quienes siempre me han brindado su apoyo incondicional.

Resumen

El presente proyecto tiene como propósito mostrar, utilizando realidad aumentada, información relevante sobre los circuitos electrónicos analógicos o digitales que se estudian. Se adquieren tanto las señales más importantes del circuito electrónico como la imagen del circuito.

Los parámetros adquiridos y la imagen del circuito se envían mediante comunicación serial a la computadora, en donde los datos son procesados. Los parámetros eléctricos, gráficas e información relevante se despliegan sobre la imagen del circuito.

Este proyecto está pensado para construirse muy fácilmente sin utilizar instrumentos de laboratorio adicionales, como el osciloscopio o generador de funciones, reduciendo significativamente el tiempo de análisis del circuito electrónico a estudiar.

Tabla de Contenido

1. Introducción	38
2. Antecedentes.....	39
3. Justificación	41
4. Objetivos.....	42
5. Marco teórico.....	43
5.1 Cálculo de parámetros de un transistor MOSFET canal n de enriquecimiento	43
5.1.1 Definición de voltajes y corrientes sobre un transistor MOSFET	43
5.1.2 Modelo híbrido pi de un transistor MOSFET	45
5.1.3 Curvas características del transistor MOSFET	46
5.1.3.1 Curva característica de entrada.....	46
5.1.3.2 Curva característica de transferencia	47
5.1.3.3 Familia de curvas características de salida	47
5.1.4 Regiones de operación	48
5.1.5 Cálculo del parámetro lambda.....	49
5.1.6 Cálculo de los parámetros Kn y Vth	50
5.1.7 Cálculo del parámetro Idss	51
5.1.8 Cálculo de las ecuaciones de las curvas características de salida	52
5.1.9 Caracterización de un transistor MOSFET 2n7000 a través de la obtención de sus parámetros y gráficas.....	53
5.2 Circuito generador de señal senoidal.....	55
5.2.1 Arquitectura de un circuito DDS	55
5.2.1.1 Acumulador de fase (1) y convertidor fase-amplitud (2)	57
5.2.1.2 Convertidor digital-analógico	58
5.2.2 Propuesta para la construcción de un circuito DDS utilizando un microcontrolador y el circuito PCF8591.....	59
5.2.2.1 Acumulador de fase (1) y convertidor fase-amplitud (2)	59

5.2.2.2 Convertidor digital-analógico (3)	60
5.3 Análisis de un circuito amplificador en fuente común con un transistor MOSFET 2N7000.....	61
5.3.1 Cálculo del punto de operación	61
5.3.2 Criterios para utilizar pequeña señal	64
5.3.3 Cálculo de parámetros del modelo híbrido pi.....	64
5.3.3.1 Cálculo del parámetro gm del modelo híbrido pi	64
5.3.3.2 Cálculo del parámetro rg del modelo híbrido pi.....	66
5.3.3.3 Cálculo del parámetro rd del modelo híbrido pi.....	66
5.3.4 Análisis en pequeña señal utilizando el modelo híbrido pi simplificado	67
5.3.4.1 Cálculo de resistencias de entrada y salida	67
5.3.4.2 Cálculo de la ganancia en voltaje Av	68
5.4 Diseño de un circuito secuencial.....	69
5.4.1 Máquina de estados finitos	69
5.4.2 Modelado del circuito secuencial como una máquina de estados finitos	70
5.4.3 Diseño para la construcción de una máquina de estados finitos utilizando flip-flops tipo D.....	73
6. Desarrollo del proyecto	75
6.1. Introducción	75
6.2 Circuito Bajo Prueba (1)	79
6.2.1. Trazador de Curvas (a) y Amplificador de una Etapa (c).....	80
6.2.2 Circuito Generador de Señal Senoidal (b)	82
6.2.3 Circuito secuencial (d)	83
6.2.3.1 Máquina de estados finitos construida con compuertas lógicas y flip-flops (i)	84
6.2.3.2 Máquina de estados finitos construida con software en un microcontrolador (ii)	86
6.2.3.2.1 Definición de la máquina de estados (0).....	89
6.2.3.2.2 Subrutina de Inicialización (1)	93
6.2.3.2.3 Subrutina principal loop	95

6.2.3.2.4 Implementación de la máquina de estados (2)	96
6.2.3.2.4.1 Subrutina leer_entradas (2a).....	98
6.2.3.2.4.2 Subrutina funcion_fsm (2b)	99
6.2.3.2.4.3 Subrutinas escribir_salidas y escribir_salidas (2c)	100
6.2.3.2.5 Subrutinas adicionales de impresión a monitor serial	103
 6.3. Etapa de Adquisición de Datos (2).....	104
6.3.1. Trazador de curvas (a).....	105
6.3.1.1. Etapa de adquisición de datos utilizada para el circuito Trazador de curvas	105
6.3.1.2. Programa para adquirir señales del circuito Trazador de Curvas.....	106
6.3.1.2.1. Subrutina de inicialización (1)	110
6.3.1.2.2. Subrutina principal loop (2,3,4)	111
6.3.1.2.3. Subrutinas para la medición de curvas barriendo Vdd (2)	113
6.3.1.2.4. Subrutinas para la medición de curvas barriendo Vgg (3)	115
6.3.1.2.5. Subrutinas para la lectura, procesamiento y envío de las muestras leídas (4)	117
6.3.1.2.5.1. Subrutina para leer los canales (4a)	119
6.3.1.2.5.2. Subrutina para escalar canales (4b)	120
6.3.1.2.5.3. Subrutina para calcular las corrientes del circuito (4c)	122
6.3.1.2.5.4. Subrutina para enviar datos seriales (4d)	123
6.3.1.2.6. Subrutinas de filtrado.....	124
6.3.2. Circuito generador de señal senoidal (DDS) (b)	126
6.3.2.1. Etapa de adquisición de datos utilizado para el circuito Generador de Señal Senoidal.....	126
6.3.2.2. Programa para adquirir señales del circuito DDS	127
6.3.2.2.1. Subrutina de Inicialización (1)	129
6.3.2.2.2. Subrutinas para la medición de curva senoidal (2).....	130
6.3.2.2.3. Subrutinas para la lectura, procesamiento y envío de datos (3)	132
6.3.2.2.3.1 Subrutina para leer los canales (3a)	133
6.3.2.2.3.2 Subrutina para escalar los canales (3b).....	134
6.3.2.2.3.3 Subrutina para enviar datos seriales (3c)	135
6.3.2.2.4 Subrutinas de filtrado.....	136
6.3.3. Amplificador de una etapa (c).....	137

6.3.3.1. Etapa de adquisición de datos utilizada para el circuito Amplificador de una etapa	137
6.3.3.2. Programa para adquirir señales del amplificador de una etapa	138
6.3.3.2.1. Subrutina de Inicialización (1)	140
6.3.3.2.2. Subrutina principal loop (2,3,4,5)	141
6.3.3.2.3. Subrutinas para la medición de curvas barriendo V _{gg} (2)	143
6.3.3.2.4. Subrutinas para la medición de curvas barriendo V _{dd} (3)	144
6.3.3.2.5. Subrutinas para la medición de curvas en pequeña señal (4)	144
6.3.3.2.6. Subrutinas para la lectura, procesamiento y envío de las muestras leídas (5)	147
6.3.3.2.7 Subrutinas de filtrado.....	147
6.3.4. Circuito secuencial (d)	148
6.3.4.1 Etapa de adquisición de datos utilizada para el circuito secuencial	148
6.3.4.2. Programa para adquirir señales del circuito secuencial.....	150
6.3.4.2.1 Definición del vector de entradas para probar la máquina de estados (0)	152
6.3.4.2.2 Subrutina de inicialización (1)	154
6.3.4.2.3 Subrutina principal loop (2)	156
6.3.4.2.3.1 Subrutina para escribir entradas (2a).....	157
6.3.4.2.3.2 Subrutina para leer salida y estado (2b)	158
6.3.4.2.3.3 Subrutina enviar_datos_seriales (2c).....	160
6.3.4.2.4 Subrutinas adicionales de impresión a monitor serial	161
6.4 Computadora y Cámara de Video (3)	162
6.4.1. Imágenes y marcadores ARUCO utilizados.....	164
6.4.1.1 Imagen utilizada para el circuito Trazador de curvas (a)	165
6.4.1.2 Imagen utilizada para el circuito generador de señal senoidal (b).....	166
6.4.1.3 Imagen utilizada para el circuito Amplificador de una etapa (c)	167
6.4.1.4 Imagen utilizada para el circuito secuencial (d).....	169
6.4.2. Programas en Python	171
6.4.2.1 Trazador de Curvas (a)	174
6.4.2.1.1 Modificaciones en la clase modelo.py	175
6.4.2.1.2 Descripción de las variables de los datos recibidos desde el microcontrolador	177

6.4.2.1.3. Clase calculadora_datos_graficas.py	181
6.4.2.1.3.1 Constructor (1)	183
6.4.2.1.3.2 Método para calcular Vth, Kn e Idss (2)	184
6.4.2.1.3.3 Método para calcular puntos límite en la gráfica id vs vds (3).....	186
6.4.2.1.3.4 Métodos para calcular lambda (4)	187
6.4.2.1.3.4.1 Método “calcular_lambda” (4a).....	187
6.4.2.1.3.4.2 Métodos “obtener_ind_min” y “obtener_ind_max” (4b).....	189
6.4.2.1.3.4.3 Método “regresión_lineal_lambda” (4a).....	191
6.4.2.1.3.4.4 Método “estimate_coef” (4a).....	192
6.4.2.1.4 Modificaciones en la clase graficador.py	193
6.4.2.1.4.1 Cálculo de datos (i)	195
6.4.2.1.4.2 Gráfica 0: Voltajes en el tiempo (ii)	197
6.4.2.1.4.3 Gráfica 1: Ig vs Vgs (iii).....	199
6.4.2.1.4.4 Gráfica 2: Id vs Vgs (iv)	201
6.4.2.1.4.5 Gráfica 3: Id vs Vds (v)	203
6.4.2.1.4.6 Gráfica 4: gm vs Id (vi).....	205
6.4.2.1.4.7 Gráfica 5: rd vs Id (vii).....	207
6.4.2.1.4.8 Gráfica 6: Parámetros híbridos (viii)	209
6.4.2.2 Circuito generador de señal senoidal (b)	211
6.4.2.2.1 Modificaciones en clase modelo.py	212
6.4.2.2.2 Descripción de las variables de los datos recibidos desde el microcontrolador	214
6.4.2.2.3 Modificaciones en la clase graficador.py	216
6.4.2.2.3.1 Definición de variables (i)	218
6.4.2.2.3.2 Gráfica 0: Comparación entre voltaje y binario 1 (ii).....	219
6.4.2.2.3.3 Gráfica 1: Comparación entre voltaje y binario 2 (iii)	221
6.4.2.2.3.3 Gráfica 2: Comparación entre voltaje y binario 3 (iv)	223
6.4.2.2.3.4 Gráfica 3: Comparación entre voltaje y binario 4 (v).....	225
6.4.2.3 Amplificador de una etapa (c)	228
6.4.2.3.1 Modificaciones en la clase modelo.py	229
6.4.2.3.2 Descripción de las variables de los datos recibidos desde el microcontrolador	231
6.4.2.3.3. Clase calculadora_datos_graficas.py	235
6.4.2.3.3.1 Constructor (1)	238

6.4.2.3.3.2 Método para realizar ajuste de corrientes (2)	239
6.4.2.3.3.3 Método para calcular V_{th} , K_n e I_{dss} (3)	239
6.4.2.3.3.4 Método para calcular g_m (4).....	240
6.4.2.3.3.5 Métodos para calcular puntos sobre gráfica I_d vs V_{gs} (5).....	241
6.4.2.3.3.6 Método para calcular puntos límite en la gráfica I_d vs V_{ds} (6)	242
6.4.2.3.3.7 Métodos para calcular lambda (7)	242
6.4.2.3.3.8 Método para calcular resistencias del modelo híbrido pi (8)	242
6.4.2.3.3.9 Método para calcular parámetros del punto de operación y parámetros híbridos asociados (9)	243
6.4.2.3.3.10 Método para calcular punto de operación, máximos y mínimos sobre la gráfica que muestra el comportamiento del transistor como amplificador y calcular ganancia experimental (10)	244
6.4.2.3.3.11 Método para calcular puntos que delimitan regiones de operación y puntos sobre la recta estática sobre las curvas I_d vs V_{ds} (11)	246
6.4.2.3.4 Modificaciones en la clase graficador.py	249
6.4.2.3.4.1 Cálculo de datos (i).....	251
6.4.2.3.4.2 Gráfica 0: Voltajes en el tiempo (ii)	254
6.4.2.3.4.3 Gráfica 1: I_g vs V_{gs} (iii).....	255
6.4.2.3.4.4 Gráfica 2: I_d vs V_{gs} (iv)	256
6.4.2.3.4.5 Gráfica 3: I_d vs V_{ds} (v)	258
6.4.2.3.4.6 Gráfica 4: g_m vs I_d (vi).....	261
6.4.2.3.4.7 Gráfica 5: r_d vs I_d (vii).....	262
6.4.2.3.4.8 Gráfica 6: Voltajes del amplificador en fuente común (viii)	263
6.4.2.3.4.9 Gráfica 7: Parámetros híbridos y punto de operación(ix)	265
6.4.2.4 Circuito secuencial (d).....	268
6.4.2.4.1 Modificaciones en la clase modelo.py	269
6.4.2.4.2 Descripción de las variables de los datos recibidos desde el microcontrolador	271
6.4.2.4.3. Clase fsm.py	273
6.4.2.4.3.1 Constructor.....	275
6.4.2.4.4. Clase calculadora_datos_graficas.py	278
6.4.2.4.4.1 Métodos para convertir números decimales a binario sin y con condiciones X de no importa (1)	280
6.4.2.4.4.1.1 Método “decimal a binario” (1a).....	280
6.4.2.4.4.1.2 Método “list_duplicates_of” (1b)	280

6.4.2.4.4.1.3 Método “obtener_bits_cambiantes” (1c).....	281
6.4.2.4.4.1.4 Método “obtener_str_de_binario_transicion” (1d)	282
6.4.2.4.4.2 Métodos para obtener las coordenadas de diferentes elementos del diagrama de estados (sin utilizar el paquete graphviz) (2)	283
6.4.2.4.4.2.1 Método “obtener_puntos_lineas_transicion” (2a).....	283
6.4.2.4.4.2.2 Método “circle_line_segment_intersection” (2b)	287
6.4.2.4.4.2.3 Método “ec_recta_dos_puntos” (2c).....	288
6.4.2.4.4.2.4 Método “ec_recta_punto_pendiente_perpendicular” (2d).....	289
6.4.2.4.4.2.5 Método “puntos_equidistantes_sobre_recta_desde_punto” (2e)	
.....	290
6.4.2.4.4.3 Métodos auxiliares para calcular coordenadas de diferentes elementos en diagramas (3)	291
6.4.2.4.4.3.1 Método “get_intersections” (3a).....	291
6.4.2.4.4.3.3 Método “obtener_coordenadas_labels” (3b)	292
6.4.2.4.5. Modificaciones en la clase graficador.py	293
6.4.2.4.5.1 Definición de variables (i)	295
6.4.2.4.5.2 Gráfica 0: Señales binarias en el tiempo.....	296
6.4.2.4.5.3 Gráfica 1: Tabla de estados binarios.....	298
6.4.2.4.5.4 Gráfica 2: Diagrama de estados (1)	302
6.4.2.4.5.5 Gráfica 3: Diagrama de estados (2)	306
6.4.2.4.5.5.1 Generación del diagrama de estados utilizando graphviz (i) .	306
6.4.2.4.5.5.2 Formato de las dimensiones de la imagen del diagrama de estados (ii)	308
6.4.2.4.5.5.3 Concatenación de título y leyenda a la imagen del diagrama de estados (iii)	310
6.4.2.4.5.5.4 Eliminación de imágenes auxiliares (iv)	311
6.4.2.4.5.5.5 Presentación de la gráfica	312
6.4.2.4.5.6 Gráfica 4: Diagrama de transiciones de la máquina de estados finitos	313
7. Resultados.....	317
7.1. Trazador de Curvas (a)	318
7.1.1 Presentación de la imagen del circuito con las gráficas	319
7.1.2. Gráfica 0: Voltajes en el tiempo.....	320

7.1.3. Gráfica 1: Ig vs Vgs.....	321
7.1.4. Gráfica 2: Id vs Vgs.....	322
7.1.5. Gráfica 3: Id vs Vds.....	323
7.1.6. Gráfica 4: gm vs Id	324
7.1.7. Gráfica 5: rd vs Id.....	325
7.1.8. Gráfica 6: Parámetros híbridos.....	326
 7.2. Circuito generador de señal senoidal (b)	327
7.2.1 Presentación de la imagen del circuito con las gráficas	327
7.2.2. Gráfica 0: Comparación entre voltaje y binario 1	329
7.2.3. Gráfica 1: Comparación entre voltaje y binario 2	330
7.2.4. Gráfica 2: Comparación entre voltaje y binario 3	331
7.2.5 Gráfica 3: Comparación entre voltaje y binario 4	332
 7.3 Amplificador de una etapa (c).....	333
7.3.1 Presentación de la imagen del circuito con las gráficas	333
7.3.2 Gráfica 0: Voltajes en el tiempo.....	335
7.3.3 Gráfica 1: Ig vs Vgs.....	336
7.3.4 Gráfica 2: Id vs Vgs.....	337
7.3.5 Gráfica 3: Id vs Vds.....	338
7.3.6 Gráfica 4: gm vs Id	339
7.3.7 Gráfica 5: rd vs Id.....	340
7.3.8 Gráfica 6: Voltajes del amplificador en fuente común.....	341
7.3.9 Gráfica 7: Parámetros híbridos y punto de operación	342
 7.4. Circuito secuencial (d)	344
7.4.1 Presentación de la imagen del circuito con las gráficas	344
7.4.2 Gráfica 0: Señales digitales binarias en el tiempo	347
7.4.3 Gráfica 1: Tabla de estados binarios.....	349
7.4.4 Gráfica 2: Diagrama de estados 1	351
7.4.5 Gráfica 3: Diagrama de estados 2	353
7.4.6 Gráfica 4: Diagrama de transiciones de la máquina de estados finitos.....	355

8. Análisis y discusión de resultados	357
8.1. Trazador de Curvas (a)	358
8.1.1 Comparación de las gráficas resultantes de las curvas características de entrada, transferencia y salida.....	358
8.1.2 Comparación de las gráficas resultantes para los parámetros gm y rd	360
8.1.3 Comparación de resultados de los parámetros del transistor MOSFET 2N7000	361
8.2. Circuito generador de señal senoidal (b)	362
8.2.1 Comparación de la gráfica resultante con la forma de onda senoidal.....	362
8.3. Amplificador de una etapa (c).....	364
8.3.1 Comparación de las gráficas resultantes para las curvas características de entrada, transferencia y salida.....	364
8.3.2 Comparación de las gráficas resultantes de los parámetros gm y rd.....	367
8.3.3 Comparación de resultados de los parámetros del transistor MOSFET 2N7000	368
8.3.3.1 Comparación de resultados de los parámetros híbridos entre los resultados para circuitos trazador de curvas (a) y los resultados para el amplificador de una etapa (c).....	369
8.3.4 Comparación de resultados de los parámetros en el punto de operación del amplificador de una etapa.....	370
8.4. Circuito secuencial (d)	371
8.4.1 Comparación de los resultados del funcionamiento de la máquina de estados finitos	371
8.4.2 Comparación de las gráficas resultantes que contienen los diagramas de estados	373
9. Conclusiones	375
10. Referencias	376
10.1 Software utilizado	377

11. Entregables	378
11.1 Programas base	378
11.1.1 Programa base de Arduino (1)	379
11.1.2 Programa base de Python (2).....	381
11.1.2.1 Clase main.py (2a)	381
11.1.2.2 Clase modelo.py (2b)	384
11.1.2.2.1 Clase comunicacion.py (2bi)	390
11.1.2.2.2 Clase muestras.py (2bii).....	393
11.1.2.2.3 Clase marcadores.py (2biii).....	402
11.1.2.2.4 clasificador.py (2biv).....	411
11.1.2.2.5 graficador.py (2bv).....	416
11.1.2.2.6 display.py (2bvi).....	427
11.1.2.2.7 asignador.py (2bvii)	440
11.1.2.3 vista.py (2c)	443
11.1.2.4 controlador.py (2d)	445
11.1.2.4.1 Clase eventomouse.py (2di).....	447
11.2 Diagramas esquemáticos y programas completos desarrollados	450
11.2.1 Circuito Trazador de curvas (a)	451
11.2.1.1 Diagrama esquemático (1).....	451
11.2.1.2 Programa de Arduino (2).....	452
11.2.1.3 Programas en Python (3)	460
11.2.1.3.1 Clase calculadora_datos_graficas.py (3i)	461
11.2.1.3.1 Clase graficador.py (3ii)	473
11.2.2 Circuito generador de señal senoidal (b)	486
11.2.2.1 Diagrama esquemático (1).....	486
11.2.2.2 Programa de Arduino (2).....	487
11.2.2.3 Programas en Python (3)	493
11.2.2.3.1 Clase graficador.py (3i)	494
11.2.3 Circuito amplificador de una etapa (c).....	505
11.2.3.1 Diagrama esquemático (1).....	505
11.2.3.2 Programa de Arduino (2).....	506

11.2.3.3 Programas en Python (3)	515
11.2.3.3.1 Clase calculadora_datos_graficas.py (3i)	516
11.2.3.3.2 Clase graficador.py (3ii)	536
11.2.4 Circuito secuencial (d)	552
11.2.4.1 Diagrama esquemático (1).....	553
11.2.4.2 Programa de Arduino (2).....	554
11.2.4.2.1 Programa de Arduino. Implementación de la máquina de estados (2i)	554
11.2.4.2.2 Programa de Arduino. Prueba y lectura de la máquina de estados (2ii)	564
11.2.4.3 Programas en Python (3)	572
11.2.4.3.1 Clase fsm.py (3i).....	573
11.2.4.3.2 Clase calculadora_datos_graficas.py (3ii)	577
11.2.4.3.3 Clase graficador.py (3iii)	588

Índice de figuras

Figura 1. Símbolo de un transistor MOSFET canal n de enriquecimiento.....	43
Figura 2 Convenciones de voltaje y corriente para un transistor MOSFET de enriquecimiento.	44
Figura 3. Modelo híbrido PI de un transistor MOSFET canal n de enriquecimiento.....	45
Figura 4. Curva característica de entrada para un transistor MOSFET canal n de enriquecimiento.	46
Figura 5. Curva característica de transferencia para un transistor MOSFET canal n de enriquecimiento.	47
Figura 6. Familia de curvas características de salida para un transistor MOSFET canal n de enriquecimiento.	47
Figura 7. Regiones de operación para un transistor MOSFET canal n de enriquecimiento	48
Figura 8. Gráfica que muestra el cálculo del parámetro lambda sobre la familia de curvas características de salida.	49
Figura 9. Medición del parámetro Kn y Vth sobre la curva característica de transferencia.	50
Figura 10. Circuito de polarización propuesto para la caracterización del transistor MOSFET canal n de enriquecimiento.	53
Figura 11. Diagrama a bloques de un circuito DDS genérico. (Fuente: Analog Devices) ..	55
Figura 12. División del diagrama a bloques en tres partes diferentes (Fuente: Analog Devices).	55
Figura 13. Flujo de una señal a través de un circuito DDS. (Fuente: Analog Devices)	56
Figura 14. Ubicación de los bloques Acumulador de fase (1) y Convertidor de fase a amplitud (2).....	57
Figura 15. Rueda de fase digital. (Fuente: Analog Devices)	57
Figura 16. Ubicación del bloque Convertidor Digital-Analógico (3).....	58

Figura 17. Diagrama a bloques del sistema propuesto para la construcción de un circuito DDS	59
Figura 18. Fragmento de código en c para la ejecución del circuito DDS (acumulador de fase (1) y convertidor de fase a amplitud (2))	59
Figura 19. Circuito de polarización propuesto.	61
Figura 20. Punto de operación y recta de carga estática sobre la gráfica de curvas características de salida.....	63
Figura 21. Modelo híbrido π simplificado del circuito propuesto de la figura 16.....	67
Figura 22. Diagrama de estados del problema propuesto.	70
Figura 23. Diagrama a bloques con las salidas y entradas para la máquina de estados ..	71
Figura 24. Diagrama a bloques del proyecto propuesto.....	75
Figura 25. Diagrama del proyecto propuesto	76
Figura 26. Ubicación del bloque del Circuito Bajo Prueba (1).....	79
Figura 27. Circuito propuesto con un transistor MOSFET canal n de enriquecimiento 2N7000.....	80
Figura 28. Circuito propuesto utilizando un microcontrolador y un PCF8591 para la construcción de un circuito DDS.	82
Figura 29. Diagrama de estados utilizado para la construcción de un circuito secuencial.	83
Figura 30. Diagrama a bloques con las salidas y entradas para la máquina de estados ..	84
Figura 31. Circuito secuencial implementado por hardware (i) utilizando compuestas lógicas y flip-flops tipo D.....	85
Figura 32. Circuito secuencial implementado por software (ii) utilizando un microcontrolador.....	86
Figura 33. Diagrama a bloques del programa completo de Arduino para la implementación del Circuito Secuencial (4).	87
Figura 34. División del diagrama a bloques en dos partes principales.....	87
Figura 35. Diagrama a bloques del funcionamiento del programa que implementa la máquina de estados finitos.....	88

Figura 36. Diagrama a bloques de las funciones que realiza la parte del programa Implementación de la máquina de estados (2).....	88
Figura 37. Ubicación del bloque “Definición de la máquina de estados”	89
Figura 38. Fragmento de código que define a la máquina de estados finitos.	90
Figura 39. Ubicación del bloque de Inicialización (1)	93
Figura 40. Subrutina de inicialización	94
Figura 41. En la subrutina loop se ubica el bloque Revisión de reset e inicialización de la máquina de estados (2).....	95
Figura 42. Subrutina principal “loop”.....	95
Figura 43. Ubicación del bloque Implementación de la máquina de estados (2).....	96
Figura 44. Subrutina para la implementación de la máquina de estados	97
Figura 45. Diagrama a bloques de la función “Implementación de la máquina de estados”	97
Figura 46. Ubicación del bloque Leer entradas (2a).	98
Figura 47. Subrutina “leer_entradas”	98
Figura 48. Ubicación del bloque Funcionamiento de la máquina de estados (2b).....	99
Figura 49. Subrutina “funcion_fsm”.....	99
Figura 50. Ubicación del bloque Escritura de salidas y estado de la máquina (2c) (parte de escritura de la salida de la máquina de estados finitos).....	100
Figura 51. Subrutina “escribir_salidas”	101
Figura 52. Ubicación del bloque Escritura de salidas y estado de la máquina (2c) (parte de escritura del estado de la máquina de estados finitos)	101
Figura 53. Subrutina “escribir_estados”	102
Figura 54. Ubicación de las subrutinas de impresión.....	103
Figura 55. Ubicación del bloque Etapa de adquisición de datos (2).....	104
Figura 56. Etapa de adquisición de datos utilizada para el circuito Trazador de curvas (a).	105

Figura 57. Diagrama a bloques del programa completo de Arduino para adquirir señales del circuito Trazador de Curvas (a).....	106
Figura 58. División del diagrama a bloques en cuatro partes principales.....	107
Figura 59. Diagrama a bloques del funcionamiento del programa para el circuito Trazador de Curvas (a).....	107
Figura 60. Diagrama a bloques de las funciones que realiza la parte del programa de medición de curvas cambiando Vdd (2).....	108
Figura 61. Diagrama a bloques de las funciones que realiza la parte del programa de medición de curvas cambiando Vgg (3).....	108
Figura 62. Diagrama a bloques de las funciones que realiza la parte de lectura, procesamiento y envío de datos (4).....	109
Figura 63. Ubicación del bloque Inicialización (1).	110
Figura 64. Subrutina de Inicialización	110
Figura 65. Ubicación de los bloques Medición de curvas cambiando Vdd (2) y Medición de curvas cambiando Vgg (3).	111
Figura 66. Subrutina principal “loop”.....	111
Figura 67. Ubicación del bloque Fijar voltaje Vgg (2ii).	113
Figura 68. Subrutina “establecer_voltaje_Vgg”	113
Figura 69. Ubicación del bloque Barrer voltaje Vdd (2iii).	114
Figura 70. Subrutina “barrer_voltaje_Vdd”.....	114
Figura 71. Ubicación del bloque Fijar voltaje Vdd	115
Figura 72. Subrutina “establecer_voltajeVdd”.	115
Figura 73. Ubicación del bloque Barrer voltaje Vgg (3ii).	116
Figura 74. Subrutina “barrer_voltaje_Vgg”.....	116
Figura 75. Ubicación del bloque Lectura, procesamiento y envío de datos (4).....	117
Figura 76. Subrutina “leer_procesar_enviar_muestras”	117
Figura 77. Ubicación del bloque Leer cananles (4a).	119

Figura 78. Subrutina “lectura_canales”	119
Figura 79. Ubicación del bloque Escalar canales (4b).	120
Figura 80. Subrutina “escalar_canales”	120
Figura 81. Subrutina “scaling”	121
Figura 82. Ubicación del bloque Calcular corrientes (4c).....	122
Figura 83. Subrutina “calcular_corrientes”.....	122
Figura 84. Ubicación del bloque Enviar datos seriales (4d).	123
Figura 85. Subrutina “enviar_datos_seriales”.....	123
Figura 86. Ubicación de las subrutinas de filtrado.....	124
Figura 87. Subrutina “agregar_lecturas_buffer”.....	124
Figura 88. Subrutina “promediador_lecturas_circular”.....	125
Figura 89. Etapa de adquisición de datos utilizada para la adquisición de datos del circuito generador de señal senoidal (2).	126
Figura 90. Diagrama a bloques del programa completo para adquirir señales del circuito Generador de Señal Senoidal (b).	127
Figura 91. División del diagrama a bloques en tres partes principales.....	127
Figura 92. Diagrama a bloques del funcionamiento del programa para el circuito trazador de curvas.	128
Figura 93. Diagrama a bloques de las funciones que realiza la parte de lectura, procesamiento y envío de datos (3).....	129
Figura 94. Ubicación del bloque de Inicialización (1).	129
Figura 95. Ubicación del bloque Medición de curva senoidal (2).....	130
Figura 96. Subrutina “loop”.....	130
Figura 97. Subrutina “barrer_voltaje_senoidal”.	130
Figura 98. Subrutina “establecer_voltaje_senoidal”.....	131
Figura 99. Ubicación del bloque Lectura, procesamiento y envío de datos (3).....	132

Figura 100. Subrutina “leer_procesar_enviar_muestras”	132
Figura 101. Ubicación del bloque Leer canales(3a).....	133
Figura 102. Subrutina “lectura_canales”	133
Figura 103. Ubicación del bloque Escalar canales (3b).....	134
Figura 104. Subrutina “escalar_canales”.....	134
Figura 105. Ubicación del bloque Enviar datos seriales (3c).....	135
Figura 106. Subrutina “enviar_datos_seriales”.....	135
Figura 107. Ubicación de las subrutinas de filtrado.....	136
Figura 108. Etapa de adquisición de datos utilizada para la adquisición de datos del circuito Amplificador de una etapa (3)	137
Figura 109. Diagrama a bloques del programa completo en Arduino para adquirir señales del circuito Amplificador de una Etapa (c).	138
Figura 110. División del diagrama a bloques en cinco partes principales.	138
Figura 111. Diagrama a bloques del funcionamiento para el circuito Amplificador de una etapa (c).....	139
Figura 112. Diagrama a bloques de las funciones que realiza la parte del programa Medición de curvas en pequeña señal (4).....	140
Figura 113. Ubicación del bloque de Inicialización (1)	140
Figura 114. Ubicación de los bloques Medición de curvas cambiando Vdd, Medición de curvas cambiando Vgg y Medición de curvas en pequeña señal (4).....	141
Figura 115. Subrutina principal “loop”.....	142
Figura 116. Ubicación del bloque Medición de curvas cambiando Vdd (2).....	143
Figura 117. Ubicación del bloque Medición de curvas cambiando Vgg (3).....	144
Figura 118. Ubicación del bloque Fijar voltaje Vdd (4i).....	144
Figura 119. Ubicación del bloque Barrer voltaje Vgg en pequeña señal (4ii).....	145
Figura 120. Subrutina “barrer_voltaje_Vgg_pequena_senal”.....	145
Figura 121. Ubicación del bloque Lectura, procesamiento y envío de datos (5).....	147

Figura 122. Ubicación de las subrutinas de filtrado.....	147
Figura 123. Etapa de adquisición de datos para el circuito secuencial (d) implementado por hardware (i).	148
Figura 124. Etapa de adquisición de datos para el circuito secuencial (d) implementado por software (ii).....	149
Figura 125. Diagrama a bloques del programa completo en Arduino para adquirir señales del Circuito secuencial (d)	150
Figura 126. División del diagrama a bloques en dos partes principales.	150
Figura 127. Diagrama a bloques del funcionamiento del programa para probar y leer la máquina de estados finitos.....	151
Figura 128. Diagrama a bloques de las funciones que realiza la parte del programa Prueba y lectura de la máquina de estados (2).....	152
Figura 129. Fragmento de código que define al vector de entradas para probar la máquina de estados.	152
Figura 130. Ubicación del bloque de Inicialización (1).	154
Figura 131. Subrutina “setup”.	155
Figura 132. Ubicación del bloque Prueba y lectura de la máquina de estados (2).	156
Figura 133. Subrutina principal “loop”.....	156
Figura 134. Ubicación del bloque Escribir entradas (2a).....	157
Figura 135. Subrutina “escribir_entradas”.	158
Figura 136. Ubicación del bloque Leer salidas y estado de la máquina de estados (2b) (parte de lectura de la salida de la máquina de estados finitos).	158
Figura 137. Subrutina “leer_salidas”.....	159
Figura 138. Ubicación del bloque Leer salidas y estado de la máquina de estados (2b) (parte de la lectura del estado de la máquina de estados finitos).	159
Figura 139. Subrutina “leer_estados”.....	159
Figura 140. Ubicación del bloque Enviar datos seriales (2c).	160
Figura 141. Subrutina “enviar_datos_seriales”.....	160

Figura 142. Ubicación de las subrutinas de impresión.....	161
Figura 143. Ubicación del bloque Cámara de video y computadora (3).....	162
Figura 144. Marcadores ARUCO utilizados.....	164
Figura 145. Imagen utilizada para el circuito Trazador de curvas (a).....	165
Figura 146. Imagen utilizada para el circuito generador de señal senoidal (b)	166
Figura 147. Imagen utilizada para el circuito Amplificador de una etapa (c)	167
Figura 148. Imagen utilizada para el circuito secuencial (d) implementado por hardware (i).	169
Figura 149. Imagen utilizada para el circuito secuencial (d) implementado por software (ii).	170
Figura 150. Diagrama de clases del programa base.....	171
Figura 151. Diagrama de clases con la clase claculadora_datos_graficas.py.	173
Figura 152. Diagrama de clases para el circuito trazador de curvas (a).	174
Figura 153. Ubicación de la clase modelo.py.	175
Figura 154. Métodos, atributos y propiedades del método modelo.py.	175
Figura 155. Definición de los títulos de las llaves para los vectores de muestras y número de muestras.....	176
Figura 156. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.	176
Figura 157. Ubicación de la clase calculadora_datos_graficas.py.	181
Figura 158. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.	181
Figura 159. Método constructor.	183
Figura 160. Método “calcular_Vth_Kn_Idss”.	185
Figura 161. Método “calcular_puntos_limite_grafica_ids_vds”....	186
Figura 162. Método “calcular_lambda”....	188
Figura 163. Métodos “obtener_ind_min” y “obtener_ind_max”....	190

Figura 164. Método “regresión_lineal_lambda”.....	191
Figura 165. Método “estimate_coef”.....	192
Figura 166. Ubicación de la clase graficador.py.....	193
Figura 167. Métodos, atributos y propiedades de la clase graficador.py.....	193
Figura 168. Fragmento de código para definir las variables a utilizar.....	195
Figura 169. Fragmento de código de para el cálculo los datos (i) necesarios para las gráficas.	196
Figura 170. Fragmento de código para la Gráfica 0: Voltajes en el tiempo.....	197
Figura 171. Gráfica 0: Voltajes en el tiempo.....	198
Figura 172. Fragmento de código para la Gráfica 1: Ig vs Vgs.....	199
Figura 173. Gráfica 1: Ig vs Vgs.....	200
Figura 174. Fragmento de código para la Gráfica 2: Id vs Vgs.....	201
Figura 175. Gráfica 2: Id vs Vds.....	202
Figura 176. Fragmento de código para la Gráfica 3: Id vs Vds.....	203
Figura 177. Gráfica 3: Id vs Vds.....	204
Figura 178. Fragmento de código para la Gráfica 4: gm vs Id.....	205
Figura 179. Gráfica 4: gm vs Id.....	206
Figura 180. Fragmento de código para la Gráfica 5: rd vs Id.....	207
Figura 181. Gráfica 5: rd vs Id.....	208
Figura 182. Fragmento de código para la Gráfica 6: Parámetros híbridos.....	209
Figura 183. Gráfica 6: Parámetros híbridos.....	210
Figura 184. Diagrama de clases para el circuito generador de señal senoidal (b).....	211
Figura 185. Ubicación de la clase modelo.py.....	212
Figura 186. Métodos, atributos y propiedades del método modelo.py.	212

Figura 187. Definición de los títulos de las llaves para los vectores de muestras y número de muestras	213
Figura 188. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.	213
Figura 189. Ubicación de la clase graficador.py.....	216
Figura 190. Métodos, atributos y propiedades de la clase graficador.py.....	216
Figura 191. Fragmento de código para definir las variables a utilizar.....	218
Figura 192. Fragmento de código para la Gráfica 0: Comparación entre voltaje y binario (1).....	219
Figura 193. Gráfica 0: Comparación entre voltaje y binario (1).....	220
Figura 194. Fragmento de código para la Gráfica 1: Comparación entre voltaje y binario (2).....	221
Figura 195. Gráfica 1: Comparación entre voltaje y binario (2).....	222
Figura 196. Fragmento de código para la Gráfica 2: Comparación entre voltaje y binario (3).....	224
Figura 197. Gráfica 2: Comparación entre voltaje y binario (3).....	224
Figura 198. Fragmento de código para la Gráfica 3: Comparación entre voltaje y binario (4).....	226
Figura 199. Gráfica 3: Comparación entre voltaje y binario (3).....	227
Figura 200. Diagrama de clases para el circuito Amplificador de una etapa (c).	228
Figura 201. Ubicación de la clase modelo.py.....	229
Figura 202. Métodos, atributos y propiedades del método modelo.py.	229
Figura 203. Definición de los títulos de las llaves para los vectores de muestras y número de muestras	230
Figura 204. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.	230
Figura 205. Ubicación de la clase calculadora_datos_gráficas.py.....	235

Figura 206. Métodos, atributos y propiedades de la clase calculadora_datos_graficas.py.....	236
Figura 207. Método constructor.....	238
Figura 208. Método “ajustar_id_con_resistencias”.....	239
Figura 209. Método “calcular_gm”.....	240
Figura 210. Método “puntos_sobre_grafica_ids_vgs”.....	241
Figura 211. Método “calcular_resistencias_hibridas”.....	242
Figura 212. Método “calcular_punto_operación_y_parametros_hibridos”.....	243
Figura 213. Método “calcular_punto_operacion_puntos_min_max_ganancia_experimental”.....	245
Figura 214. Método “calcular_puntos_límite_y_operacion_sobre_grafica_ids_vds”.....	247
Figura 215. Ubicación de la clase graficador.py.....	249
Figura 216. Métodos, atributos y propiedades de la clase graficador.py.....	249
Figura 217. Fragmento de código para definir las variables a utilizar.....	251
Figura 218. Fragmento de código para el cálculo de datos (i) necesarios para las gráficas.	253
Figura 219. Gráfica 0: Voltajes en el tiempo.....	254
Figura 220. Gráfica 1: Ig vs Vgs.....	255
Figura 221. Fragmento de código para la Gráfica 2: Id vs Vgs.....	256
Figura 222. Gráfica 2: Id vs Vgs.....	257
Figura 223. Fragmento de código para la Gráfica 3: Id vs Vds.....	259
Figura 224. Gráfica 3: Id vs Vds.....	260
Figura 225. Gráfica 4: gm vs Id.....	261
Figura 226. Gráfica 5: rd vs Id.....	262
Figura 227. Fragmento de código para la Gráfica 6: Voltajes del amplificador en fuente común.....	263

Figura 228. Gráfica 6: Voltajes del amplificador en fuente común.....	264
Figura 229. Fragmento de código para la Gráfica 7: Parámetros híbridos y punto de operación.....	266
Figura 230. Gráfica 7: Parámetros híbridos y punto de operación.	267
Figura 231. Diagrama de clases para el circuito secuencial (d).....	268
Figura 232. Ubicación de la clase modelo.py.	269
Figura 233. Métodos, atributos y propiedades del método modelo.py.	269
Figura 234. Definición de los títulos de llaves para los vectores de muestras y número de muestras.	270
Figura 235. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.	270
Figura 236. Ubicación de la clase fsm.py.	273
Figura 237. Métodos, atributos y propiedades de la clase fsm.py.	273
Figura 238. Método “ <u>__init__</u> ”	276
Figura 239. Ubicación de la clase calculadora_datos_gráficas.py.	278
Figura 240. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.	278
Figura 241. Método “decimal a binario”.	280
Figura 242. Método “list_duplicates_of”.....	280
Figura 243. Método “obtener_bits_cambiantes”.....	281
Figura 244. Método “obtener_str_de_binario_transicion”.	282
Figura 245. Método “obtener_puntos_lineas_transicion”.....	286
Figura 246. Ejemplo de para el método “obtener_puntos_lineas_transición”.	286
Figura 247. Método “circle_line_segment_intersection”.....	288
Figura 248. Método “ec_recta_dos_puntos”.....	289
Figura 249. Método “ec_recta_punto_pendiente_perpendicular”.....	289

Figura 250. Método “puntos_equidistantes_sobre_recta_desde_punto”.....	290
Figura 251. Método “get_intersections”.....	291
Figura 252. Método ”obtener_coordenadas_labels”.....	292
Figura 253. Ubicación de la clase graficador.py.....	293
Figura 254. Métodos, atributos y propiedades de la clase graficador.py.....	293
Figura 255. Fragmento de código para definir las variables a utilizar.....	295
Figura 256. Fragmento de código para la Gráfica 0: Señales binarias en el tiempo.....	296
Figura 257. Gráfica 0: Señales binarias en el tiempo.....	297
Figura 258. Fragmento de código para la Gráfica 1: Tabla de estados binarios	300
Figura 259. Gráfica 1: Tabla de estados binarios	301
Figura 260. Fragmento de código para la Gráfica 2: Diagrama de estados 1.....	304
Figura 261. Gráfica 2: Diagrama de estados 1.....	305
Figura 262. Fragmento de código para la Gráfica 3: Diagrama de estados 2 (parte i). ...	307
Figura 263. Fragmento de código para la Gráfica 3: Diagrama de estados 2 (parte ii). ..	309
Figura 264. Fragmento de código para la Gráfica 3: Diagrama de estados 2 (parte iii)...	311
Figura 265. Fragmento de código para la Gráfica 3: Diagrama de estados 2 (parte iv). .	311
Figura 266. Gráfica 3: Diagrama de estados 2.....	312
Figura 267. Fragmento de código para la Gráfica 4: Diagrama de transiciones de la máquina de estados finitos.....	315
Figura 268. Gráfica de transiciones de la máquina de estados finitos.	316
Figura 269. Ventana inicial con la imagen del circuito.	319
Figura 270. Ventanas desplegadas de todas las gráficas del circuito.	319
Figura 271. Presentación de la Gráfica 0: Voltajes en el tiempo.	320
Figura 272. Presentación de la Gráfica 1: Ig vs Vgs.	321
Figura 273. Presentación de la Gráfica 2: Id vs Vds.	322

Figura 274. Presentación de la Gráfica 3: Id vs Vds.	323
Figura 275. Presentación de la Gráfica 4: gm vs Id	324
Figura 276. Presentación de la Gráfica 5: rd vs Id.	325
Figura 277. Presentación de la Gráfica 6: Parámetros híbridos.	326
Figura 278. Ventana inicial con la imagen del circuito.	327
Figura 279. Ventanas desplegadas de todas las gráficas del circuito.	328
Figura 280. Presentación de la Gráfica 0: Comparación entre voltaje y binario 1	329
Figura 281. Presentación de la Gráfica 1: Comparación entre voltaje y binario 2	330
Figura 282. Presentación de la Gráfica 2: Comparación entre voltaje y binario 3	331
Figura 283. Presentación de la Gráfica 3: Comparación entre voltaje y binario 4	332
Figura 284. Ventana inicial con la imagen del circuito.	333
Figura 285. Ventanas desplegadas de todas las gráficas del circuito.	334
Figura 286. Presentación de la Gráfica 0: Voltajes en el tiempo.	335
Figura 287. Presentación de la Gráfica 1: Ig vs Vgs.	336
Figura 288. Presentación de la Gráfica 2: Id vs Vgs.	337
Figura 289. Presentación de la Gráfica 3: Id vs Vds.	338
Figura 290. Presentación de la Gráfica 4: gm vs Id.	339
Figura 291. Presentación de la Gráfica 5: rd vs Id.	340
Figura 292. Presentación de la Gráfica 6: Voltajes del amplificador en fuente común.	341
Figura 293. Presentación de la Gráfica 7: Parámetros híbridos y punto de operación	342
Figura 294. Ventana inicial con la imagen del circuito (circuito secuencial implementado por hardware).	344
Figura 295. Ventana inicial con la imagen del circuito (circuito secuencial implementado por software)	345
Figura 296. Ventanas desplegadas de todas las gráficas del circuito (circuito secuencial implementado por hardware)	345

Figura 297. Ventanas desplegadas de todas las gráficas del circuito (circuito secuencial implementado por software).....	346
Figura 298. Presentación de la Gráfica 0: Señales digitales en el tiempo (circuito secuencial implementado por hardware).....	347
Figura 299. Presentación de la Gráfica 0: Señales digitales en el tiempo (circuito secuencial implementado por software)	347
Figura 300. Presentación de la Gráfica 1: Tabla de estados binarios (circuito secuencial implementado por hardware).	349
Figura 301. Presentación de la Gráfica 1: Tabla de estados binarios (circuito secuencial implementado por software).....	349
Figura 302. Presentación de la Gráfica 2: Diagrama de estados 1 (circuito secuencial implementado por hardware).	351
Figura 303. Presentación de la Gráfica 2: Diagrama de estados 1 (circuito secuencial implementado por software).....	351
Figura 304. Presentación de la Gráfica 3: Diagrama de estados 2 (circuito secuencial implementado por hardware)	353
Figura 305. Presentación de la Gráfica 3: Diagrama de estados 2 (circuito secuencial implementado por software).....	353
Figura 306. Presentación de la Gráfica 4: Diagrama de transiciones de la máquina de estados finitos (circuito secuencial implementado por hardware).....	355
Figura 307. Presentación de la Gráfica 4: Diagrama de transiciones de la máquina de estados finitos (circuito secuencial implementado por software).	355
Figura 308. Comparación de la forma de onda del valor binario enviado con el voltaje de salida.....	362
Figura 309. Comparación voltaje a la salida vs valor binario.....	363
Figura 310. Diagrama a bloques del programa base de Arduino.	379
Figura 311. Programa completo base de Arduino.....	380
Figura 312. Diagrama de clases completo del programa base de Python.	381
Figura 313. Ubicación de la clase main.py.....	381
Figura 314. Código completo de la clase main.py.....	383

Figura 315. Ubicación de la clase modelo.py	384
Figura 316. Métodos, atributos y propiedades de la clase modelo.py.....	384
Figura 317. Código completo de la clase modelo.py.....	389
Figura 318. Ubicación de la clase comunicacion.py.....	390
Figura 319. Métodos, atributos y propiedades de la clase comunicación.py.....	390
Figura 320. Código completo de la clase comunicacion.py	392
Figura 321. Ubicación de la clase muestras.py.....	393
Figura 322. Métodos, atributos y propiedades de la clase muestras.py.....	393
Figura 323. Código completo de la clase muestras.py.....	401
Figura 324. Ubicación de la clase marcadores.py.....	402
Figura 325. Métodos, atributos y propiedades de la clase marcadores.py.....	402
Figura 326. Código completo de la clase marcadores.py	410
Figura 327. Ubicación de la clase clasificador.py.....	411
Figura 328. Métodos, atributos y propiedades de la clase clasificador.py.....	411
Figura 329. Código completo de la clase clasificador.py	415
Figura 330. Ubicación de la clase graficador.py.....	416
Figura 331. Métodos, atributos y propiedades de la clase graficador.py.....	416
Figura 332. Código completo de la clase graficador.py	426
Figura 333. Ubicación de la clase display.py.....	427
Figura 334. Métodos, atributos y propiedades de la clase display.py.....	428
Figura 335. Código completo de la clase display.py	439
Figura 336. Ubicación de la clase asignador.py.....	440
Figura 337. Métodos, atributos y propiedades de la clase asignador.py.....	440
Figura 338. Código completo de la clase asignador.py.....	442

Figura 339. Ubicación de la clase vista.py.....	443
Figura 340. Métodos, atributos y propiedades de la clase vista.py.	443
Figura 341. Código completo de la clase vista.py	444
Figura 342. Ubicación del bloque controlador.py.	445
Figura 343. Métodos, atributos y propiedades de la clase controlador.py.....	445
Figura 344. Código completo de la clase controlador.py	446
Figura 345. Ubicación de la clase eventomouse.py.....	447
Figura 346. Métodos, atributos y propiedades de la clase eventomouse.py.....	447
Figura 347. Código completo de la clase eventomouse.py.....	449
Figura 348. Diagrama esquemático completo del circuito Trazador de curvas (a).	451
Figura 349. Diagrama a bloques del programa completo de Arduino para adquirir señales del circuito Trazador de Curvas (a).....	452
Figura 350. Programa completo de Arduino.....	459
Figura 351. Diagrama de clases completo para el circuito Trazador de curvas (a).	460
Figura 352. Ubicación de la clase calculadora_datos_graficas.py.	461
Figura 353. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.	461
Figura 354. Código completo de la clase calculador_datos_graficas.py	472
Figura 355. Ubicación de la clase graficador.py.....	473
Figura 356. Métodos y atributos de la clase graficador.py.....	473
Figura 357. Código completo de la clase graficador.py	485
Figura 358'. Diagrama esquemático completo para el circuito generador de señal senoidal (b).....	486
Figura 359. Diagrama a bloques del programa completo para adquirir señales del circuito Generador de Señal Senoidal (b).	487
Figura 360. Programa completo de Arduino.....	492

Figura 361. Diagrama de clases para el circuito generador de señal senoidal (b).....	493
Figura 362. Ubicación de la clase graficador.py.....	494
Figura 363. Métodos y atributos de la clase graficador.py.....	494
Figura 364. Código completo de la clase graficador.py	504
Figura 365. Diagrama esquemático completo del circuito Amplificador de una etapa (c).	505
Figura 366. Diagrama a bloques del programa completo en Arduino para adquirir señales del circuito Amplificador de una Etapa (c)	506
Figura 367. Programa completo de Arduino.....	514
Figura 368. Diagrama de clases para el circuito Amplificador de una etapa (c).	515
Figura 369. Ubicación de la clase calculadora_datos_gráficas.py.	516
Figura 370. Métodos y atributos de la clase calculadora_datos_graficas.py.....	517
Figura 371. Código completo de la clase calculadora_datos_graficas.py	535
Figura 372. Ubicación de la clase graficador.py.....	536
Figura 373. Métodos y atributos de la clase graficador.py.....	536
Figura 374. Código completo de la clase graficador.py	551
Figura 375. Diagrama esquemático completo del circuito secuencial (d) implementado por hardware (i).....	553
Figura 376. Diagrama esquemático completo del circuito secuencial (d) implementado por software (ii).	553
Figura 377. Diagrama a bloques del programa completo de Arduino para la implementación del Circuito Secuencial (4).....	554
Figura 378. Programa completo de Arduino.....	563
Figura 379. Diagrama a bloques del programa completo en Arduino para adquirir señales del Circuito secuencial (d).	564
Figura 380. Programa completo de Arduino.....	571
Figura 381. Diagrama de clases para el circuito secuencial (d).....	572

Figura 382. Ubicación de la clase fsm.py	573
Figura 383. Métodos, atributos y propiedades de la clase fsm.py.....	573
Figura 384. Código completo de la clase fsm.py.....	576
Figura 385. Ubicación de la clase calculadora_datos_gráficas.py.	577
Figura 386. Métodos y atributos de la clase calculadora_datos_gráficas.py.....	577
Figura 387. Código completo de la clase calculadora_datos_graficas.py	588
Figura 388. Ubicación de la clase graficador.py.....	588
Figura 389. Métodos y atributos de la clase graficador.py.....	588
Figura 390. Código completo de la clase graficador.py	608

Índice de tablas

Tabla 1. Tabla de funciones de transición entre estados	72
Tabla 2. Asignación de números binarios a los estados de la máquina de estados finitos	73
Tabla 3. Tabla del funcionamiento del flip-flop tipo D 74LS74.....	73
Tabla 4. Tabla de funciones de transición entre estados incluyendo las entradas para los flip-flops tipo D a utilizar	74
Tabla 5. Comparación para la tabla “tabla_transiciones_estados”	91
Tabla 6. Comparación para la tabla “tabla_transiciones_estado_futuro”	92
Tabla 7. Comparación de las tablas “tabla_transiciones_estados” con “tabla_transiciones_estado_futuro”	92
Tabla 8. Comparación para la tabla “tabla_output_por_estado”	93
Tabla 9. Pines del microcontrolador encargados de leer los voltajes del circuito trazador de curvas	106
Tabla 10. Comparación de las funciones que realiza la parte del programa Medición de curvas cambiando el voltaje Vdd (2) con su diagrama a bloques.....	112
Tabla 11. Comparación de las funciones que realiza la parte del programa Medición de curvas cambiando el voltaje Vgg (3) con su diagrama a bloques.....	112
Tabla 12. Comparación de las funciones que realiza la parte del programa Lectura, procesamiento y envío de datos (4) con su diagrama a bloques.....	118
Tabla 13. Orden de envío de variables del circuito a la computadora a través del puerto serial.	123
Tabla 14. Voltaje generado a partir de un número binario.	131
Tabla 15. Comparación de las funciones que realiza la parte del programa Lectura, procesamiento y envío de datos (3) con su diagrama a bloques.....	132
Tabla 16. Comparación de la lectura de variables con el código.....	134
Tabla 17. Comparación de las funciones que realiza la parte del programa Medición de curvas cambiando el voltaje Vdd (2) con su diagrama a bloques.....	142

Tabla 18. Comparación de las funciones que realiza la parte del programa Medición de curvas cambiando el voltaje Vgg (3) con su diagrama a bloques.....	142
Tabla 19. Comparación de las funciones que realiza la parte del programa Medición de curvas en pequeña señal (4) con su diagrama a bloques.....	143
Tabla 20. Significado de los valores binarios de entrada del vector de entradas para probar la máquina de estados.....	153
Tabla 21. Descripción de los canales de muestras.....	178
Tabla 22. Descripción de los canales con filtro circular y pasa bajas.....	178
Tabla 23. Orden de los títulos en la variable muestras_llaves.....	179
Tabla 24. Orden de los índices de voltajes para la variable ind_voltajes.	180
Tabla 25. Orden de los índices de corrientes para la variable ind_corrientes.....	180
Tabla 26. Descripción de los canales de muestras.....	214
Tabla 27. Descripción de los canales con filtro circular y pasa bajas.....	215
Tabla 28. Orden de los títulos en la variable muestras_llaves.....	215
Tabla 29. Descripción de los canales de muestras.....	232
Tabla 30. Descripción de los canales con filtro circular y pasa bajas.....	232
Tabla 31. Orden de los títulos en la variable muestras_llaves.....	233
Tabla 32. Orden de los índices de voltajes para la variable ind_voltajes.	234
Tabla 33. Orden de los índices de corrientes para la variable ind_corrientes.....	234
Tabla 34. Descripción de los canales con filtro circular y pasa bajas.....	271
Tabla 35. Orden de los títulos en la variable muestras_llaves.....	272
Tabla 36. Resultados de los parámetros híbridos del transistor MOSFET 2N7000.	326
Tabla 37. Valores seleccionados de valor binario y voltaje a la salida.	331
Tabla 38. Resultados de los parámetros híbridos del transistor MOSFET 2N7000.	342
Tabla 39. Resultados de los parámetros en el punto de operación del transistor MOSFET 2N7000.	343

Tabla 40. Estados por los que pasa la máquina de estados finitos, mostrando entrada y salida correspondiente a cada estado.	348
Tabla 41. Tabla comparativa de las gráficas resultantes.	359
Tabla 42. Tabla comparativa de las gráficas de los parámetros gm y rd	360
Tabla 43. Valores seleccionados de valor binario y voltaje a la salida.	363
Tabla 44. Tabla comparativa de las gráficas resultantes.	365
Tabla 45. Comparación del voltaje Vgs con el voltaje Vds(sat) y la corriente Ids(sat) para las curvas Id vs Vds.....	366
Tabla 46. Tabla comparativa de las gráficas de los parámetros gm y rd	367
Tabla 47. Comparación de resultados	369
Tabla 48. Comparación de resultados para el amplificador en fuente común.....	370
Tabla 49. Comparación del funcionamiento de la máquina de estados finitos aplicándole el vector de prueba. (El estado inicial es 00).....	371
Tabla 50. Comparación de las gráficas resultantes que contienen diagramas de estados.	373

1. Introducción

Hoy en día la evolución de nuevas tecnologías, como el desarrollo de herramientas de visión por computadora, nos han permitido realizar nuevos experimentos que antes no se creían posible, tal es el caso de la realidad aumentada.

La realidad aumentada es aquella que permite observar una parte del mundo real en un dispositivo tecnológico (computadora, celular, tablet, etc.), añadiendo sobre dicha imagen, información virtual generada por el dispositivo tecnológico. Proporciona ciertas características a una imagen o video capturado del mundo real, pues añade animaciones o parámetros (información) que sean de interés, para así poder visualizar el mundo real desde una perspectiva diferente.

Por lo general, incluye los siguientes componentes:

- **Cámara:** Capta la imagen del mundo real.
- **Procesador:** Combina la imagen del mundo real con la información a superponer.
- **Software:** Gestiona el proceso de superposición.
- **Pantalla:** Muestra la imagen con la información superpuesta.
- **Activador:** Elemento del mundo real que utiliza el software para reconocer el entorno físico y seleccionar la información virtual asociada que se debe añadir.
- **Marcador:** Encargado de reproducir las imágenes creadas por el procesador y la posición en dónde se verá el modelo en 3D.

Una de las aplicaciones de la realidad aumentada ha sido en el sector de la educación, ya que ha permitido incrementar la interacción y motivación de los estudiantes. Algunos ejemplos en donde se ha utilizado la realidad aumentada son:

- Laboratorios. Actividades asociadas a videos y tutoriales.
- Trabajos de campo de diferentes temáticas.
- Jornadas de puertas abiertas. La comunidad educativa a través de la utilización de códigos QR pueden obtener información adicional y relevante.
- Trabajos colaborativos facilitando el trabajo en equipo.

En ingeniería, el análisis de circuitos electrónicos muchas veces llega a ser complicado, ya que se debe de tener una gran imaginación en los fenómenos y efectos que están presentes en el funcionamiento del circuito, así como el efecto que produce el cambiar ciertos parámetros del mismo (ej. voltajes nodales, valores de resistencia, capacitancia, etc.).

El sistema de realidad aumentada propuesto permite visualizar los principales parámetros y funcionamiento del circuito en tiempo real, mostrándolos sobre la misma imagen de éste. Así el estudio de los circuitos electrónicos resulta mucho más fácil, ya que se pueden observar los diferentes efectos que se tienen en el mismo, variando sus diferentes parámetros eléctricos, haciendo posible, de esta manera, conocer el comportamiento del circuito bajo prueba de una manera rápida y muy completa.

2. Antecedentes

Existen diversos trabajos de realidad aumentada aplicada al aprendizaje de circuitos eléctricos y electrónicos. A continuación, se mencionan los títulos de cuatro trabajos, se da una breve explicación de los mismos y se comentan las coincidencias y diferencias que presentan con relación al proyecto propuesto:

1. Los trabajos “**Using augmented reality to teach fifth grade students about electrical circuits**” [1] y “**Augmented reality to improve STEM motivation**” [2] mencionan una manera en la que se puede utilizar realidad aumentada para enseñar a niños en escuelas primarias. En estos trabajos se muestra la realidad aumentada a través de un dispositivo electrónico, como una Tablet o un celular, utilizando la cámara del dispositivo. Tanto [1] como [2] afirman que es un estímulo positivo para el aprendizaje, proporcionando una “*gran satisfacción y revelando una buena percepción de los estudiantes sobre estas perspectivas de aprendizaje*”. [2] Indica que tiene un gran potencial educacional.

Ambos trabajos coinciden con el proyecto que se propone en la utilización de un enfoque didáctico. En particular [1] se dedica a la enseñanza de circuitos eléctricos, y [2] a los campos de Ciencia, Tecnología, Ingeniería y Matemáticas (STEM). La diferencia principal con el proyecto propuesto radica en el nivel universitario y el enfoque de análisis y diseño electrónico que se abordará en el mismo.

2. En el trabajo “**Evaluating an online augmented reality puzzle for DC circuits: Students' feedback and conceptual knowledge gain**” [3] se desarrolló la App “*AR DC Circuit Design*”, donde su base fundamental es el modelado en 3D de componentes a partir de tarjetas de papel con símbolos. Permite observar algunos conceptos de corriente directa en diferentes topologías de circuitos sencillos y la simulación de circuitos eléctricos básicos.

La diferencia con el proyecto propuesto radica en que no se utilizará el modelado en 3D de los componentes, ni tarjetas con símbolos y no se hará uso de la simulación de los circuitos. En el proyecto que se desarrollará, se utilizarán marcadores para obtener información de los voltajes nodales y, con ello, generar datos relacionados con el comportamiento y funcionamiento del circuito bajo prueba, para posteriormente mostrarlos en diferentes formatos sobre la imagen del circuito que se analiza.

3. El proyecto “**Realidad aumentada móvil aplicada a mediciones eléctricas**” [4] es una propuesta donde se utiliza un circuito resistivo armado en una tablilla de pruebas (protoboard) y la cámara de un Smartphone, junto con una App desarrollada para mostrar información sobre la fotografía del circuito. Este proyecto realiza el procesamiento digital de imágenes y reconocimiento de patrones para la identificación de las bandas de colores de las resistencias del circuito eléctrico, calculando su valor nominal resistivo. El proyecto recopila también los valores de los voltajes nodales del circuito resistivo y calcula valores como corrientes, voltajes

y potencias en cada resistor. Finalmente, se despliegan sobre la fotografía del circuito original en el Smartphone, tanto los valores recopilados como los calculados. Afirma que es un sistema eficiente ya que presenta un resultado satisfactorio.

La principal diferencia del proyecto mencionado respecto al proyecto propuesto es que se analizarán circuitos electrónicos y no solamente circuitos resistivos, además de generar una gran cantidad de datos, pues no solo se prevé la visualización de parámetros eléctricos de voltaje, corriente y potencia como lo hace [4], sino también la visualización de los diferentes parámetros que conforman el circuito electrónico, así como su funcionamiento. Por ejemplo, en un trazador de curvas con MOSFET, se visualizarán los parámetros del transistor analizado, como: I_G (corriente de compuerta), V_{GS} (voltaje compuerta-fuente), i_D (corriente de drenaje), V_{DS} (voltaje drenaje-fuente), (λ), g_m (transconductancia), r_d (parámetro del modelo híbrido pi), por mencionar algunos.

Tanto el proyecto propuesto como las referencias mencionadas en [1], [2], [3] y [4] tienen el mismo fin, realizar un sistema que facilite y haga atractivo el estudio de circuitos eléctricos y electrónicos con la ayuda de realidad aumentada.

3. Justificación

La realidad aumentada puede utilizarse en el campo de los circuitos electrónicos para: (1) mostrar de manera simultánea gran cantidad de datos; (2) visualizar conceptos abstractos relacionados con los mismos y (3) facilitar la detección de fallas en los circuitos.

En este proyecto se pretende iniciar este camino, experimentando con circuitos electrónicos básicos, tanto analógicos como digitales, en los rubros mencionados anteriormente.

1. **Visualización de información completa y en tiempo real de un circuito electrónico.** Tanto en los laboratorios de docencia como en los de investigación, es necesario realizar mediciones (“una a la vez”), lo cual puede ser muy tardado y tedioso si el circuito que se analiza o diseña es complejo. Con este proyecto se planea mostrar la mayor cantidad de información del circuito bajo prueba de manera simultánea, utilizando la herramienta de la realidad aumentada. De esta manera, se tendrá la oportunidad de generar, tanto una vista global del funcionamiento de todo el circuito (“plano llave del circuito”), como del funcionamiento de ciertas zonas del mismo (“planos locales de ciertas zonas del circuito”).
2. **Visualización de parámetros abstractos en un circuito electrónico.** Con el auxilio de la realidad aumentada, se pretende facilitar la explicación y “conexión” de los conceptos abstractos, mostrando una representación visual de éstos sobre la imagen misma de los circuitos que se estudien.
3. **Posibilidad de detectar errores más fácil y rápidamente.** Una vez teniendo información muy completa del circuito, se puede determinar si funciona correctamente y si su funcionamiento es el deseado. Esto ayudará tanto en el diseño de circuitos electrónicos como en la detección de fallas, o en la identificación de componentes defectuosos.

Estos tres puntos explican la contribución del proyecto propuesto al campo de la realidad aumentada aplicada a los circuitos electrónicos.

4. Objetivos

Objetivo General:

- **Diseñar y construir** un conjunto de prototipos electrónicos para el análisis y diseño de circuitos electrónicos, utilizando un enfoque didáctico con realidad aumentada.

Objetivos Particulares:

- **Diseñar** una etapa de adquisición de datos para su implementación con un microcontrolador, para así, obtener las distintas señales eléctricas de interés en un circuito electrónico a analizar.
- **Validar** el funcionamiento de un circuito electrónico (con transistores MOSFET o un circuito digital secuencial) mediante el análisis de los parámetros adquiridos de la etapa de adquisición de datos.
- **Asociar** los diferentes parámetros eléctricos adquiridos del circuito (ej. voltajes nodales, corrientes, curvas características) con su imagen, para su visualización con realidad aumentada.

5. Marco teórico

5.1 Cálculo de parámetros de un transistor MOSFET canal n de enriquecimiento

5.1.1 Definición de voltajes y corrientes sobre un transistor MOSFET

El símbolo de un transistor MOSFET canal n de tipo enriquecimiento es el siguiente:

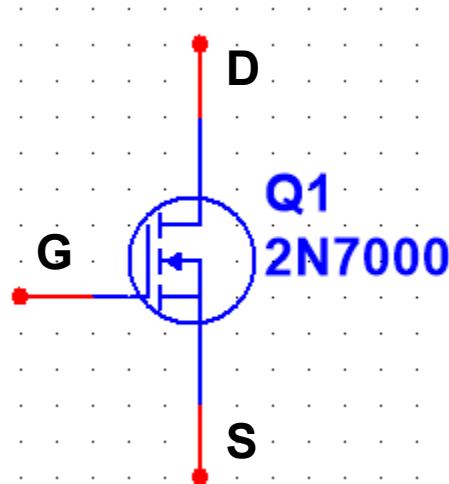


Figura 1. Símbolo de un transistor MOSFET canal n de enriquecimiento.

En donde:

- La terminal D es *drain* o drenaje.
- La terminal G es *gate* o compuerta.
- La terminal S es *source* o fuente.

Para este transistor se definen los siguientes voltajes y corrientes.

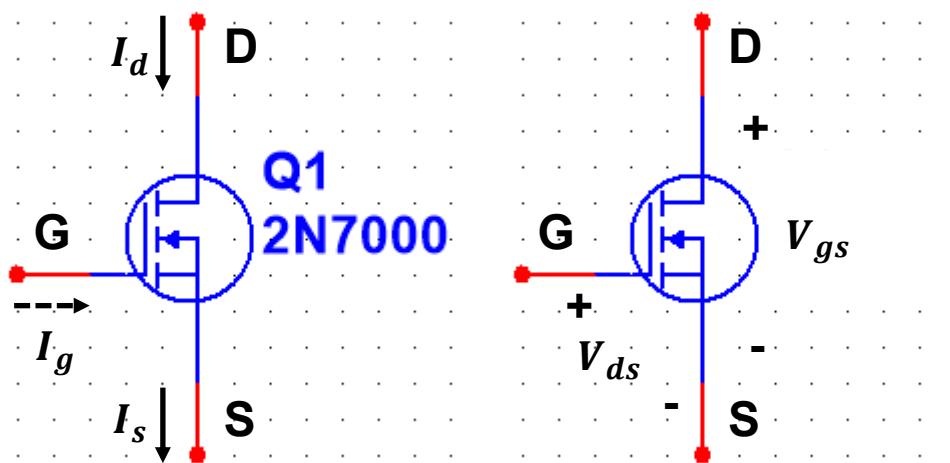


Figura 2 Convenciones de voltaje y corriente para un transistor MOSFET de enriquecimiento.

Se definen los voltajes:

- V_{gs} entre las terminales *gate* y *source*.
- V_{ds} entre las terminales *drain* y *source*.

Se definen las corrientes:

- I_d que es la corriente que pasa por la terminal de *drain*
- I_g que es la corriente que pasa por la terminal de *gate*
- I_s que es la corriente que pasa por la terminal de *source*

5.1.2 Modelo híbrido pi de un transistor MOSFET

El modelo híbrido pi para un transistor MOSFET es el siguiente:

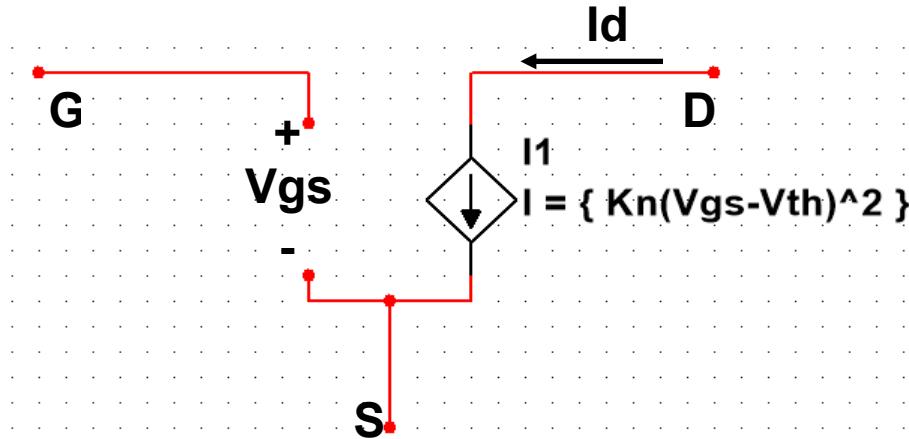


Figura 3. Modelo híbrido PI de un transistor MOSFET canal n de enriquecimiento.

Un transistor MOSFET funciona como una fuente de corriente controlada por voltaje. Se define como circuito abierto entre *gate* y *source* porque en la construcción de un transistor MOSFET, en *gate* hay una placa de metal que está aislada del semiconductor, impidiendo el paso de corriente ($I_g = 0$).

La ganancia del transistor está dada por la siguiente ecuación:

$$I_d = K_n(V_{gs} - V_{th})^2 \quad \forall V_{gs} \geq V_{th}$$

Esta ecuación es válida para valores de V_{gs} mayores al voltaje de umbral V_{th} . Esta ecuación está simplificada, ya que no considera los efectos que produce V_{ds} sobre la corriente I_d . El parámetro K_n es una constante de conducción del transistor MOSFET.

5.1.3 Curvas características del transistor MOSFET

5.1.3.1 Curva característica de entrada

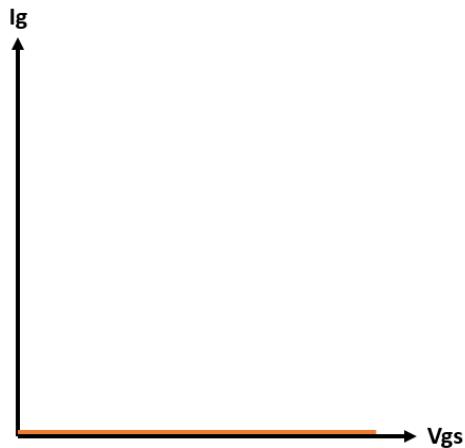


Figura 4. Curva característica de entrada para un transistor MOSFET canal n de enriquecimiento.

En esta curva característica de entrada se muestra la dependencia de la corriente de *gate* I_g (parámetro de entrada) y con el voltaje entre *gate* y *source* V_{gs} (parámetro de entrada), que son los parámetros calculados a la entrada del transistor. En el diagrama del modelo híbrido pi del transistor MOSFET se define como circuito abierto entre las terminas de *gate* y *drain*. Por lo tanto, la corriente de *gate* I_g es 0A.

5.1.3.2 Curva característica de transferencia

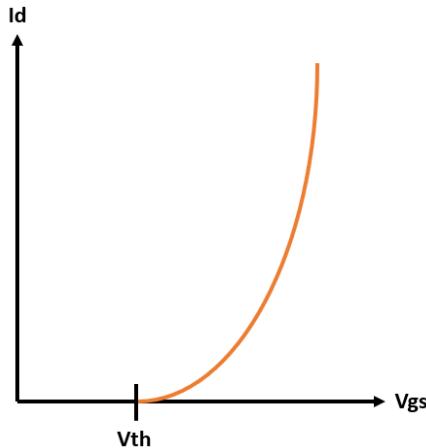


Figura 5. Curva característica de transferencia para un transistor MOSFET canal n de enriquecimiento.

En esta curva característica de transferencia se muestra la dependencia de la corriente de drain I_d (parámetros de salida) con el voltaje entre gate y source V_{gs} (parámetro de entrada). La ecuación de esta curva es la de una parábola con vértice en V_{th} :

$$I_d = K_n(V_{gs} - V_{th})^2 \quad \forall V_{gs} \geq V_{th}$$

5.1.3.3 Familia de curvas características de salida

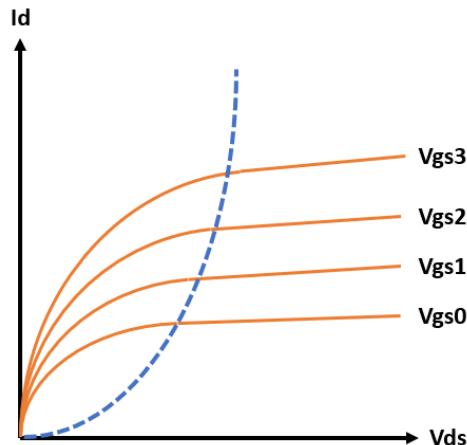


Figura 6. Familia de curvas características de salida para un transistor MOSFET canal n de enriquecimiento.

En esta familia de curvas características de salida, se compara la corriente de *drain* I_d (parámetro de salida) y el voltaje entre *drain* y *source* V_{ds} (parámetro de salida). Hay varias curvas porque se utilizan diferentes valores de voltaje V_{gs} .

5.1.4 Regiones de operación

De la familia de curvas características de salida, se pueden identificar las siguientes regiones de operación:

- Óhmica, triodo o no saturación:
- Transición, umbral
- Saturación
- Corte

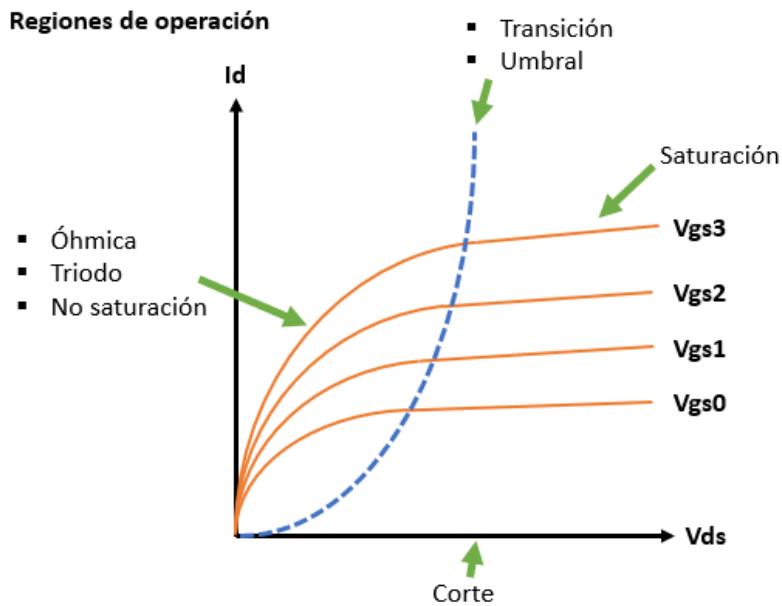


Figura 7. Regiones de operación para un transistor MOSFET canal n de enriquecimiento

Se observa que la línea discontinua delimita estas regiones. Mientras el transistor MOSFET opere en la región de no saturación, la familia de curvas características de salida se comporta como parábolas. Cuando el transistor MOSFET opera en la región de saturación, se comporta más linealmente.

5.1.5 Cálculo del parámetro lambda

Se puede calcular el parámetro lambda a partir de la gráfica de la familia de curvas características de salida.

Al prolongar la parte lineal de cada una de las curvas hacia el cuadrante 2, estas líneas intersecan al eje x de voltaje V_{ds} . Este punto es el recíproco del valor del parámetro lambda.

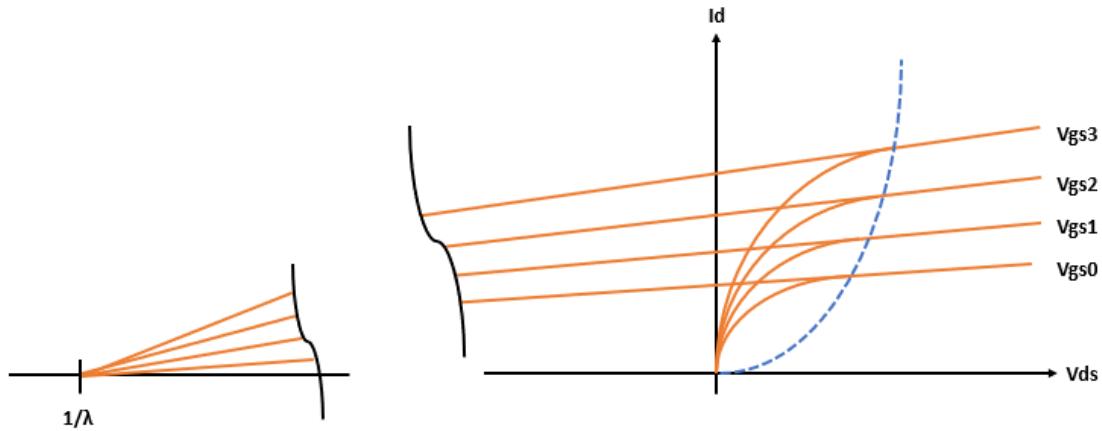


Figura 8. Gráfica que muestra el cálculo del parámetro lambda sobre la familia de curvas características de salida.

5.1.6 Cálculo de los parámetros Kn y Vth

Para calcular el voltaje de umbral y K_n , se propone el siguiente método. De la curva característica de transferencia, se eligen dos voltajes V_{gs1} y V_{gs2} sobre la curva de transferencia.

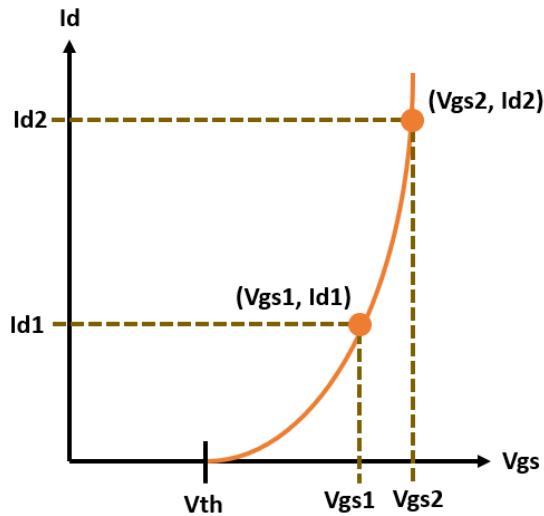


Figura 9. Medición del parámetro Kn y Vth sobre la curva característica de transferencia.

Estos voltajes V_{gs1} y V_{gs2} tienen una corriente I_{d1} e I_{d2} asociada. Utilizando la ecuación de esta curva característica de transferencia que es $I_d = K_n(V_{gs} - V_{th})^2 \forall V_{gs} \geq V_{th}$:

$$I_{d1} = K_n(V_{gs1} - V_{th})^2$$

$$I_{d2} = K_n(V_{gs2} - V_{th})^2$$

Despejando para K_n de la ecuación para V_{gs2} :

$$K_n = \frac{I_{d2}}{(V_{gs2} - V_{th})^2}$$

Sustituyendo K_n para V_{gs1} :

$$I_{d1} = \frac{I_{d2}}{(V_{gs2} - V_{th})^2} (V_{gs1} - V_{th})^2$$

Despejando V_{th} :

$$\sqrt{\frac{I_{d1}}{I_{d2}}} = \frac{V_{gs1} - V_{th}}{V_{gs2} - V_{th}}$$

$$(V_{gs2} - V_{th}) \sqrt{\frac{I_{d1}}{I_{d2}}} = V_{gs1} - V_{th}$$

$$V_{gs2} \sqrt{\frac{I_{d1}}{I_{d2}}} - V_{th} \sqrt{\frac{I_{d1}}{I_{d2}}} = V_{gs1} - V_{th}$$

$$V_{th} \left(1 - \sqrt{\frac{I_{d1}}{I_{d2}}} \right) = V_{gs1} - V_{gs2} \sqrt{\frac{I_{d1}}{I_{d2}}}$$

$$V_{th} \left(1 - \sqrt{\frac{I_{d1}}{I_{d2}}} \right) = V_{gs1} - V_{gs2} \sqrt{\frac{I_{d1}}{I_{d2}}}$$

Finalmente, la ecuación para calcular V_{th} es la siguiente:

$$V_{th} = \frac{V_{gs1} - V_{gs2} \sqrt{\frac{I_{d1}}{I_{d2}}}}{1 - \sqrt{\frac{I_{d1}}{I_{d2}}}}$$

Una vez calculado el voltaje V_{th} , se calcula K_n :

$$K_n = \frac{I_{d1}}{(V_{gs1} - V_{th})^2}$$

5.1.7 Cálculo del parámetro Idss

En las hojas de datos de un transistor, por lo general se especifica el parámetro I_{dss} , en vez de los parámetros K_n y V_{th} . Este parámetro también se puede calcular de la siguiente manera:

$$I_d = K_n (V_{gs} - V_{th})^2$$

Factorizando el voltaje V_{th} :

$$I_d = K_n V_{th}^2 \left(\frac{V_{gs}}{V_{th}} - 1 \right)^2$$

El parámetro I_{dss} es el producto de K_n por el voltaje de umbral V_{th} cuadrado

$$I_{dss} = K_n V_{th}^2$$

5.1.8 Cálculo de las ecuaciones de las curvas características de salida

La ecuación que define el comportamiento de la región de no saturación es la siguiente:

$$I_d = K_n [2(V_{gs} - V_{th})V_{ds} - V_{ds}^2]$$

La ecuación que define el comportamiento de la región de saturación es la siguiente:

$$I_d = K_n (V_{gs} - V_{th})^2 (1 + \lambda V_{ds})$$

La ecuación de la curva de umbral que delimita las regiones de no saturación y saturación es la siguiente:

$$V_{ds(sat)} = V_{gs} - V_{th}$$

5.1.9 Caracterización de un transistor MOSFET 2n7000 a través de la obtención de sus parámetros y gráficas.

Para caracterizar al transistor MOSFET, se propone el siguiente circuito de polarización:

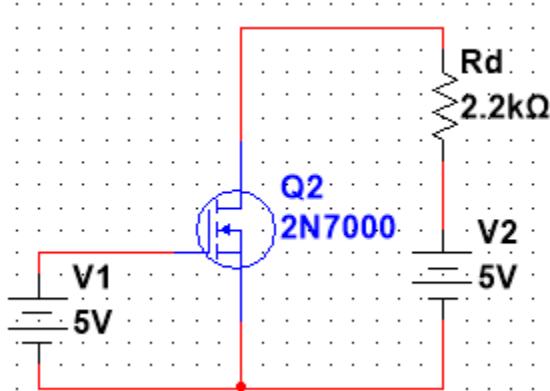


Figura 10. Circuito de polarización propuesto para la caracterización del transistor MOSFET canal n de enriquecimiento.

Para obtener los datos de las **gráficas de curva característica de entrada** y **curva característica de transferencia**, se propone que la fuente conectada V_{dd} a la terminal de drain sea constante a 5V, mientras se **barre el voltaje V_{gg}** de 0 a 5V.

Se medirá tanto el voltaje V_{gs} como la corriente I_g e I_d . Como se define en la figura 3 en el diagrama del modelo híbrido pi para un transistor MOSFET, se deduce que la corriente I_g es muy cercana a cero.

Con los valores de voltaje y corriente obtenidos de estas mediciones se pueden también calcular los parámetros V_{th} , K_n e I_{dss} .

Por otro lado, para obtener los datos de la **gráfica de familia de curvas características de salida**, se propone que para cada uno de los voltajes seleccionados de la fuente V_{gg} conectada a *gate*, se **barra el voltaje V_{dd}** . Por cada uno de los valores seleccionados de V_{gg} , se obtendrá una de las curvas.

Se medirá el voltaje V_{ds} y la corriente I_d a través de la diferencia de voltajes en la resistencia R_d .

Consultando la hoja de datos del transistor 2N7000 de Onsemi, se obtienen los siguientes valores:

El voltaje de umbral de encendido V_{th} se espera que esté entre 0.8 y 3V, siendo 2.1 su valor típico (cuando $V_{gs} = V_{ds}$ e $I_d = 1mA$). Se esperaría que el valor de voltaje $V_{ds(sat)}$ cuando el

transistor funciona en la región de saturación sea de 140 mV típico y 400mV máximo (cuando $V_{gs} = 4.5V$ e $I_d = 75mA$).

El parámetro de transconductancia g_m se espera que sea de 320 mS típico y 100 mS mínimo (cuando $V_{ds} = 10V$ e $I_d = 200mA$)

Por otra parte, los parámetros K_n ni I_{dss} no vienen indicados en las hojas de datos para este transistor.

Suponiendo $K_n = 6 \frac{mA}{V^2}$ y $V_{th} = 1.8V$, el valor de I_{dss} es:

$$I_{dss} = 6 \frac{mA}{V^2} * (1.8V)^2$$

$$\mathbf{I_{dss} = 19.44\ mA}$$

5.2 Circuito generador de señal senoidal

5.2.1 Arquitectura de un circuito DDS

La **técnica DDS (Direct Digital Synthesis)** se utiliza para producir formas de onda analógicas. Genera una señal que varía en el tiempo de forma digital y después realiza una conversión digital-analógico.

Puesto que la forma de onda es generada digitalmente en un principio, se puede programar diferentes formas de onda, como la triangular, cuadrada y senoidal.

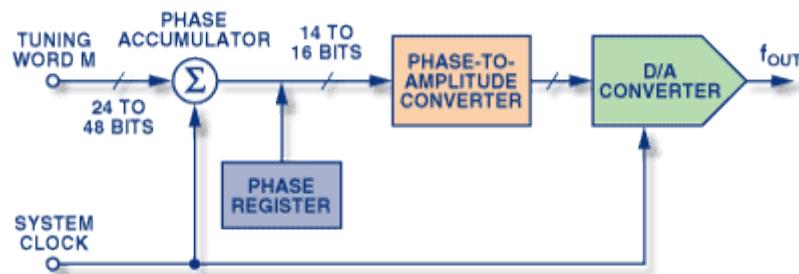


Figura 11. Diagrama a bloques de un circuito DDS genérico. (Fuente: Analog Devices)

En general, un circuito DDS se puede representar en tres bloques, como se muestra en la figura 12:

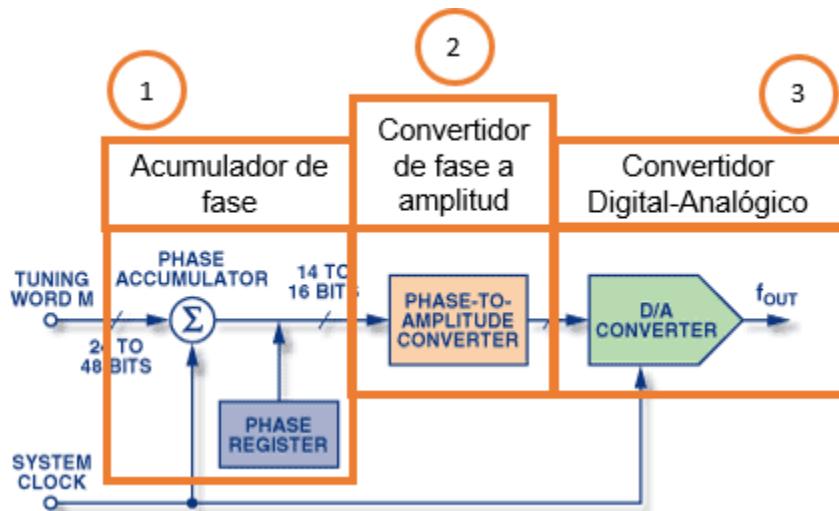


Figura 12. División del diagrama a bloques en tres partes diferentes (Fuente: Analog Devices).

1. Un **acumulador de fase (1)** (Phase Accumulator): Calcula un conjunto de ángulos entre 0 a 2π , en saltos discretos de igual magnitud determinados por la palabra de ajuste M.
2. Un **convertidor de fase a amplitud (2)** (Phase-to-amplitude converter): Convierte este ángulo en un número digital en binario, de acuerdo con una tabla que relaciona ángulo con amplitud.
3. Un **convertidor digital analógico (3)** (D/A converter): Convierte el número digital en binario a un voltaje o corriente analógico.

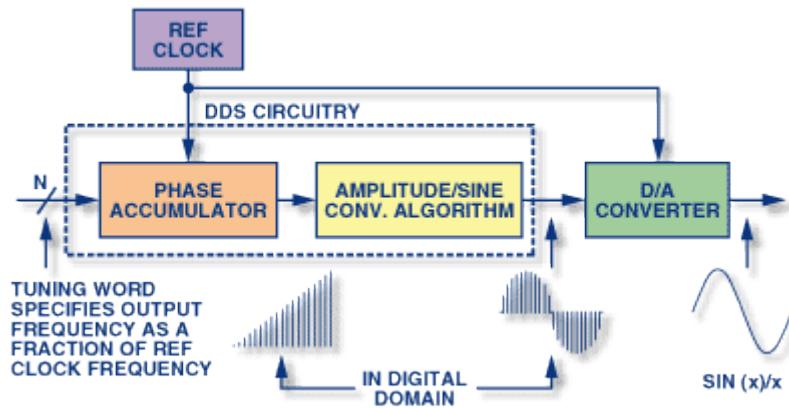


Figura 13. Flujo de una señal a través de un circuito DDS. (Fuente: Analog Devices)

En la figura 13 se observa el flujo de la generación de una señal senoidal a través de un circuito DDS. Para generar esta señal senoidal, se especifica una palabra de ajuste digital de N bits y una frecuencia de referencia (**system clock**). El **acumulador de fase (1)** en naranja convierte esta palabra en una secuencia de valores discretos de ángulos con igual separación. El **convertidor de fase a amplitud (2)** en amarillo convierte este ángulo en valores digitales en binario que corresponden a la forma de onda de una señal senoidal. Por último, el **convertidor digital analógico (3)** en verde convierte este valor digital en binario en un voltaje o corriente analógico.

5.2.1.1 Acumulador de fase (1) y convertidor fase-amplitud (2)

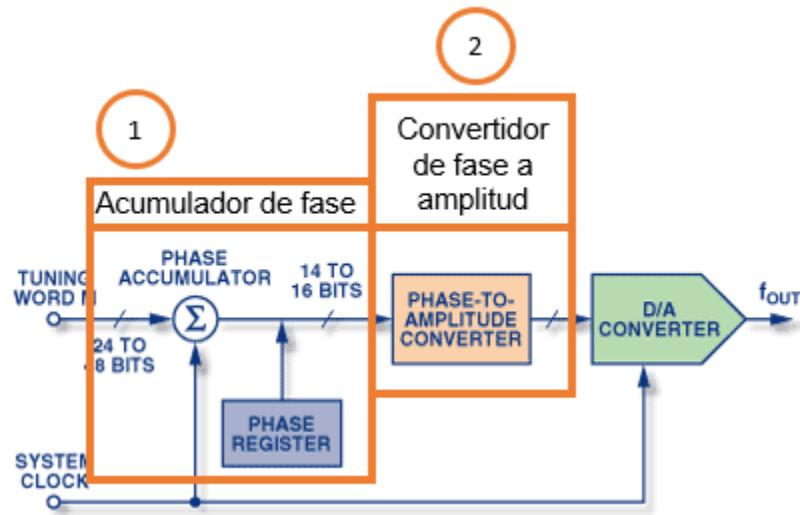


Figura 14. Ubicación de los bloques **Acumulador de fase (1)** y **Convertidor de fase a amplitud (2)**.

Un circuito DDS puede generar una señal senoidal a una determinada frecuencia de salida a partir de una frecuencia interna de referencia (**system clock**). Esta frecuencia interna de referencia (**system clock**) es la entrada principal del **acumulador de fase (1)**, el cual genera un ángulo dado el **contador módulo M**.

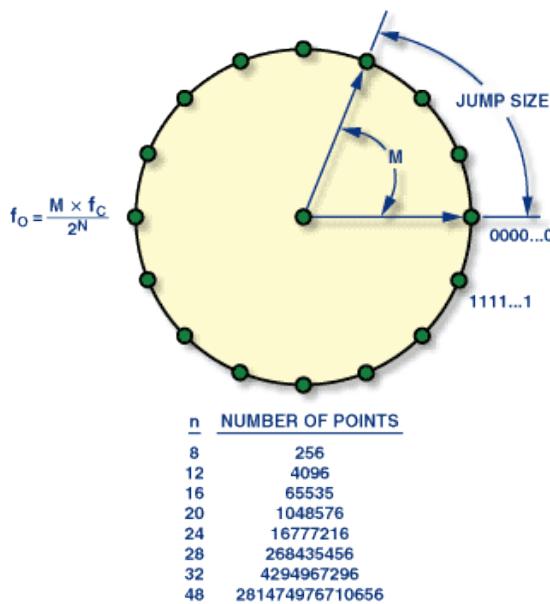


Figura 15. Rueda de fase digital. (Fuente: Analog Devices)

El **acumulador de fase (1)** tiene un **contador módulo M** que se incrementa con cada ciclo de reloj. Este **contador módulo M** determina el salto entre ángulos y determina cuántos puntos debe utilizar de la **rueda de fase digital** de la figura 15. El número total de puntos contenidos en esta rueda de fase está determinado por la **resolución del acumulador de fase (n)** y determina la resolución de ajuste del circuito DDS.

Este ángulo resultante del **acumulador de fase (1)** pasa al **convertidor fase-amplitud (2)**. En esta parte se busca este ángulo en una tabla que contiene valores digitales de amplitud para una determinada forma de onda (relaciona ángulo con valor digital de amplitud). El **convertidor fase-amplitud (2)** determina cuál es el valor digital de la amplitud de esa determinada forma de onda de acuerdo con esa tabla.

La relación entre la frecuencia de salida de la señal analógica, la frecuencia de referencia (**system clock**), el **contador módulo M** y la **resolución del acumulador de fase (n)** la da la siguiente ecuación:

$$f_{OUT} = \frac{M \times f_c}{2^n}$$

Donde

f_{OUT} es la frecuencia de salida del circuito DDS.

M es la palabra binaria de ajuste.

f_c es la frecuencia interna de referencia (o frecuencia del sistema).

n es la longitud del acumulador de fase, en bits.

Esta ecuación debe satisfacer el criterio de Nyquist. Esto es que debe de haber al menos dos muestras por ciclo o $f_{OUTMAX} \leq \frac{f_c}{2}$.

5.2.1.2 Convertidor digital-analógico

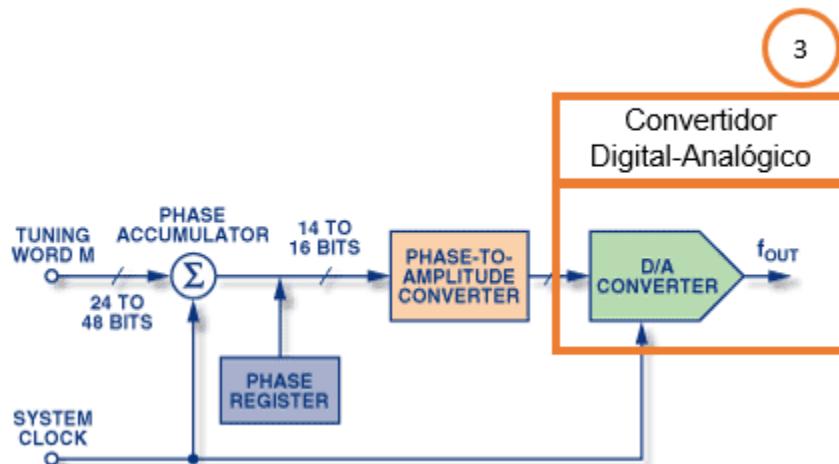


Figura 16. Ubicación del bloque Convertidor Digital-Analógico (3).

La tabla del **convertidor de fase a amplitud (2)** convierte instantáneamente la salida del acumulador de fase a un valor de amplitud digital. Este valor es presentado al **convertidor digital analógico (3)**. El valor digital de amplitud generado anteriormente es usado por el convertidor digital-analógico para generar una salida analógica, ya sea voltaje o corriente.

5.2.2 Propuesta para la construcción de un circuito DDS utilizando un microcontrolador y el circuito PCF8591

Para implementar el circuito DDS que genere una señal senoidal, se propone el siguiente diagrama a bloques:

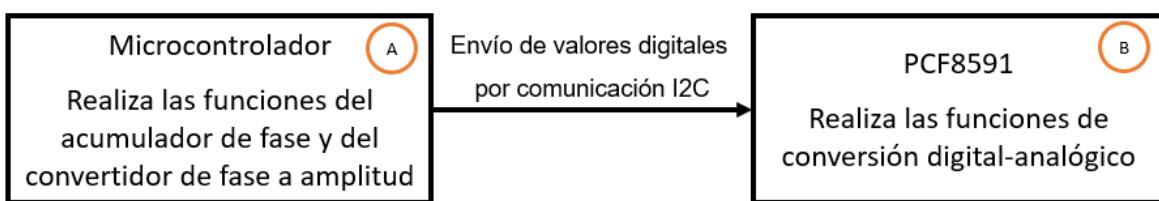


Figura 17. Diagrama a bloques del sistema propuesto para la construcción de un circuito DDS.

Este diagrama de la figura 17 se compone de dos partes:

- Microcontrolador (A):** Realiza las funciones del **acumulador de fase (1)** y del **convertidor de fase a amplitud (2)**. Envía el número digital en binario resultante de estas partes (correspondiente a la amplitud de la señal senoidal) a través de una interfaz para comunicación I2C.
- Circuito PCF8591 (B):** Realiza la función de **convertidor digital analógico (3)**. Recibe el número digital en binario enviado por el microcontrolador a través de una interfaz de comunicación I2C y realiza la conversión digital a analógico.

5.2.2.1 Acumulador de fase (1) y convertidor fase-amplitud (2)

Por software se puede implementar el acumulador de fase y el conversor fase-amplitud.

Con una frecuencia base de 10 Hz, el siguiente fragmento de código en c se ejecuta recurrentemente:

```

for (int i_muestra=0; i_muestra<250; i_muestra+=5)
    sine = byte(127.5*(1+sin((1000/sampleTime)*TWO_PI*i_muestra/(numMuestras-1)));

```

Figura 18. Fragmento de código en c para la ejecución del circuito DDS (**acumulador de fase (1)** y **convertidor de fase a amplitud (2)**)

De la figura 18 se observa que:

El **contador módulo M** viene siendo la variable `i_muestra`. Se toman 50 muestras para cada ciclo de la señal.

El tamaño de una variable byte es de 256, que puede ser representado por 8 bits y que viene siendo la longitud de la **resolución del acumulador de fase n**.

Si la frecuencia de referencia interna (**system clock**) f_c es 10 Hz, la frecuencia de salida es:

$$f_{out} = \frac{50 \times 10}{2^8} = 1.95\text{Hz}$$

La función utilizada en el fragmento de código de la figura 18 da como resultado el valor digital en binario de la amplitud. Puesto que se realiza la conversión a un tipo de dato byte, hay 256 valores posibles para la generación de la forma de onda de una señal senoidal.

5.2.2.2 Convertidor digital-analógico (3)

El circuito PCF8591 es un convertidor D/A y A/D. Contiene 4 entradas analógicas, una salida analógica y una interfaz para comunicación I2C.

Para el circuito DDS se utiliza su única salida analógica AOUT. El convertidor D/A interno es de 256 bits. La velocidad máxima del convertidor está determinada por la velocidad de la comunicación I2C.

5.3 Análisis de un circuito amplificador en fuente común con un transistor MOSFET 2N7000

5.3.1 Cálculo del punto de operación

Utilizando circuito de polarización de la figura 19 se puede realizar un análisis en pequeña señal cuando el transistor funciona como amplificador en fuente común.

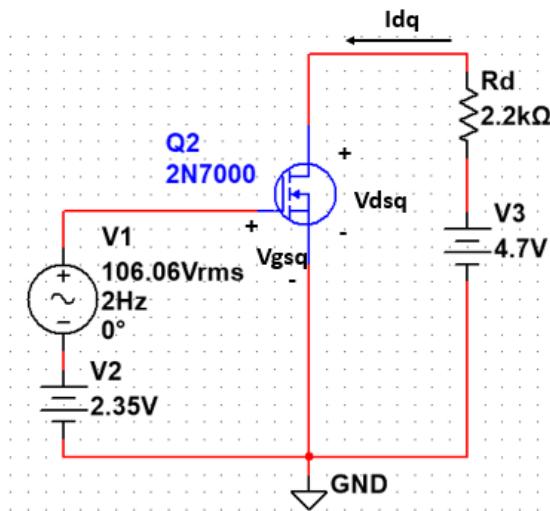


Figura 19. Circuito de polarización propuesto.

Determinando el punto de operación del transistor MOSFET.

Analizando la red de entrada, se propone que $V_{gsq} = 2.18\text{V}$

Calculando la corriente I_{dq} :

$$I_{dq} = K_n(V_{gsq} - V_{th})^2$$

Suponiendo $K_n = 6 \frac{\text{mA}}{\text{V}^2}$ y $V_{th} = 1.8\text{V}$:

$$I_d = 6 \frac{\text{mA}}{\text{V}^2} (2.18\text{V} - 1.8\text{V})^2$$

$$I_d = 830.983 \mu\text{A}$$

Analizando la red de salida del transistor MOSFET, suponiendo que la fuente de voltaje es $V_{dd} = 4.7V$:

$$V_{dd} - R_d I_{dq} - V_{ds} = 0$$

$$V_{ds} = V_{dd} - R_d I_{dq}$$

$$V_{dsq} = 4.7V - 2.2K\Omega * 830.983 \mu A$$

$$V_{dsq} = 2.87 V$$

Comprobando que el transistor trabaja en la región de saturación al compararlo con $V_{ds(sat)}$:

$$V_{ds(sat)} = V_{gsq} - V_{th}$$

$$V_{ds(sat)} = 2.18V - 1.8V = 0.38V$$

Se comprueba que:

$$V_{dsq} = 2.87 V > V_{ds(sat)} = 0.38 V$$

Por lo tanto, el transistor trabaja en la región de saturación.

Calculando la ecuación de la recta de carga estática:

$$i_d = -\frac{1}{R_d} V_{ds} + \frac{V_{dd}}{R_d}$$

$$I_d = -\frac{1}{2200\Omega} V_{ds} + \frac{4.7V}{2200\Omega}$$

La gráfica de la curva de salida sobre la que se ubica el punto de operación se muestra en la figura 20:

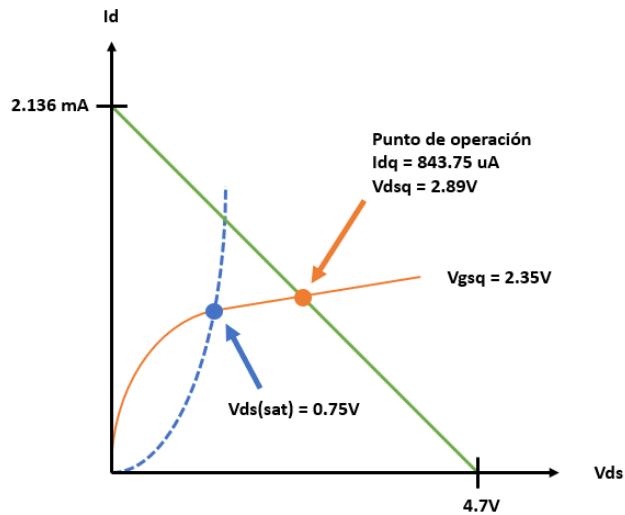


Figura 20. Punto de operación y recta de carga estática sobre la gráfica de curvas características de salida.

5.3.2 Criterios para utilizar pequeña señal

Para que la señal utilizada sea de pequeña señal, la componente de voltaje alterno v_{gs} debe cumplir:

$$I_{dq} + i_d = K_n(V_{gsq} + v_{gs} - V_{th})^2$$

$$I_{dq} + i_d = K_n([V_{gsq} - V_{th}] + v_{gs})^2$$

$$I_{dq} + i_d = K_n[V_{gsq} - V_{th}]^2 + 2K_n[V_{gsq} - V_{th}]v_{gs} + K_n v_{gsq}^2$$

Como $I_{dq} = K_n[V_{gsq} - V_{th}]^2$:

$$i_d = 2K_n[V_{gsq} - V_{th}]v_{gs} + K_n v_{gs}^2$$

Para minimizar los efectos de $K_n v_{gs}^2$ sobre la ecuación y que sea lineal se propone:

$$2K_n[V_{gsq} - V_{th}]v_{gs} \gg K_n v_{gs}^2$$

$$2[V_{gsq} - V_{th}] \gg v_{gs}$$

Transformando esta inecuación en una desigualdad:

$$v_{gs} \leq \frac{2[V_{gsq} - V_{th}]}{10}$$

Para este ejemplo, la amplitud máxima que puede tener la componente alterna v_{gs} es:

$$v_{gs} \leq \frac{2[2.18V - 1.8V]}{10} = 76 \text{ mV}$$

5.3.3 Cálculo de parámetros del modelo híbrido pi

5.3.3.1 Cálculo del parámetro gm del modelo híbrido pi

Obteniendo el parámetro híbrido g_m a partir de la curva característica de transferencia:

$$g_m = \left. \frac{\partial i_d}{\partial v_{gs}} \right|_Q = \left. \frac{\partial}{\partial v_{gs}} K_n (V_{gsq} - V_{th})^2 \right|_Q$$

Se obtiene la siguiente ecuación:

$$g_m = 2K_n(V_{gsq} - V_{th})$$

Para este ejemplo, el valor de g_m es:

$$g_m = 2 * 6 \frac{mA}{V^2} (2.18V - 1.8V)$$

$$\mathbf{g_m = 4.56 mS}$$

También se puede obtener con la siguiente fórmula, en términos de I_{dq} :

$$g_m = 2K_n(V_{gsq} - V_{th})$$

$$g_m = 2\sqrt{K_n(V_{gsq} - V_{th})K_n(V_{gsq} - V_{th})}$$

$$g_m = 2\sqrt{K_n [K_n(V_{gsq} - V_{th})^2]}$$

Se obtiene la siguiente ecuación:

$$g_m = 2\sqrt{K_n I_{dq}}$$

Para este ejemplo, se comprueba que valor de g_m es casi el mismo que el obtenido con la fórmula anterior:

$$g_m = 2\sqrt{6 \frac{mA}{V^2} * 830.983\mu A}$$

$$\mathbf{g_m = 4.46 mS}$$

5.3.3.2 Cálculo del parámetro r_g del modelo híbrido pi

Obteniendo el parámetro híbrido r_g a partir de la curva característica de entrada:

$$r_g = \frac{1}{\left. \frac{\partial i_g}{\partial v_{gs}} \right|_Q} \rightarrow \infty$$

Puesto que por la construcción del transistor MOSFET la corriente de *gate* I_g es 0A, la resistencia r_g tiene a infinito.

5.3.3.3 Cálculo del parámetro r_d del modelo híbrido pi

Obteniendo el parámetro híbrido r_d a partir de la curva característica de salida:

$$r_d = \left. \frac{1}{\frac{\partial i_d}{\partial v_{ds}}} \right|_Q$$

Resolviendo primero la derivada parcial:

$$\begin{aligned} \left. \frac{\partial i_d}{\partial v_{ds}} \right|_Q &= \frac{\partial}{\partial v_{ds}} \left(K_n (V_{gs} - V_{th})^2 (1 + \lambda V_{ds}) \right) \Big|_Q \\ \left. \frac{\partial i_d}{\partial v_{ds}} \right|_Q &= \lambda K_n (V_{gsq} - V_{th})^2 \\ \left. \frac{\partial i_d}{\partial v_{ds}} \right|_Q &= \lambda I_{dq} \end{aligned}$$

Se obtiene la siguiente ecuación:

$$r_d = \frac{1}{\lambda I_{dq}}$$

Suponiendo que $\lambda = 30 \frac{1}{mV}$, el valor de r_d es para este ejemplo:

$$r_d = \frac{1}{30 \frac{1}{mV} * 830.983 \mu A}$$

$$r_d = 40.113 k\Omega$$

5.3.4 Análisis en pequeña señal utilizando el modelo híbrido pi simplificado

El modelo híbrido pi del circuito de la figura 21 queda de la siguiente manera:

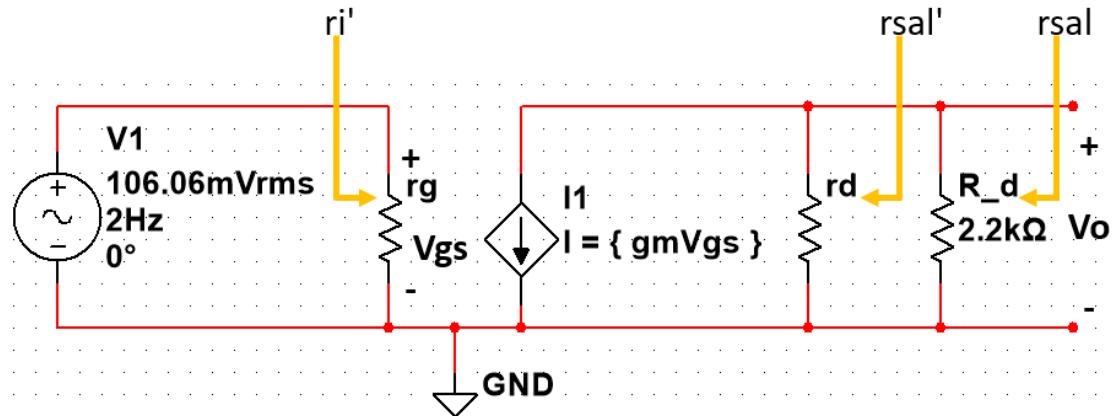


Figura 21. Modelo híbrido π simplificado del circuito propuesto de la figura 16.

En el circuito de la figura 21 no se consideran las capacitancias que aparecen entre *gate* y *drain* C_{gd} , entre *gate* y *source* C_{gs} ni entre *drain* y *source* C_{ds} .

Tampoco se considera si hay resistencias de carga o la resistencia interna de la fuente de alimentación V_1 .

Cabe aclarar que la señal senoidal de voltaje considerada como pequeña señal está montada sobre una componente de dc igual al voltaje $V_{gs} = 2.18V$

5.3.4.1 Cálculo de resistencias de entrada y salida

Este circuito tiene los siguientes parámetros híbridos:

$$g_m = 4.56mS$$

$$r_g \rightarrow \infty$$

$$r_d = 40.113k\Omega$$

Calculando las resistencias r'_i y r'_{sal} :

$$r'_i = r_g \rightarrow \infty$$

$$r'_{sal} = r_d = \mathbf{40.113k\Omega}$$

Para r_{sal} :

$$r_{sal} = r'_{sal} || R_d$$

$$r_{sal} = 168k\Omega || 2.2k\Omega$$

$$r_{sal} = \mathbf{2.08k\Omega}$$

5.3.4.2 Cálculo de la ganancia en voltaje Av

Para calcular la ganancia $A_v = \frac{V_o}{V_i}$, se parte de la ecuación para V_o en términos de V_1

$$V_o = -g_m V_{gs} r_{sal}$$

Como $V_i = V_{gs} = V_1$:

$$A_v = \frac{V_o}{V_i} = -\frac{g_m V_{gs} r_{sal}}{V_{gs}}$$

$$A_v = -g_m r_{sal} = -g_m (r'_{sal} || R_d)$$

$$A_v = -4.56mS * 2.08k\Omega$$

$$A_v = \mathbf{-9.48}$$

5.4 Diseño de un circuito secuencial

Para realizar la prueba del circuito secuencial se desarrolló una máquina de estados finitos.

5.4.1 Máquina de estados finitos

Una máquina de estados finitos es un modelo computacional que puede ser implementado tanto por hardware como por software y puede ser utilizado para simular lógica secuencial y algunos programas computacionales.

Estas máquinas de estado finito pueden utilizarse para modelar problemas en muchos ámbitos.

Para esta parte se desarrolló una máquina de estados finitos determinista, que significa que solo puede estar en un estado a la vez.

Una máquina estados finita, o un autómata determinístico finito, se define formalmente como una tupla de 5 elementos:

$$A = (Q, \Sigma, \delta, q_0, F)$$

Donde:

Q es un **conjunto finito de estados**.

Σ es un **alfabeto finito**, que son los símbolos de entrada.

δ es un **conjunto de funciones transición**.

F es un **conjunto de estados finales o aceptados**.

q_0 es el **estado inicial**.

5.4.2 Modelado del circuito secuencial como una máquina de estados finitos

Se propone una máquina de estados que resuelve el siguiente problema:

Se tiene una fuente de alimentación de voltaje y se quiere diseñar un circuito de control. Para ello debe cumplir con los siguientes requisitos:

- La fuente de alimentación suministra voltaje cuando está encendida.
- Cuando hay un cortocircuito, la salida de la fuente de alimentación se abre, es decir, deja de suministrar voltaje, aunque sigue encendida.
- La salida de la fuente no podrá volverse a activar mientras haya un cortocircuito.
- Cuando ya se quite el corto circuito, la salida de la fuente de alimentación se reactivará y suministrará voltaje.
- Cuando la fuente se apaga, deja de suministrar voltaje.

Con estos requisitos, se propone el siguiente diagrama de estados de la figura 22:

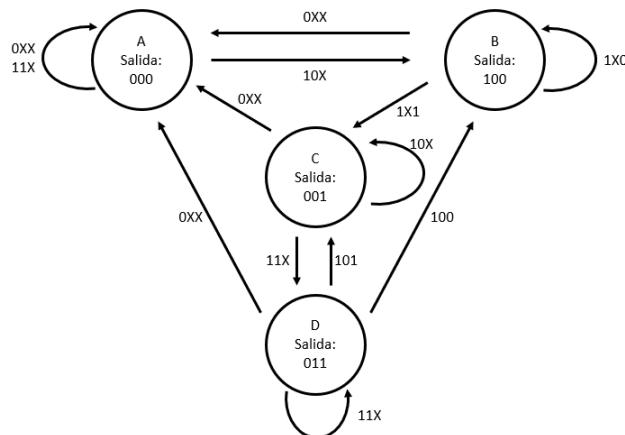


Figura 22. Diagrama de estados del problema propuesto.

En cada circulo se representa un estado diferente. Para cada uno de los 4 estados hay una letra que identifica al estado y se presenta su salida correspondiente de 3 bits. Cada flecha representa un cambio de estado y el número sobre la flecha indica cuál es la entrada con la que la máquina de estados cambia de estado.

Con base en el diagrama de estados presentado en la figura 22, se propone el siguiente circuito, en donde se definen las entradas y salidas de la máquina de estados finitos:

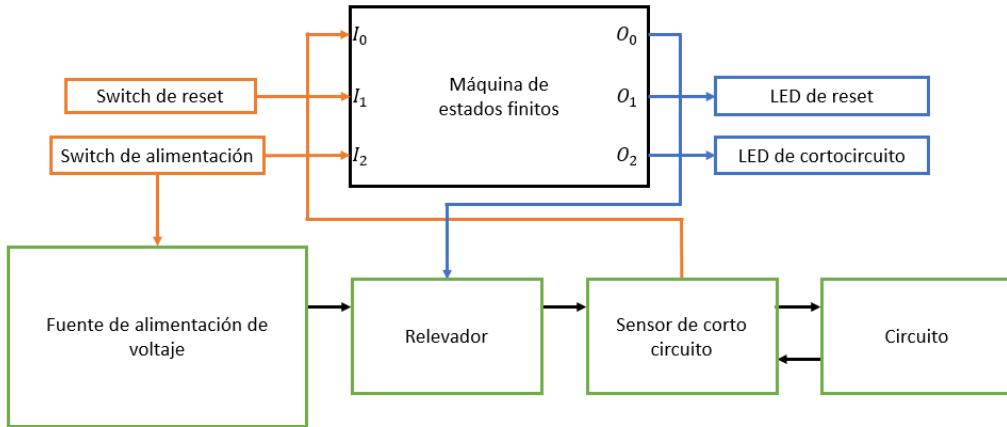


Figura 23. Diagrama a bloques con las salidas y entradas para la máquina de estados

Con base en el diagrama a bloques de la figura 23, se contemplan 3 entradas.

- **I_2 : Botón de encendido (Bit más significativo).** Un 1 lógico significa que la fuente de alimentación está encendida y un 0 lógico que está apagada.
- **I_1 : Botón de reset.** Un 1 lógico significa que hay un reset de la fuente de alimentación y un 0 lógico que no hay reset.
- **I_0 : Sensor de cortocircuito (Bit menos significativo).** Un 1 lógico significa que hay cortocircuito y un 0 lógico que no hay cortocircuito.

Se contemplan 3 salidas.

- **O_2 : Relevador de salida (Bit más significativo).** Cuando se cierra el relevador (1 lógico) hay salida de voltaje y cuando se abre el relevador (0 lógico) deja de haber salida de voltaje.
- **O_1 : LED de reset.** El LED se enciende cuando hay un 1 lógico y significa que el botón de reset está presionado, la fuente está encendida y hay un cortocircuito. Un 0 lógico significa que no está presionado.
- **O_0 : LED de cortocircuito (Bit menos significativo).** El LED se enciende cuando hay un 1 lógico y significa que hay un cortocircuito. Un 0 lógico significa que no hay cortocircuito.

Se contemplan 4 estados diferentes.

- **Estado A. Apagado.** La fuente de alimentación está apagada y por lo tanto no hay salida de voltaje.
- **Estado B. Encendido funcionando.** En este estado la fuente de alimentación está encendida y hay salida de voltaje. Esto quiere decir que no hay cortocircuitos.
- **Estado C. Cortocircuito.** La fuente está encendida pero el relevador se abre, dejando a la fuente sin salida de voltaje.
- **Estado D. Reset.** En este estado el botón de reset está presionado, mientras que el relevador de la fuente de alimentación está abierto y no deja que haya una salida de voltaje.

Formalmente, esta máquina de estados finitos se define de la siguiente manera:

Conjunto finito de estados $Q = \{A, B, C, D\}$

Alfabeto finito $\Sigma = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Estado inicial $q_0 = A$

Estado final $F = B$

La tabla 1 describe al **conjunto de funciones transición** δ :

Estado actual	Salida $O_2O_1O_0$	Entrada	Estado siguiente
		$I_2I_1I_0$	
A	000	000, 001, 010, 011, 110, 111	A
		100, 101	B
B	100	000, 001, 010, 011	A
		100, 110	B
		101, 111	C
C	001	000, 001, 010, 011	A
		100, 101	C
		110, 111	D
D	011	000, 001, 010, 011	A
		100	B
		101	C
		110, 111	D

Tabla 1. Tabla de funciones de transición entre estados

5.4.3 Diseño para la construcción de una máquina de estados finitos utilizando flip-flops tipo D

Para esta parte, se pasará del modelo de la máquina de estados finitos descrito en la sección anterior a las ecuaciones lógicas que la describen y que permiten su armado con flip-flops tipo D.

Para ello, se propone que los estados se representen como se muestra en la tabla 2:

Estado	$S_1 S_0$
A	00
B	01
C	10
D	11

Tabla 2. Asignación de números binarios a los estados de la máquina de estados finitos

Se proponen utilizar los flip flops tipo D 74LS74, los cuales tienen la siguiente tabla de verdad, mostrada en la tabla 3:

Entradas				Salidas	
PRE	CLR	D	CLK	Q	\bar{Q}
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	1	1
1	1	1	↑	1	0
1	1	0	↓	0	1
1	1	X	0	Q_0	\bar{Q}_0

Tabla 3. Tabla del funcionamiento del flip-flop tipo D 74LS74

Con estas definiciones, se modifica la tabla 1 para que se pueda representar esta máquina de estados con flip-flops:

Estado actual	Salida	Entrada	Estado siguiente	Flip-Flops
S_1S_0	$O_2O_1O_0$	$I_2I_1I_0$	Q_1Q_0	S_1S_0
00	000	0XX, 11X	00	00
		10X	01	01
01	100	0XX	00	00
		1X0	01	01
		1X1	10	10
10	001	0XX	00	00
		10X	10	10
		11X	11	11
11	011	0XX	00	00
		100	01	01
		101	10	10
		11X	11	11

Tabla 4. Tabla de funciones de transición entre estados incluyendo las entradas para los flip-flops tipo D a utilizar

De esta tabla 4 se obtienen las siguientes ecuaciones booleanas para los dos flip-flops Q_0 y Q_1 :

$$Q_0 = I_2[S_1I_1 + S_0\bar{I}_0 + \bar{S}_0\bar{S}_1\bar{I}_1]$$

$$Q_1 = I_2[S_1\bar{S}_0 + S_1I_1 + S_0I_0]$$

De esta tabla 4 se obtienen también las ecuaciones para las salidas O_0 , O_1 y O_2 :

$$O_0 = S_1$$

$$O_1 = S_0S_1$$

$$O_2 = \bar{S}_0S_1$$

6. Desarrollo del proyecto

6.1. Introducción

El proyecto de “Realidad Aumentada como una Herramienta para el Análisis y Diseño de Circuitos Electrónicos” es un proyecto que tiene como fin mostrar la información relevante de un circuito bajo prueba, de manera visual, utilizando realidad aumentada. Envía los datos recopilados del circuito a una computadora con una etapa de adquisición de datos, se procesan y se despliegan sobre una fotografía del circuito tomada anteriormente, con el fin de visualizar de manera sencilla la información relevante del circuito.

En la figura 24 se muestra el diagrama a bloques del proyecto. Como se observa, se tienen 3 bloques que conforman el sistema. Estos bloques tienen la función de realizar el análisis del circuito bajo prueba y poder visualizar los parámetros del circuito y su funcionamiento utilizando realidad aumentada. A continuación, se detalla cada bloque del sistema:

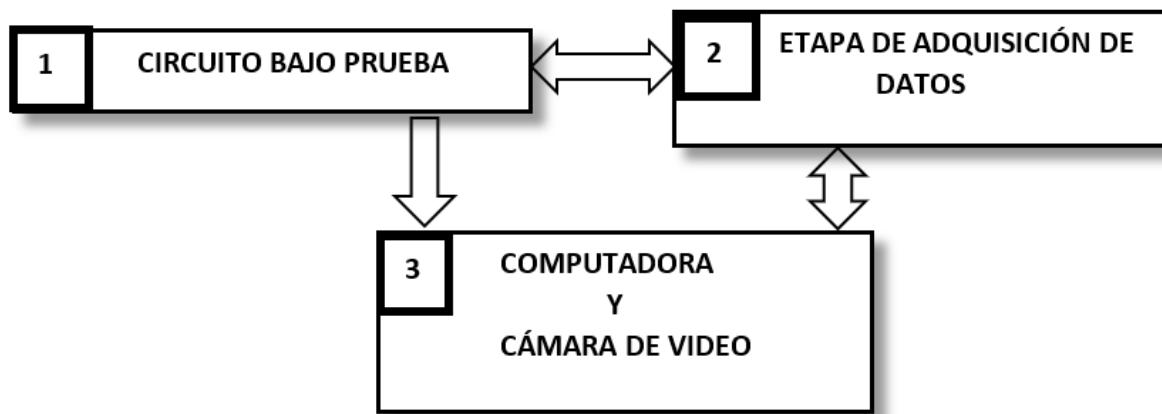


Figura 24. Diagrama a bloques del proyecto propuesto

En la figura 25 se muestra el diagrama detallado del proyecto propuesto con un ejemplo.

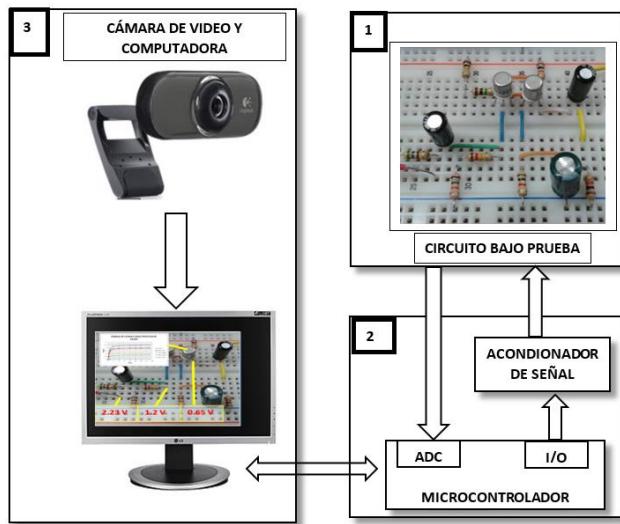


Figura 25. Diagrama del proyecto propuesto

1. Circuito Bajo Prueba. Este bloque contiene al circuito que se analiza. Se realizó el análisis sobre los siguientes circuitos:

- Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- Amplificador de una etapa (c)** con un transistor MOSFET.
- Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Cabe mencionar que el circuito de prueba debe ser montado sobre un protoboard, donde las conexiones de este circuito deben ser adecuadas, es decir, el circuito debe de ser estético.

2. Etapa de adquisición de datos. Se utiliza un microcontrolador y circuitos electrónicos para adquirir señales del **Circuito Bajo Prueba (1)**.

3. Computadora y cámara de video. La **Etapa de adquisición de datos (2)** envía las señales adquiridas para que sean procesadas por un programa de Python. Este programa realiza la comunicación con la **Etapa de adquisición de datos (2)**, procesa los datos, adquiere la imagen del circuito y despliega la información sobre el circuito.

Para los circuitos bajo prueba propuestos, se mostrarán las siguientes gráficas:

a. **Trazador de curvas (a).** Se presentarán 7 gráficas:

1. **Voltajes en el tiempo** usados para generar las diferentes señales del circuito.
2. **Gráfica I_g vs V_{gs} .** Curva característica de entrada del transistor MOSFET.
3. **Gráfica I_d vs V_{gs} .** Curva característica de transferencia del transistor MOSFET.
4. **Gráfica I_d vs V_{ds} .** Familia de curvas características de salida del transistor MOSFET.
5. **Gráfica g_m vs I_d .** Comportamiento de la transconductancia g_m en función de la corriente I_d con la que opera el transistor MOSFET.
6. **Gráfica r_d vs I_d .** Comportamiento de la resistencia r_d del modelo híbrido pi en función de la corriente I_d con la que opera el transistor MOSFET.
7. **Gráfica de parámetros híbridos.** Se presentan los valores de λ , K_n , voltaje de umbral V_{th} , resistencia híbrida r_g del transistor MOSFET.

b. **Circuito generador de señal senoidal digital (b).** Se presentarán 4 gráficas con 4 formas diferentes de mostrar la relación entre el número binario enviado al convertidor digital analógico y el voltaje a la salida de este convertidor.

1. **Comparación entre voltaje y binario (1)** en forma de una gráfica comparativa entre voltaje analógico de salida y número digital en binario que genera ese voltaje.
2. **Comparación entre voltaje y binario (2)** en forma de tabla comparativa entre voltaje analógico de salida y número digital en binario que genera ese voltaje.
3. **Comparación entre voltaje y binario (3)** mostrando los números binarios sobre la gráfica de voltaje analógico.
4. **Comparación entre voltaje y binario (4)** en forma de diagrama de estados.

c. **Amplificador de una etapa (c).** Se presentarán 8 gráficas:

1. **Voltajes en el tiempo** usados para generar las diferentes señales del circuito.
2. **Gráfica I_g vs V_{gs} .** Curva característica de entrada.
3. **Gráfica I_d vs V_{gs} .** Curva característica de transferencia.
4. **Gráfica I_d vs V_{ds} .** Familia de curvas características de salida.
5. **Gráfica g_m vs I_d .** Comportamiento de la transconductancia g_m del transistor.
6. **Gráfica r_d vs I_d .** Comportamiento de la resistencia r_d del modelo híbrido pi del transistor.
7. **Voltajes del amplificador en fuente común** en donde se observa el comportamiento del transistor como amplificador de pequeña señal.
8. **Gráfica de parámetros híbridos y punto de operación.** Se presentan los valores de λ , K_n , voltaje de umbral V_{th} , resistencia híbrida r_g del transistor. Adicionalmente, se presentarán parámetros del punto de operación, como V_{dsq} , I_{dq} , transconductancia g_{mq} , resistencias híbridas r_d y r_g , resistencia equivalente R_{lac} y ganancia.

d. **Círcuito secuencial (d).** Se presentarán 5 gráficas y son mostradas para las dos implementaciones de la máquina de estados finitos, tanto por hardware como por software.

1. **Señales digitales binarias en el tiempo** en donde se observan la entrada, estado y salida de la máquina de estados finitos.
2. **Tabla de estados binarios** en donde se muestran los diferentes estados de la máquina de estados, la salida correspondiente a cada estado y las transiciones hacia otros estados, según la sea la entrada que tenga la máquina de estados.
3. **Diagrama de estados (1)** de la máquina de estados finitos.
4. **Diagrama de estados (2)** de la máquina de estados finitos.
5. **Diagrama de transiciones de la máquina de estados finitos.**

6.2 Circuito Bajo Prueba (1)

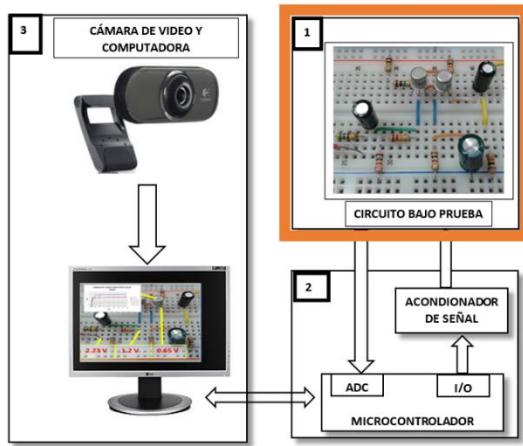


Figura 26. Ubicación del bloque del **Circuito Bajo Prueba (1)**.

El **Circuito Bajo Prueba (1)** es el circuito a estudiar. Se medirán varios parámetros y se desplegarán sobre la imagen del circuito. Para ello se proponen los siguientes circuitos:

- a. **Circuito Trazador de curvas.** En este circuito se caracteriza un transistor MOSFET canal n de enriquecimiento. Se utiliza tanto un microcontrolador como dos PCF8591 para variar diferentes voltajes del circuito.
- b. **Circuito Generador de Señal Senoidal.** En este circuito se construye y se prueba un circuito generador de senoidal utilizando un microcontrolador y un circuito PCF8591.
- c. **Circuito Amplificador de una Etapa.** Se utiliza el mismo circuito que el **Circuito Trazador de curvas (a)**, pero esta vez se utiliza el transistor MOSFET como un amplificador de una etapa en fuente común.
- d. **Circuito Secuencial.** Se modela una máquina de estados finitos para resolver un problema y se implementa de dos maneras diferentes
 - i. **Por hardware.** Se utilizan compuertas lógicas y flip flops para implementar el comportamiento de una máquina de estados.
 - ii. **Por software.** Se utiliza un microcontrolador programado en lenguaje c que implementa el comportamiento de esta máquina de estados.

6.2.1. Trazador de Curvas (a) y Amplificador de una Etapa (c)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Círculo secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

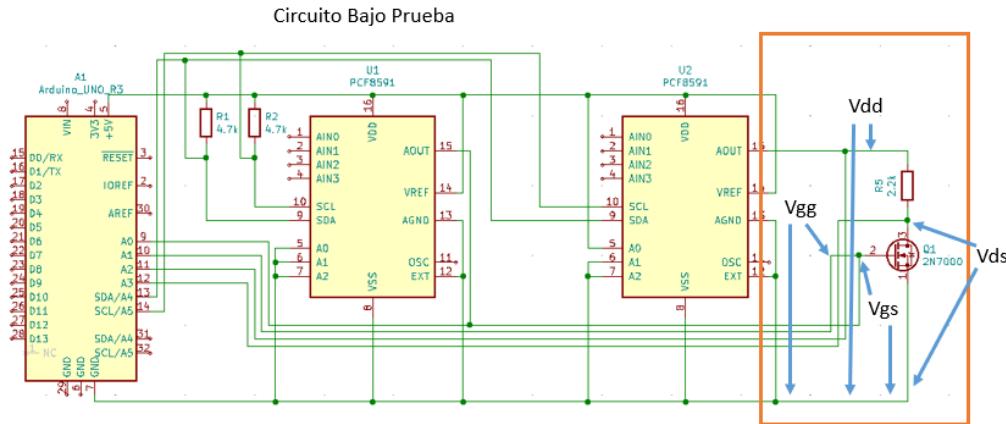


Figura 27. Circuito propuesto con un transistor MOSFET canal n de enriquecimiento 2N7000.

En la figura 27, el recuadro en naranja se observa el circuito bajo prueba para los circuitos **Trazador de curvas (a)** y **Amplificador de una Etapa (b)**.

Se utilizó un transistor MOSFET 2N7000 con una resistencia en drenaje de $2.2\text{k}\Omega$. El valor de esta resistencia de $2.2\text{k}\Omega$ se propone para que la corriente de drenaje sea menor a 10mA, el cual es un poco menos de la mitad del valor máximo de corriente que puede proporcionar el circuito PCF8591, con el fin generar las gráficas del circuito de manera óptima. El valor de esta resistencia puede variar de transistor a transistor, dependiendo de su ganancia.

Se medirán cuatro voltajes, que son:

- V_{gg} : Es el voltaje de polarización aplicado al circuito de compuerta del transistor MOSFET.
- V_{gs} : Es el voltaje en la terminal de compuerta y la terminal de fuente del transistor MOSFET.
- V_{dd} : Es el voltaje de alimentación en el circuito de drenaje del transistor MOSFET.
- V_{ds} : Es el voltaje definido entre la terminal de drenaje y la terminal de fuente del transistor MOSFET.

6.2.2 Circuito Generador de Señal Senoidal (b)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Círculo secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

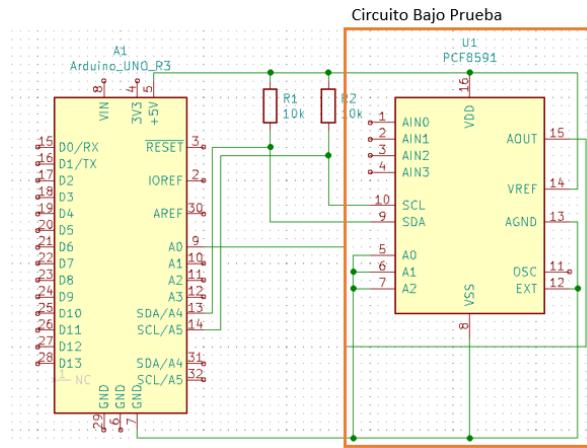


Figura 28. Circuito propuesto utilizando un microcontrolador y un PCF8591 para la construcción de un circuito DDS.

En la figura 28 se muestra el **circuito Generador de Señal Senoidal (b)**. Se utilizó el circuito PCF8591, el cual se comunica con la etapa de adquisición de datos a través del protocolo I2C. Con números en valor binario enviados por comunicación I2C a través del pin 9 (SDA) se pueden generar voltajes analógicos a través del pin 15 (AOUT), los cuales podrán tener diferentes formas de onda. En el caso de este proyecto, se generó una señal senoidal.

6.2.3 Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el **Circuito Secuencial (d)**, se construyó una máquina de estados utilizando dos métodos:

- i. **Por hardware** usando compuertas lógicas y flip flops.
- ii. **Por software** en un microcontrolador mediante la programación en el **lenguaje C**.

El diagrama de estados correspondiente para ambos métodos se muestra a continuación:

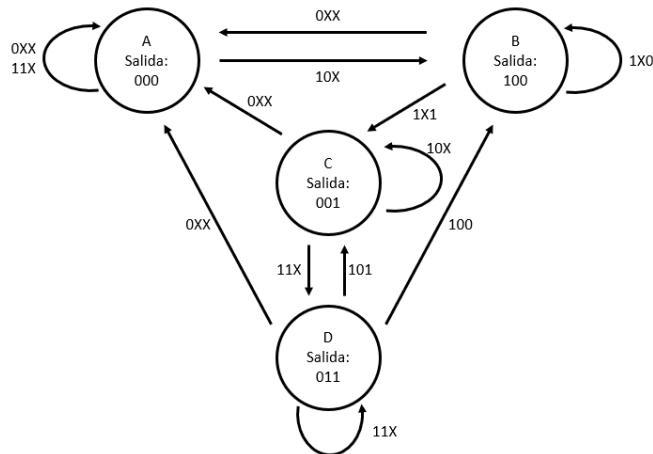


Figura 29. Diagrama de estados utilizado para la construcción de un circuito secuencial.

Este diagrama de estados de la figura 29 se implementó tanto por software utilizando el lenguaje c y por hardware utilizando compuertas lógicas y flip flops.

6.2.3.1 Máquina de estados finitos construida con compuertas lógicas y flip-flops (i)

Con base en el diagrama de estados presentado en la figura 29, se propone el siguiente circuito, en donde se definen las entradas y salidas de la máquina de estados finitos:

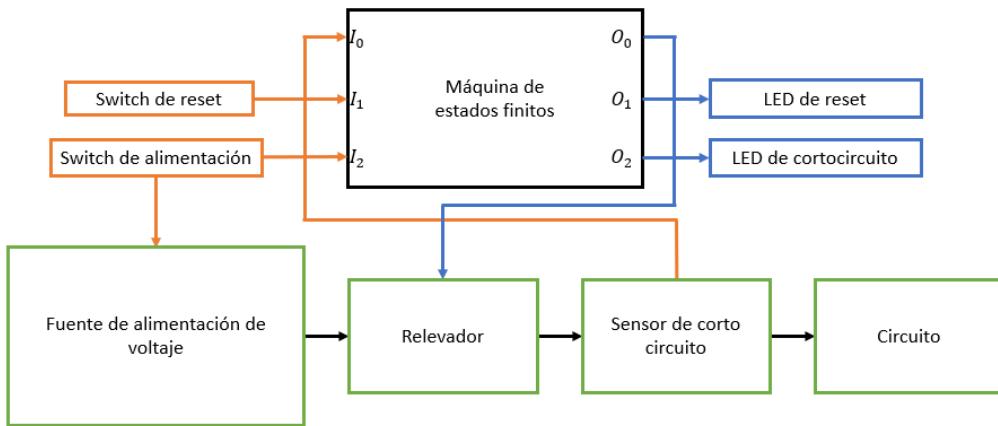


Figura 30. Diagrama a bloques con las salidas y entradas para la máquina de estados

Se consideraron las siguientes **entradas**:

$$\begin{aligned} I_0 &= \text{Sensor de Corto Circuito} \\ I_1 &= \text{Switch de Reset} \\ I_2 &= \text{Switch de Alimentación} \end{aligned}$$

Se consideraron las siguientes **salidas**:

$$\begin{aligned} O_0 &= \text{Led de Corto Circuito} \\ O_1 &= \text{Led de Reset} \\ O_2 &= \text{Relevador de Alimentación} \end{aligned}$$

Con base en las entradas y salidas propuestas, así como en el funcionamiento de la máquina de estados explicada en el apartado 5.4.3, se propuso el empleo de **dos flip flops** tipo D para representar cada uno de los cuatro estados de la máquina:

$$\begin{aligned} S_0 &= \text{Flip flop 0} \\ S_1 &= \text{Flip flop 1} \end{aligned}$$

Las ecuaciones booleanas correspondientes al diagrama de estados de la figura 26, usando **flip flops** tipo D son:

$$\begin{aligned} Q_0 &= I_2[S_1I_1 + S_0\bar{I}_0 + \bar{S}_0\bar{S}_1\bar{I}_1] \\ Q_1 &= I_2[S_1\bar{S}_0 + S_1I_1 + S_0I_0] \end{aligned}$$

El circuito resultante de estas ecuaciones es el siguiente:

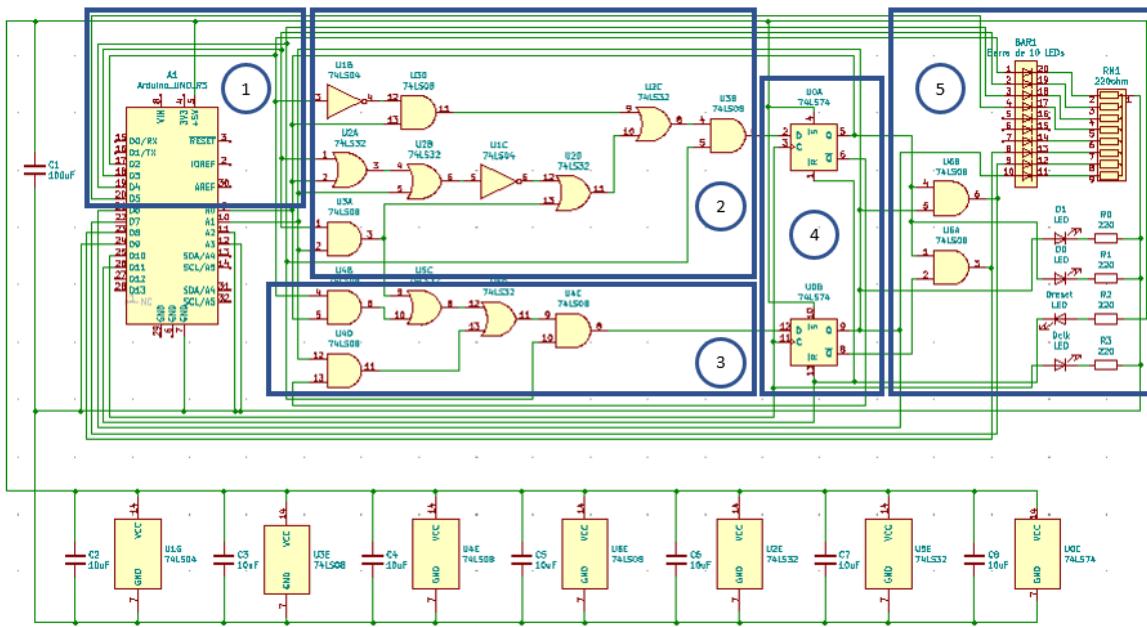


Figura 31. Circuito secuencial implementado por hardware (i) utilizando compuertas lógicas y flip-flops tipo D.

De la figura 31, se observa:

1. En el **recuadro 1**, las entradas I_0 , I_1 e I_2 son generadas por los pines D2, D3 y D4 del microcontrolador.
2. En el **recuadro 2** están las compuertas lógicas para implementar la **ecuación del flip flop Q_0** .
3. En el **recuadro 3** está las compuertas lógicas para implementar la **ecuación del flip flop Q_1** .
4. En el **recuadro 4** están las salidas de los flip flops S_0 y S_1 .
5. En el **recuadro 5** las compuertas lógicas para implementar las ecuaciones de las **salidas O_0 , O_1 y O_2** . También están LEDs que muestran diferentes señales del circuito junto con resistencias de 220Ω .

6.2.3.2 Máquina de estados finitos construida con software en un microcontrolador (ii)

El circuito que implementa la máquina de estados finitos implementada por software es la siguiente:

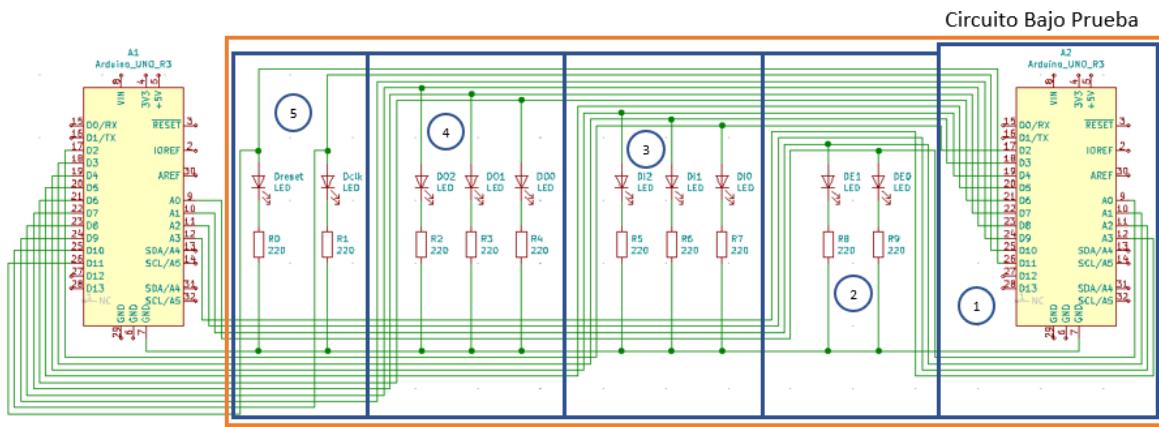


Figura 32. Circuito secuencial implementado por software (ii) utilizando un microcontrolador.

De la figura 32, se observa:

1. En el **recuadro 1** se observa el microcontrolador que contiene el programa que implementa la máquina de estados finitos.
2. En el **recuadro 2** se observan los LEDs junto con sus resistencias que muestran el **estado** de la máquina de estados finitos, generados por los pines A0 y A1 del microcontrolador de la parte del circuito bajo prueba.
3. En el **recuadro 3** se observan los LEDs junto con sus resistencias que muestran las **entradas** de la máquina de estados finitos, generados por los pines D2, D3 y D4 del microcontrolador de la parte del circuito bajo prueba.
4. En el **recuadro 4** se observan los LEDs junto con sus resistencias que muestran las **salidas** de la máquina de estados finitos, generados por los pines D6, D7 y D8 del microcontrolador de la parte del circuito bajo prueba.
5. En el **recuadro 5** se observan los LEDs junto con sus resistencias que muestran el **estado del reloj** y del **reset**, generados por los pines D10 y D11 del microcontrolador correspondientemente de la parte del circuito bajo prueba.

Programa de Arduino para la implementación del Circuito Secuencial

Nombre del programa: Arduino AR 65 Implementación FSM

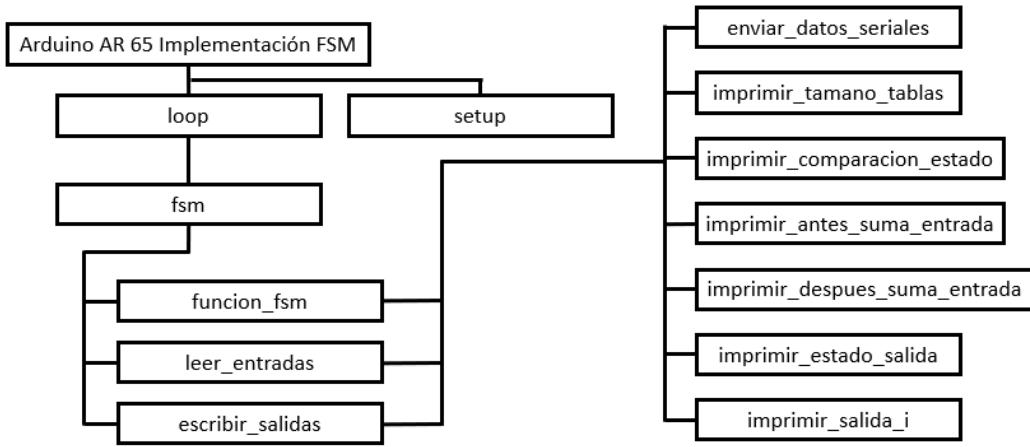


Figura 33. Diagrama a bloques del programa completo de Arduino para la implementación del **Círculo Secuencial (4)**.

Sobre este diagrama, el programa se puede dividir en dos partes, como se indica en la figura 34:

Programa de Arduino para la implementación del Circuito Secuencial
Nombre del programa: Arduino AR 65 Implementación FSM

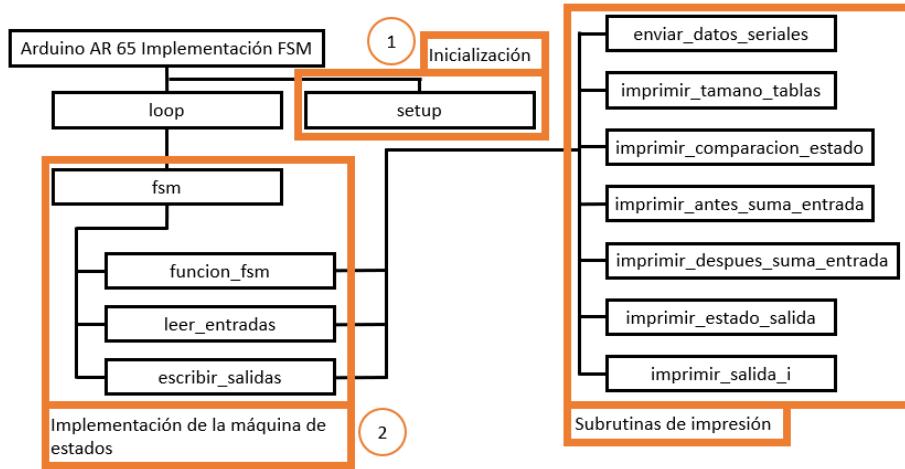


Figura 34. División del diagrama a bloques en dos partes principales.

Para la máquina de estados finita implementada por software utilizando un microcontrolador, el programa correspondiente realiza las siguientes funciones:

Nombre del programa: Arduino AR 65 Implementación FSM

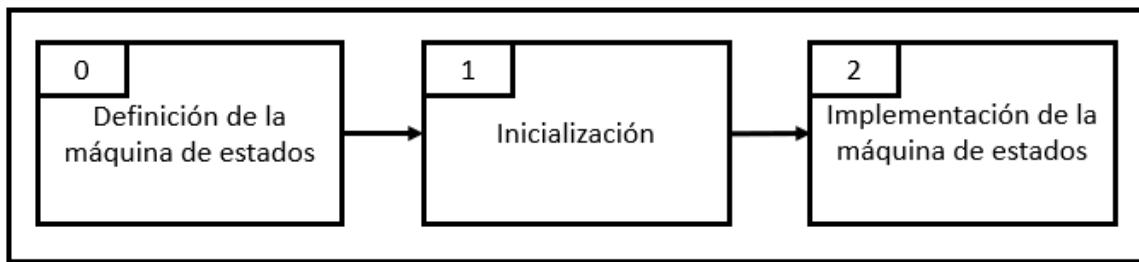


Figura 35. Diagrama a bloques del funcionamiento del programa que implementa la máquina de estados finitos.

Para que pueda funcionar este programa, se realiza la **Definición de la máquina de estados (0)**, en la cual se describe el comportamiento de la máquina de estados finitos.

Explicación de las funciones de cada bloque:

0. **Definición de la máquina de estados:** En esta parte se definen los parámetros de la máquina de estados finitos. Se definen las entradas, estados y salidas de la máquina de estados y las funciones de transiciones entre estados.
1. **Inicialización:** Inicializa los pines de entrada, salida, estados y la comunicación serial. Coloca a la máquina de estados finitos en su estado inicial.
2. **Implementación de la máquina de estados:** Ejecuta el comportamiento de la máquina de estados.

La subrutina de **implementación de la máquina de estados (2)** realiza las siguientes funciones:

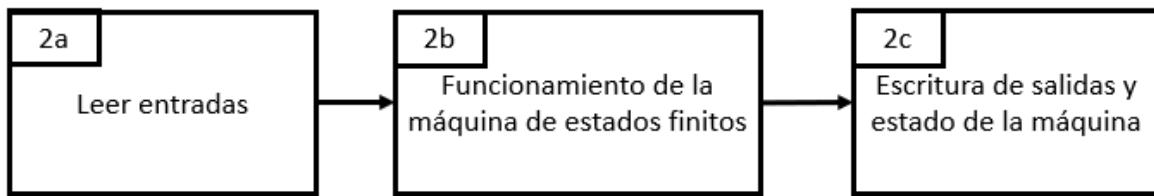


Figura 36. Diagrama a bloques de las funciones que realiza la parte del programa **Implementación de la máquina de estados (2)**.

- 2a. **Lectura de entradas (2a):** Lee los pines de entrada y se convierte este número en un número decimal.
- 2b. **Funcionamiento de la máquina de estados (2b):** En esta parte se verifica si la máquina de estados cambia de estado.
- 2c. **Escritura de salidas y estado de la máquina (2c):** En esta parte se escribe el número binario de salida correspondiente en los pines de salida y el número binario del estado correspondiente en los pines de estado según se encuentre la máquina de estados.

Opcionalmente, en las **subrutinas de impresión** se pueden enviar datos seriales para observar si funciona correctamente la máquina. Sin embargo, esta parte puede omitirse, puesto que solo es para depuración del programa.

El programa completo se presenta en la sección 10.2.4.1.

6.2.3.2.1 Definición de la máquina de estados (0)

Nombre del programa: Arduino AR 65 Implementación FSM



Figura 37. Ubicación del bloque “Definición de la máquina de estados”.

Para poder implementar la máquina de estados finitos de la figura 29, se propone utilizar cuatro constantes y tres tablas:

```
/**  
 * -----  
 * num_estados -> indica cuantos estados tiene la maquina  
 * bits_transicion -> Indica cuantos bits tiene los numeros de las  
 * entradas de transiciones que permiten cambiar entre estados  
 * bits_salida -> Indica la cantidad de bits de salida de la maquina de  
 * estados  
 * -----  
 */  
const int num_estados = 4;  
const int bits_transicion = 3;  
const int bits_salida = 3;  
const int estado_inicial = 0;  
  
/**  
 * -----  
 * Las siguientes tablas especifican el funcionamiento de la maquina de  
 * estados finitos  
 * El prefijo 0B indica que el numero que sigue esta en binario  
 * El tamano de estas tablas esta determinado por las constantes  
 * num_estados, bits_salida y bits_transicion
```

```

 * Cada estado de la maquina corresponde a una entrada de la tabla, como
se indica en cada una de las tablas.
 *
 * tabla_transiciones_estados -> Indica cuales son las transiciones con
las que la maquina cambia de estado
 *                                     Las transiciones que no aparecen, se
asume que la maquina no cambia de estado
 *                                     IMPORTANTE: Se completan la tabla con -1
para las transiciones en las que no hay cambio de estado
 *
 * tabla_transiciones_estado_futuro -> Indica el estado al cual va a
cambiar la maquina en el siguiente pulso
 *                                     de reloj. Cada estado especificado
en esta tabla corresponde a la
 *                                     transicion de la tabla
tabla_transiciones_estados
 *
 * tabla_output_por_estado -> Indica la salida del estado actual de la
maquina
 * -----
-----
 */
const int
tabla_transiciones_estados[num_estados][round(pow(2,bits_transicion))] =
{
{0B100,0B101,-1,-1,-1,-1,-1},//estado 0: Apagado
{0B000,0B001,0B010,0B011,0B101,0B111,-1,-1},//estado 1: Encendido
{0B000,0B001,0B010,0B011,0B110,0B111,-1,-1},//estado 2: Corto circuito
{0B000,0B001,0B010,0B011,0B100,0B101,-1,-1}//estado 3: Reset
};

const int
tabla_transiciones_estado_futuro[num_estados][round(pow(2,bits_transicion))] = {
Apagado           {1,1},           //estado 0:
Encendido funcionando {0,0,0,0,2,2}, //estado 1:
Corto circuito    {0,0,0,0,3,3}, //estado 2:
Reset             {0,0,0,0,1,2} //estado 3:
};

const int tabla_output_por_estado[num_estados] = {
funcionando      (0B000), //estado 0: Apagado
circuito          (0B100), //estado 1: Encendido
                      (0B001), //estado 2: Corto
                      (0B011) //estado 3: Reset
};

```

Figura 38. Fragmento de código que define a la máquina de estados finitos.

Donde:

num_estados es el número de estados de la máquina de estados finitos.

bits_transicion es el número de bits utilizados para las transiciones entre estados.

bits_salida es el número de bits utilizados para la salida de la máquina.

tabla_transiciones_estados es la tabla que indica con cual entrada la máquina de estados cambiará de estado.

tabla_transiciones_estado_futuro es la tabla que indica el estado futuro al que cambia la máquina de estados.

tabla_output_por_estado es la tabla que indica la salida de los estados de la máquina de estados.

Cada renglón de cada una de las tablas corresponde a un estado diferente de la máquina de estado: Como hay 4 estados diferentes, indica que hay 4 renglones por tabla correspondientes a los estados 0 al 3.

Los números en binario que aparecen en las tablas tienen el prefijo 0B seguido de tres bits.

Comparando la tabla **tabla_transiciones_estados** con la tabla 4 de funciones de transición, esta tabla contiene la columna “**entrada**”:

Estado actual	Entrada	Estado siguiente	tabla_transiciones_estados
S_1S_0	$I_2I_1I_0$	Q_1Q_0	
00	0XX, 11X	00	{0B100,0B101,-1,-1,-1,-1,-1}
	10X	01	
01	0XX	00	{0B000,0B001,0B010,0B011,0B101,0B111,-1,-1}
	1X0	01	
	1X1	10	
10	0XX	00	{0B000,0B001,0B010,0B011,0B110,0B111,-1,-1}
	10X	10	
	11X	11	
11	0XX	00	{0B000,0B001,0B010,0B011,0B100,0B101,-1,-1}
	100	01	
	101	10	
	11X	11	

Tabla 5. Comparación para la tabla “tabla_transiciones_estados”

Se observa de la tabla 5 que las combinaciones de bits que no hacen cambiar a la máquina a un estado diferente, se indican con un “-1”. De lo contrario se expresan en binario.

Comparando la tabla **tabla_transiciones_estado_futuro** con la tabla 4 de funciones de transición, esta tabla contiene la columna “**Estado siguiente**”:

Estado actual	Entrada	Estado siguiente	tabla_transiciones_estado_futuro
S_1S_0	$I_2I_1I_0$	Q_1Q_0	
00	0XX	00	{1,1}
	10X	01	
	11X	00	
01	0XX	00	{0,0,0,0,2,2}
	1X0	01	
	1X1	10	
10	0XX	00	{0,0,0,0,3,3}
	10X	10	
	11X	11	
11	0XX	00	{0,0,0,0,1,2}
	100	01	
	101	10	
	11X	11	

Tabla 6. Comparación para la tabla “tabla_transiciones_estado_futuro”

Comparando las tablas 5 y 6, se observa lo siguiente:

Estado	tabla_transiciones_estados	tabla_transiciones_estado_futuro
00	{0B100,0B101,-1,-1,-1,-1,-1,-1}	{1,1}
01	{0B000,0B001,0B010,0B011,0B101,0B111,-1,-1}	{0,0,0,0,2,2}
10	{0B000,0B001,0B010,0B011,0B110,0B111,-1,-1}	{0,0,0,0,3,3}
11	{0B000,0B001,0B010,0B011,0B100,0B101,-1,-1}	{0,0,0,0,1,2}

Tabla 7. Comparación de las tablas “tabla_transiciones_estados” con “tabla_transiciones_estado_futuro”

Se observa que, por cada entrada en valor binario de la tabla “tabla_transiciones_estados” tiene un estado siguiente indicado por el número en decimal indicado en la tabla “tabla_transiciones_estado_futuro”.

Por último, la tabla “**tabla_output_por_estado**” contiene la **salida** del estado correspondiente, como se muestra en la tabla 8:

Estado actual	Salida	tabla_transiciones_estado_futuro
S_1S_0	$O_2O_1O_0$	
00	000	0B000
01	100	0B100
10	001	0B001
11	011	0B011

Tabla 8. Comparación para la tabla “**tabla_output_por_estado**”

6.2.3.2.2 Subrutina de Inicialización (1)

Nombre del programa: Arduino AR 65 Implementación FSM

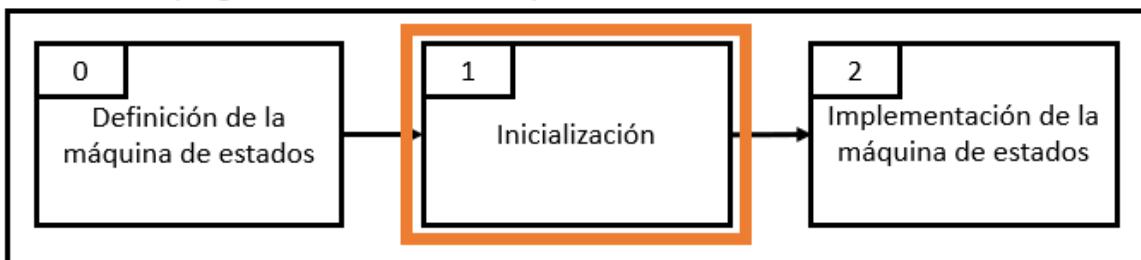


Figura 39. Ubicación del bloque de **Inicialización (1)**.

```

/**
 * Inicializa todos los pines a utilizar, inicializa el reloj y la
 maquina de estados, la comunicacion serial y la comunicacion I2C para
 comunicarse con los dos PCF8591
 */
void setup() {
 // Inicializa los pines en donde se escribirá el estado de la maquina
 for(int i=0;i<sizeof(pin_estados)/sizeof(pin_estados[0]);i++)
 {
   pinMode(pin_estados[i],OUTPUT);
 }

```

```

// Inicializa los pines en donde se leera la entrada
for(int i=0;i<sizeof(pin_entradas)/sizeof(pin_entradas[0]);i++)
{
    pinMode(pin_entradas[i],INPUT);
}

// Inicializa los pines en donde se escribira la salida de la maquina
for(int i=0;i<sizeof(pin_salidas)/sizeof(pin_salidas[0]);i++)
{
    pinMode(pin_salidas[i],OUTPUT);
}

// Se inicializan los pines en donde se leera el reloj y el reset
pinMode(pinCLK,INPUT);
pinMode(pinReset,INPUT);

// --- Inicializa velocidad de transmision serial ---
Serial.begin(9600);

// --- Sincronizacion de comunicacion serial ---
Serial.println("inicio");
estado_actual = estado_inicial;
fsm();
firstTime = millis();
}

```

Figura 40. Subrutina de inicialización

En esta subrutina de inicialización inicializa 3 conjuntos de pines:

- Pines de entradas como **entrada** de la máquina de estados finitos como entrada.
- Pines que muestran el **estado** de la máquina de estados finitos como salida.
- Pines de salida como **salida** de la máquina de estados finitos como salida.

Adicionalmente inicializa otros pines:

- Pin de **reset** para inicializar la máquina
- Pin de **reloj** para sincronizar todas las operaciones que realiza la máquina.

6.2.3.2.3 Subrutina principal loop

Nombre del programa: Arduino AR 65 Implementación FSM

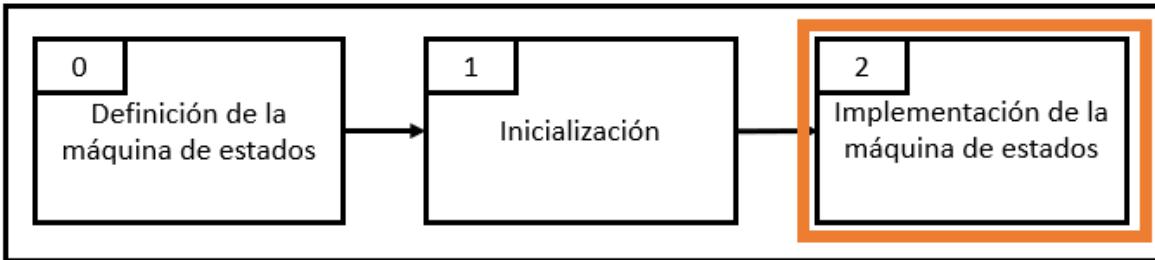


Figura 41. En la subrutina loop se ubica el bloque **Revisión de reset e inicialización de la máquina de estados (2)**.

```
void loop() {
    // put your main code here, to run repeatedly:

    // Lee e identifica un cambio a 1 en el pin del reloj
    // La maquina realiza cambios en los flancos positivos del pin del
    reloj
    clk = digitalRead(pinCLK);
    if(clk == 1)
    {
        // Lee el pin de reset. Si es 1 se inicializa la maquina a su estado
        inicial
        // De lo contrario continua con el funcionamiento de la maquina
        reset = digitalRead(pinReset);
        if(reset == 0)
        {
            estado_actual = estado_inicial;
        }
        fsm();

        // Espera a que el pin de reloj cambie a 0 para continuar
        while(clk == 1){clk = digitalRead(pinCLK);}

    }

    // (Opcional) Cada sampleTime milisegundos envia informacion al puerto
    serie para observar su comportamiento
    /*if ((millis()-lastTime) >= sampleTime)
    {
        //Serial.println(millis()-firstTime);
        //Serial.println(millis()-lastTime);
        lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
        tomaron muestras
        //imprimir_estado_maquina();

    }*/
}
```

Figura 42. Subrutina principal “loop”.

Esta función primero revisa si el botón de reset está presionado al ocurrir en el flanko positivo del reloj:

- Si está presionado, que su valor sea 0, manda a la máquina de estados a su estado inicial.
- En el caso que sea 1 (que no esté presionado) ejecuta el comportamiento de la máquina de estados.

Después se mandan a llamar las funciones que implementan el comportamiento de la máquina de estados.

6.2.3.2.4 Implementación de la máquina de estados (2)

Nombre del programa: Arduino AR 65 Implementación FSM



Figura 43. Ubicación del bloque **Implementación de la máquina de estados (2)**.

La siguiente subrutina, que se ejecuta al final del programa descrito en la sección anterior 6.2.3.2.3, implementa el comportamiento de la máquina de estados:

```
/**  
 * Implementa el funcionamiento de la maquina de estados finitos, de  
acuerdo a los parametros  
 * establecidos al principio del programa  
 */  
void fsm()  
{  
    // Lee la entrada  
    leer_entradas();  
  
    // Determina si cambia de estado  
    funcion_fsm();  
    //escribir_salidas(pin_salidas,salida);  
  
    // Escribe la salida correspondiente al estado actual de la maquina de  
estados  
    escribir_salidas();  
  
    // Escribe el estado actual de la maquina de estados  
    escribir_estados();
```

```

    // (Opcional) Envía datos seriales sobre la entrada, estado y salida de
    // la máquina de estados
    //enviar_datos_seriales();

    // Actualiza el estado para el siguiente pulso de reloj
    estado_actual = estado_futuro;
}

```

Figura 44. Subrutina para la implementación de la máquina de estados

La subrutina de implementación de la máquina de estados realiza las siguientes funciones:

Nombre del programa: Arduino AR 65 Implementación FSM

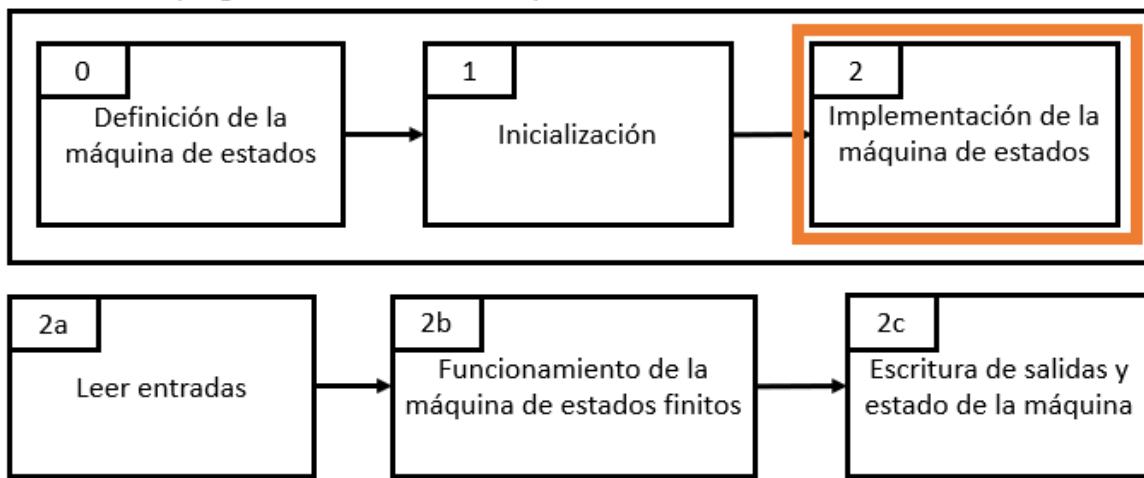


Figura 45. Diagrama a bloques de la función “Implementación de la máquina de estados”

Esta subrutina ejecuta las siguientes funciones:

- 2a. **leer_entradas (2a):** Lee cada uno de los pines de entrada definidos para la máquina
- 2b. **funcion_fsm (2b):** Determina el siguiente estado de la máquina con base en la entrada leída.
- 2c. **escribir_salidas y escribir_estados (2c):** Actualiza los pines de salida con el estado determinado en la función anterior y actualiza los pines de estado de la máquina de estados, determinado por la función funcion_fsm.

6.2.3.2.4.1 Subrutina leer_entradas (2a)

Nombre del programa: Arduino AR 65 Implementación FSM

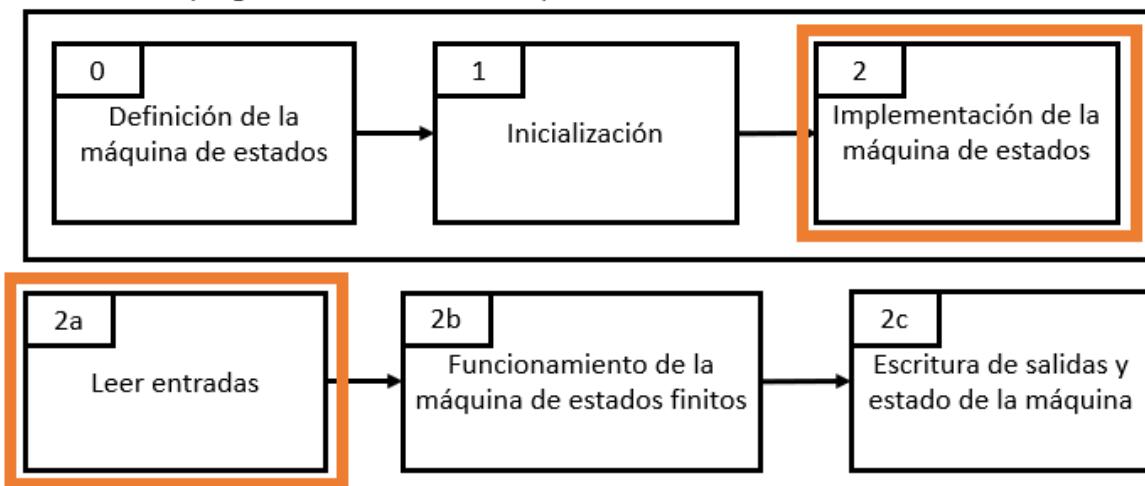


Figura 46. Ubicación del bloque Leer entradas (2a).

```
/**  
 * Lee las entradas en binario que le llegan a la maquina de estados  
 * Convierte este valor leido en un valor en decimal  
 */  
void leer_entradas()  
{  
    entrada = 0x00;  
    for(int i=0;i<bits_transicion;i++) {  
        //imprimir_anter_suma_entrada();  
        entrada += (byte)round(digitalRead(pin_entradas[i])*pow(2,i));  
        //imprimir_despues_suma_entrada(i);  
    }  
    //Serial.print("entrada(final): ");  
    //Serial.println(entrada);  
}
```

Figura 47. Subrutina “leer_entradas”

Esta subrutina lee cada pin de entrada. Cada pin de entrada es visualizado como un bit. Al conjunto de bits se les realiza una conversión para observarlos como un número decimal. Por ejemplo, si se leen en las entradas 101, se convierte a 5.

6.2.3.2.4.2 Subrutina *funcion_fsm* (2b)

Nombre del programa: Arduino AR 65 Implementación FSM

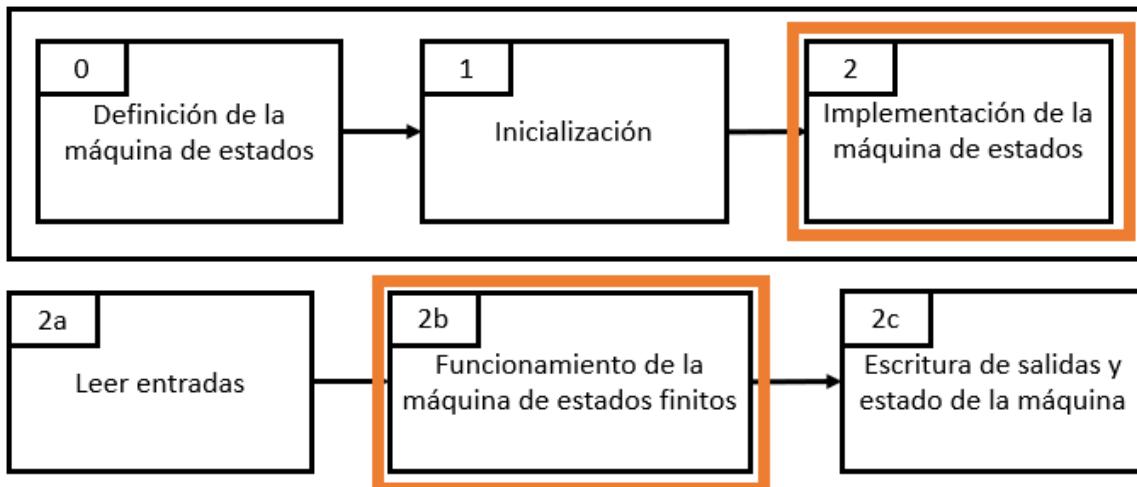


Figura 48. Ubicación del bloque **Funcionamiento de la máquina de estados (2b)**.

```
/**  
 * Implementa el funcionamiento de la maquina de estados, de acuerdo a  
los parametros establecidos al inicio del programa  
*/  
void funcion_fsm()  
{  
    //imprimir_tamano_tablas();  
    // Actualiza la salida de la maquina  
    salida = tabla_output_por_estado[estado_actual];  
  
    // Verifica si hay que hacer algun cambio de estado, comparando con las  
    posibles entradas que indican un cambio de estado  
    for(int i=0;i<(sizeof(tabla_transiciones_estados[estado_actual])/2);i++)  
    {  
        //imprimir_comparacion_estado(i);  
        if(entrada == tabla_transiciones_estados[estado_actual][i])  
        {  
            estado_futuro = tabla_transiciones_estado_futuro[estado_actual][i];  
            //imprimir_cambio_estado();  
            return;  
        }  
    }  
}
```

Figura 49. Subrutina “funcion_fsm”

Esta subrutina realiza la actualización de la máquina de estados de acuerdo con el estado actual en el que se encuentra. Verifica si la máquina de estados necesita cambiar de estado y lo almacena en el estado futuro.

6.2.3.2.4.3 Subrutinas `escribir_salidas` y `escribir_salidas (2c)`

Nombre del programa: Arduino AR 65 Implementación FSM

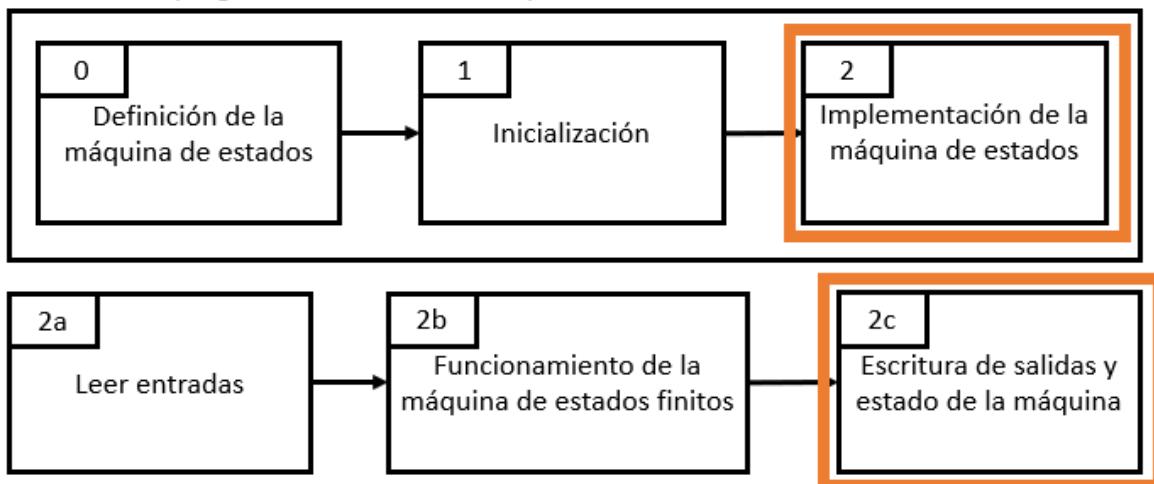


Figura 50. Ubicación del bloque **Escritura de salidas y estado de la máquina (2c)**
(parte de escritura de la salida de la máquina de estados finitos)

```

/**
 * Escribe la salida de la maquina de estados
 * De un valor en decimal lo convierte en binario
 */
void escribir_salidas()
{
    int temp = salida;
    for(int i=(sizeof(pin_salidas)/2)-1;i>=0;i--)
    {
        //imprimir_estado_salida(i,salida);
        if(salida>=(pow(2,i)))
        {
            salida -= round(pow(2,i));
            digitalWrite(pin_salidas[i],HIGH);
            //imprimir_salida_i(i,1);
        }
        else
        {
            digitalWrite(pin_salidas[i],LOW);
            //imprimir_salida_i(i,0);
        }
    }
    salida = temp;
}

```

Figura 51. Subrutina “escribir_salidas”

Esta subrutina actualiza todos los bits de salida de la máquina de estados, convirtiendo el número decimal de salida correspondiente al estado actual de la máquina a un número binario que se escribe en los pines correspondientes de salida. Por ejemplo, si la salida del estado actual es 5, se escribe en los pines de salida como 101.

Nombre del programa: Arduino AR 65 Implementación FSM

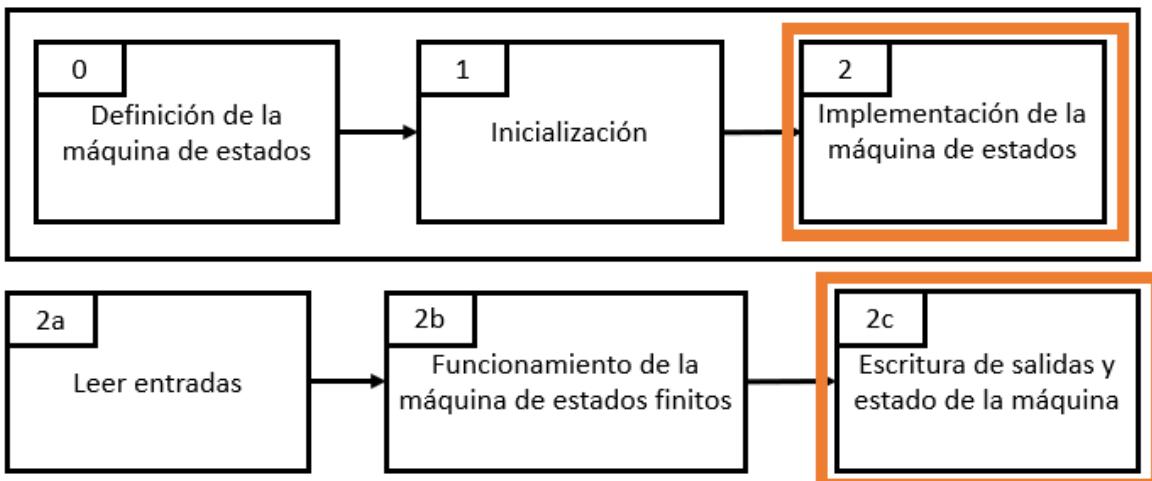


Figura 52. Ubicación del bloque **Escritura de salidas y estado de la máquina (2c)** (parte de escritura del estado de la máquina de estados finitos)

```

/**
 * Escribe el estado actual de la maquina de estados
 * De un valor en decimal lo convierte en binario
 */
void escribir_estados()
{
    int temp = estado_actual;
    //Serial.print("estado actual: ");
    //Serial.println(estado_actual);
    for(int i=(sizeof(pin_estados)/2)-1;i>=0;i--)
    {
        //imprimir_estado_salida(i,estado_actual);
        if(estado_actual>=(pow(2,i)))
        {
            estado_actual -= round(pow(2,i));
            digitalWrite(pin_estados[i],HIGH);
            //imprimir_salida_i(i,1);
        }
        else
        {
            digitalWrite(pin_estados[i],LOW);
            //imprimir_salida_i(i,0);
        }
    }
    //delay(500);
    estado_actual = temp;
}

```

Figura 53. Subrutina “escribir_estados”

Esta función actualiza los pines de estado con el número en binario del estado actual de la máquina de estados. Por ejemplo, si el estado actual es 3, se representa como 11 en los pines de estado.

6.2.3.2.5 Subrutinas adicionales de impresión a monitor serial

Programa de Arduino para la implementación del Circuito Secuencial

Nombre del programa: Arduino AR 65 Implementación FSM

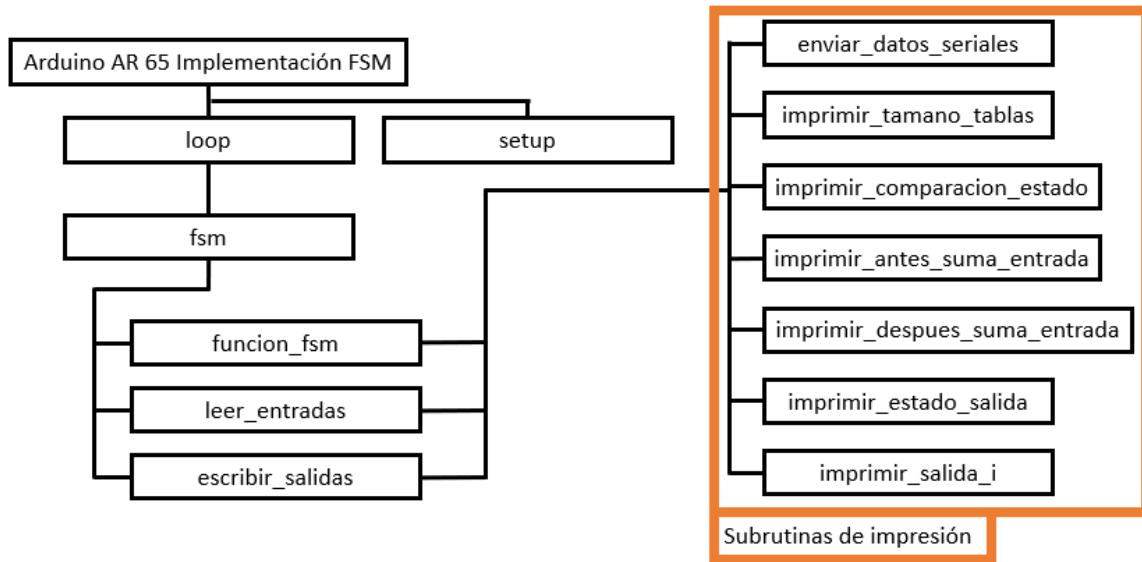


Figura 54. Ubicación de las **subrutinas de impresión**.

Este conjunto de subrutinas sirve para observar el comportamiento de la máquina de estados finitos en el monitor serial. Se pueden observar diferentes partes de la máquina de estados, como la entrada leída, la salida y el estado de la máquina de estados. Cualquiera de estas subrutinas se puede comentar sin afectar el comportamiento del programa, ya que es solo para depuración.

Las subrutinas completas se encuentran descritas en la sección 10.2.4.1.

6.3. Etapa de Adquisición de Datos (2)

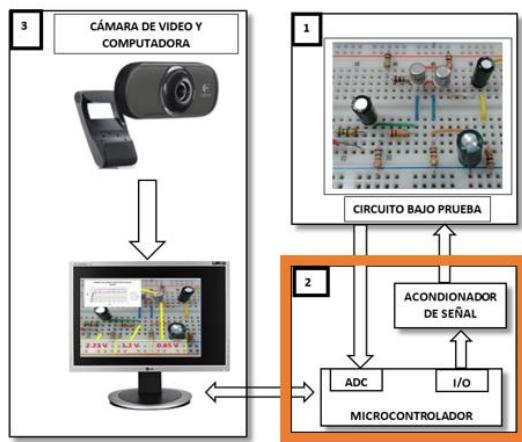


Figura 55. Ubicación del bloque **Etapa de adquisición de datos (2)**.

Este bloque utiliza un microcontrolador junto con circuitos electrónicos que permiten muestrear y procesar señales clave del circuito a analizar. Estas señales serán voltajes nodales y/o niveles lógicos que son importantes para el análisis del circuito.

El diseño detallado de este bloque depende del circuito a analizar, ya que será diferente para un circuito digital con compuertas lógicas o uno analógico con transistores y amplificadores.

Se presentará la etapa de adquisición de datos para los 4 circuitos desarrollados:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.3.1. Trazador de curvas (a)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
 - b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
 - c. **Amplificador de una etapa (c)** con un transistor MOSFET.
 - d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.3.1.1. Etapa de adquisición de datos utilizada para el circuito Trazador de curvas

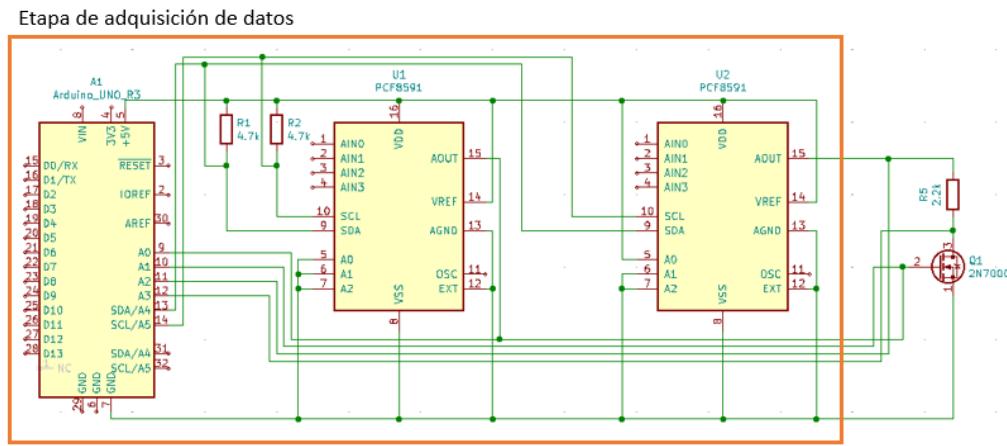


Figura 56. Etapa de adquisición de datos utilizada para el circuito **Trazador de curvas** (a).

Para el circuito **Trazador de Curvas (a)** se utilizaron 2 PCF8591 para generar voltajes analógicos en drenaje y en compuerta. Se pueden variar los voltajes de alimentación V_{gg} (Compuerta) y V_{dd} (Drenaje) al indicar con un valor binario el voltaje a establecer. El número binario que establece estos voltajes lo indica el microcontrolador, el cual está programado para generar una secuencia de voltajes analógicos en los PCF8591 utilizados.

El microcontrolador se conecta con ambos PCF8591 mediante comunicación I2C. De la figura 51, el PCF8591 U1 controla el voltaje de compuerta V_{gg} del transistor MOSFET, mientras que el PCF8591 U2 controla el voltaje de drenaje V_{dd} . Estos voltajes se generan al mandar a los PCF8591 un número hexadecimal que puede variar entre 0x00 y 0xFF.

El microcontrolador también es el encargado de capturar los voltajes del circuito y realizar un filtrado. Lee voltajes del circuito en los pines correspondientes a las entradas analógicas de la siguiente manera:

Voltaje	Pin
V_{gg}	A_0
V_{gs}	A_1
V_{dd}	A_2
V_{ds}	A_3

Tabla 9. Pines del microcontrolador encargados de leer los voltajes del circuito trazador de curvas

Para este circuito, V_{gg} y V_{gs} son prácticamente idénticos. Sin embargo, existe la posibilidad de probar el circuito utilizando un divisor de voltaje para V_{gs} , en cuyo caso las lecturas de V_{gg} y V_{gs} serán diferentes. No se recomienda esta opción (utilizar divisor de voltaje para V_{gs}) ya que el solo con el PCF8591 es más que suficiente para realizar todas las pruebas necesarias.

6.3.1.2. Programa para adquirir señales del circuito Trazador de Curvas

El diagrama a bloques del programa completo es el siguiente:

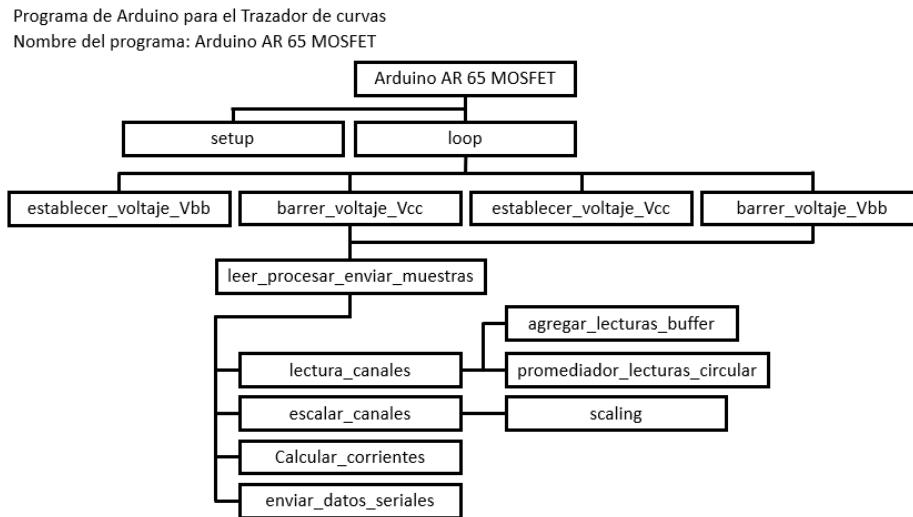


Figura 57. Diagrama a bloques del programa completo de Arduino para adquirir señales del circuito **Trazador de Curvas (a)**.

Sobre este diagrama, el programa se puede dividir en tres partes, como se indica en la figura 58:

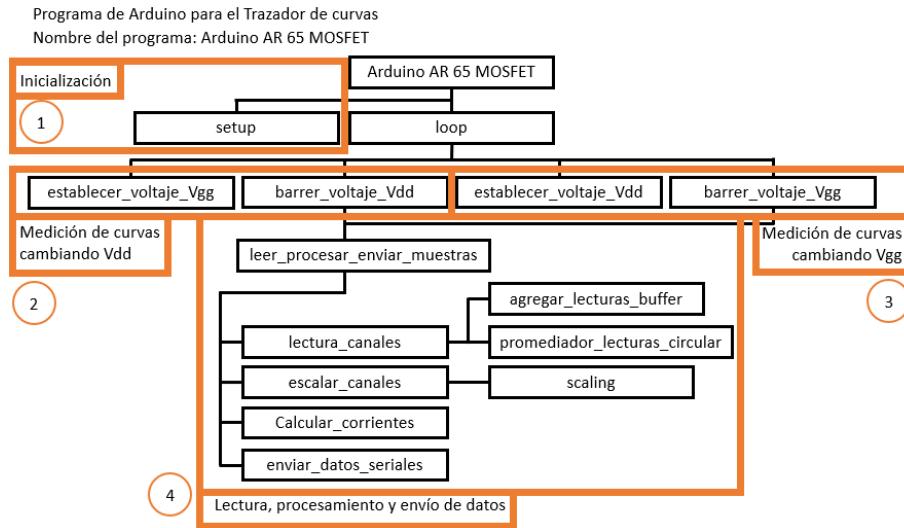


Figura 58. División del diagrama a bloques en cuatro partes principales.

La secuencia del funcionamiento a bloques del programa es el mostrado en la figura 59.

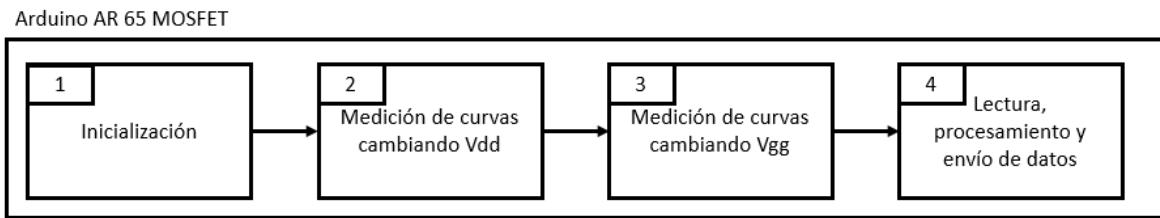


Figura 59. Diagrama a bloques del funcionamiento del programa para el circuito **Trazador de Curvas (a)**.

El programa que se puede dividir en cuatro partes:

1. **Inicialización (1):** Inicia la comunicación serial con la computadora a través de un puerto USB e inicializa la comunicación I2C con los dos circuitos PCF8591.

La subrutina loop agrupa las tres partes restantes:

2. **Medición de curvas cambiando V_{dd} (2):** El microcontrolador envía valores en binario al PCF8591 que controla el voltaje V_{dd} para barrer este voltaje de 0 a 5V, mientras fija el PCF que controla el voltaje V_{gg} . Con estas mediciones es posible obtener las curvas I_d vs V_{ds} .
3. **Medición de curvas cambiando V_{gg} (3):** El microcontrolador envía valores en binario al PCF8591 que controla el voltaje V_{gg} para barrer este voltaje, mientras fija

el PCF8591 que controla el voltaje V_{dd} . Con estas mediciones es posible obtener las demás curvas.

4. **Lectura, procesamiento y envío de datos (4):** Para cada parte **Medición de curvas cambiando V_{dd} (2)** y **Medición de curvas cambiando V_{gg} (3)**, el microcontrolador lee los diferentes voltajes del circuito Trazador de Curvas, los procesa y los envía a la computadora.

La parte de **Medición de curvas cambiando V_{dd} (2)** realiza 3 funciones:

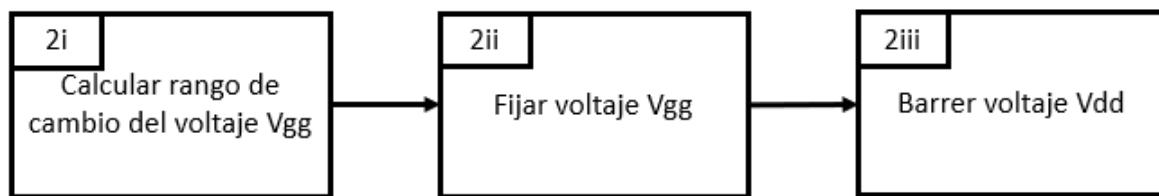


Figura 60. Diagrama a bloques de las funciones que realiza la parte del programa de medición de curvas cambiando V_{dd} (2).

- 2i. **Calcular rango de cambio de voltaje V_{gg} (2i):** Se calcula entre qué valores va a cambiar V_{gg} . Se calculan 5 voltajes entre los límites inferior y superior de ese rango.
- 2ii. **Fijar el voltaje V_{gg} (2ii):** Con uno de los voltajes de voltaje obtenidos en el punto anterior, se actualiza el valor de V_{gg} al enviar el número binario correspondiente al PCF8591 que controla este voltaje.
- 2iii. **Barrer voltaje V_{dd} (2iii):** Se actualiza el voltaje V_{dd} desde 0 hasta 5V enviando números binarios al PCF8591 del voltaje V_{dd} . Cada vez que se actualiza, se toman muestras de los diferentes voltajes de la respuesta del transistor.

Para la parte de **Medición de curvas cambiando V_{gg} (3)** se siguen los mismos pasos, invirtiendo los voltajes V_{gg} por V_{dd} y viceversa. Esto es:

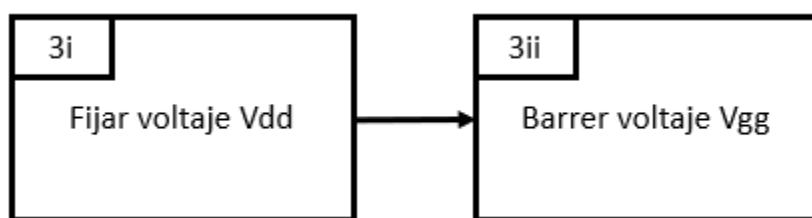


Figura 61. Diagrama a bloques de las funciones que realiza la parte del programa de medición de curvas cambiando V_{gg} (3).

- 3i. **Fijar el voltaje V_{dd} (3i):** Se actualiza el valor de V_{dd} al enviar el número binario correspondiente al PCF8591 que controla este voltaje.

- 3ii. **Barrer voltaje V_{gg} (3ii):** Se actualiza el voltaje V_{gg} desde 0 hasta 5V enviando números binarios al PCF8591 del voltaje V_{gg} . Cada vez que se actualiza, se toman muestras de los diferentes voltajes de la respuesta del transistor.

Para la parte de **lectura, procesamiento y envío de datos (4)** se realizan 4 funciones:

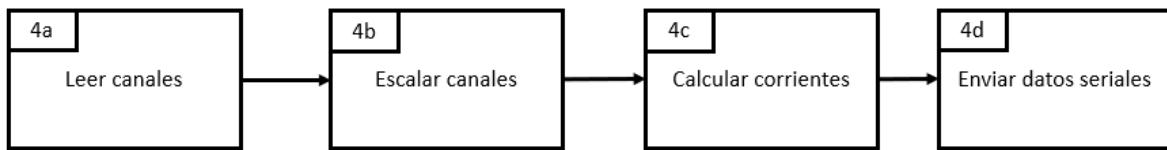


Figura 62. Diagrama a bloques de las funciones que realiza la parte de **lectura, procesamiento y envío de datos (4)**.

Una vez realizado esta parte, se toman muestras de la respuesta del transistor, para lo cual se siguen los siguientes pasos:

- 4a. **Leer canales (4a):** Lee cuatro voltajes que son: V_{gg} , V_{gs} , V_{dd} y V_{ds} . Es posible aplicar filtros a estas mediciones.
- 4b. **Escalar voltajes (4b):** Escala estos cuatro voltajes a el valor de la fuente de alimentación, en caso de que no entregue exactamente 5V.
- 4c. **Calcular corrientes (4c):** Calcula las corrientes I_g e I_d como la diferencia de los voltajes.
- 4d. **Enviar datos seriales (4d):** Envía los datos leídos y procesados a la computadora.

El programa completo se presenta al final del documento en la sección 10.2.1.1.

6.3.1.2.1. Subrutina de inicialización (1)

Nombre del programa: Arduino AR 65 MOSFET

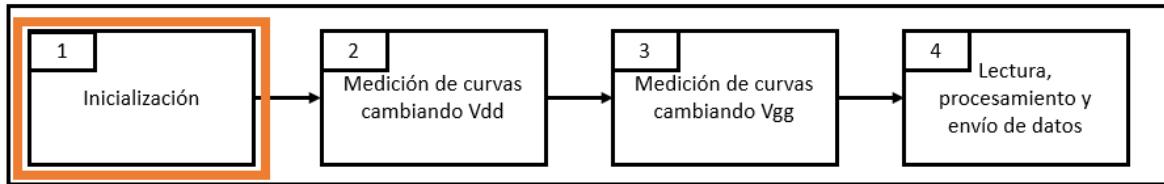


Figura 63. Ubicación del bloque Inicialización (1).

```
/**  
 * Inicializa tanto la comunicacion serial como la comunicacion I2C para  
comunicarse con los dos PCF8591  
*  
*/  
void setup() {  
    Wire.begin();  
    // --- Inicializa velocidad de transmision serial ---  
    Serial.begin(9600);  
  
    // --- Sincronizacion de comunicacion serial ---  
    Serial.println("inicio");  
}
```

Figura 64. Subrutina de Inicialización

Se observa que se inicializa la comunicación I2C y serial. Se manda un mensaje de inicio para sincronización con el programa de Python en la computadora.

6.3.1.2.2. Subrutina principal loop (2,3,4)

Nombre del programa: Arduino AR 65 MOSFET

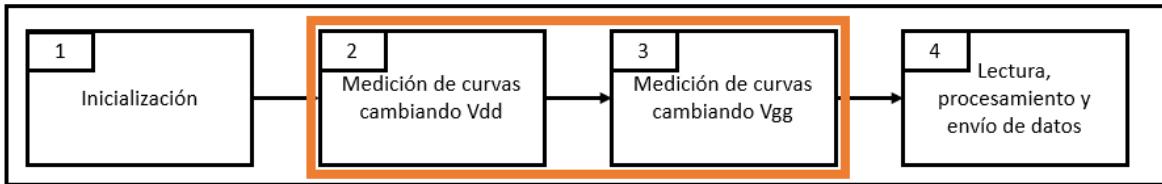


Figura 65. Ubicación de los bloques **Medición de curvas cambiando Vdd (2)** y **Medición de curvas cambiando Vgg (3).**

```

/**
 * Se miden cinco curvas para graficar Id vs Vds. Posteriormente se toma
una ultima curva para graficar las demás curvas
*/
void loop() {
    // --- Toma muestras cada sampleTime ms ---
    if (millis()-lastTime > sampleTime) {

        lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
tomaron muestras

        if(numero_curva<5) // ¿Se toman muestras de las primeras 5 curvas
para Id vs Vds?
        {
            int valor;
            // --- 5 Curvas para Id vs Vds [MOSFET] ---
            valor = map(numero_curva,0,5,102,122); //probar tambien con
116 - 132

            establecer_voltaje_Vgg(valor);
            barrer_voltaje_Vdd();
        }
        else if(numero_curva==5)
        {
            // --- 1 Curva para Ig vs Vgs , Id vs Vgs , entre otras ---
            establecer_voltaje_Vdd(255);
            barrer_voltaje_Vgg();
        }
        else
        {
            numero_curva = 0;
        }
    }
}

```

Figura 66. Subrutina principal “loop”.

Primero se verifica que ha pasado el tiempo para tomar una muestra. Después se toman mediciones para 5 curvas utilizando el bloque **Medición de curvas cambiando el voltaje V_{dd} (2)** y después se toman mediciones para 1 curva utilizando el bloque **Medición de curvas cambiando el voltaje V_{gg} (3)**.

Comparando el código de la figura 66 con el diagrama para la parte de **Medición de curvas cambiando el voltaje V_{dd} (2)** de la figura 60, cada una de las siguientes tres líneas de código corresponde a una función del bloque.

Diagrama a bloques de la parte Medición de curvas cambiando el voltaje V_{dd} (2)	Código
Calcular rango de voltaje V_{gg} (2i)	valor = <code>map(numero_curva, 0, 5, 102, 122);</code> //116 - 132 anterior
Fijar el voltaje V_{gg} (2ii)	establecer_voltaje_Vgg(valor);
Barrer voltaje V_{dd} (2iii)	barrer_voltaje_Vdd();

Tabla 10. Comparación de las funciones que realiza la parte del programa **Medición de curvas cambiando el voltaje V_{dd} (2)** con su diagrama a bloques.

Comparando el código de la figura 66 con el diagrama para la parte de **Medición de curvas cambiando el voltaje V_{gg} (3)** de la figura 61, cada una de las siguientes dos líneas de código corresponde a una función del bloque.

Diagrama a bloques de la parte Medición de curvas cambiando el voltaje V_{gg} (3)	Código
Fijar el voltaje V_{dd} (3i)	establecer_voltaje_Vdd(255);
Barrer voltaje V_{gg} (3ii)	barrer_voltaje_Vgg();

Tabla 11. Comparación de las funciones que realiza la parte del programa **Medición de curvas cambiando el voltaje V_{gg} (3)** con su diagrama a bloques.

6.3.1.2.3. Subrutinas para la medición de curvas barriendo Vdd (2)

Nombre del programa: Arduino AR 65 MOSFET

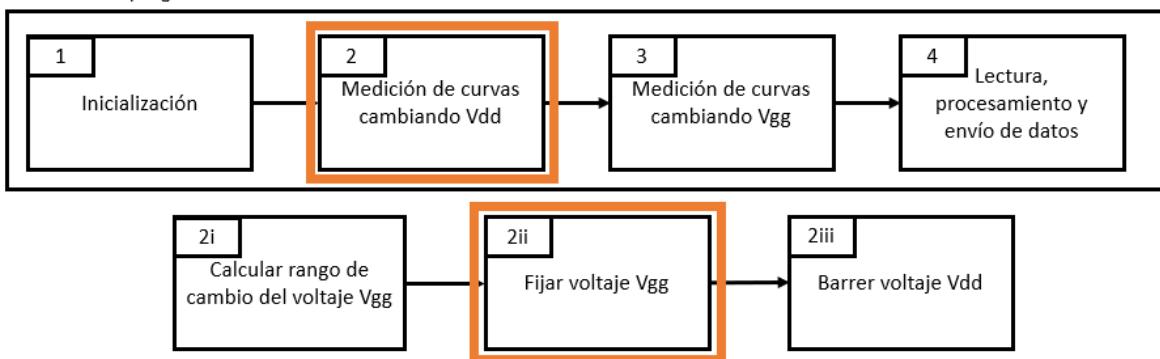


Figura 67. Ubicación del bloque **Fijar voltaje Vgg (2ii)**.

```
/**  
 * Establece un valor de voltaje a Vgg en el PCF8591 correspondiente  
 * @param valor -> Es un numero en formato hexadecimal [0,255] que  
 representa un voltaje [0,5V]  
 */  
void establecer_voltaje_Vgg(int valor){  
    Wire.beginTransmission(PCF_Vgg); // wake up PCF_Vgg  
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)  
    Wire.write(valor);  
    Wire.endTransmission(); // end transmission  
}
```

Figura 68. Subrutina “establecer_voltaje_Vgg”

Esta función establece una comunicación I2C con el PCF8591 que controla el voltaje V_{gg} y cambia el valor de este voltaje.

Nombre del programa: Arduino AR 65 MOSFET

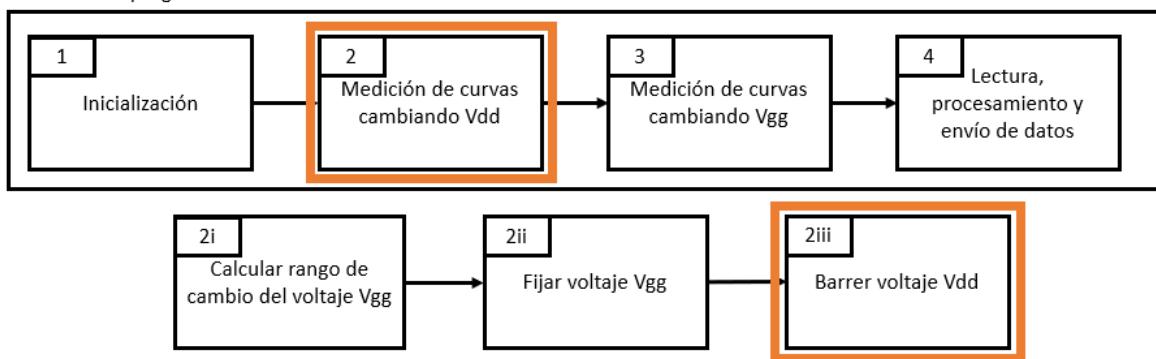


Figura 69. Ubicación del bloque **Barrer voltaje Vdd (2iii)**.

```

/**
 * Actualiza Vdd con diferentes valores de entre 0V y 5V
 * Despues envia por el puerto serie las muestras leidas
 *
 */
void barrer_voltaje_Vdd() {
    // --- Barre durante 50 muestras ---
    for (int i_muestra=0; i_muestra<250; i_muestra+=5) {
        establecer_voltaje_Vdd(i_muestra);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}
  
```

Figura 70. Subrutina “barrer_voltaje_Vdd”.

Esta subrutina establece el voltaje V_{dd} entre un valor de 0 a 5V. Luego indica que lee, procesa y envía los datos leídos a la computadora.

La subrutina establecer_voltaje_Vdd se presenta en la siguiente sección 6.3.1.2.4.

6.3.1.2.4. Subrutinas para la medición de curvas barriendo V_{gg} (3)

Nombre del programa: Arduino AR 65 MOSFET

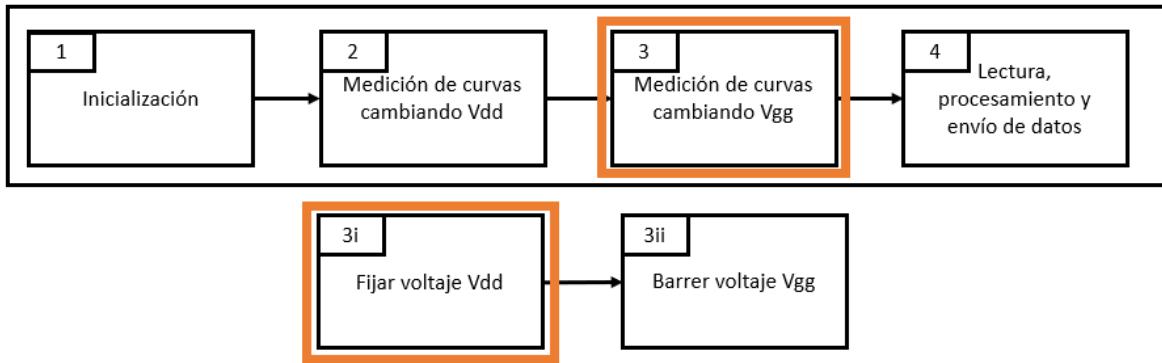


Figura 71. Ubicación del bloque Fijar voltaje Vdd

```

/**
 * Establece un valor de voltaje a Vdd en el PCF8591 correspondiente
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
 */
void establecer_voltaje_Vdd(int valor){
    Wire.beginTransmission(PCF_Vdd); // wake up PCF_Vdd
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}

```

Figura 72. Subrutina “establecer_voltajeVdd”.

Esta función establece una comunicación I2C con el PCF que controla el voltaje V_{gg} y cambia el valor de este voltaje.

Nombre del programa: Arduino AR 65 MOSFET

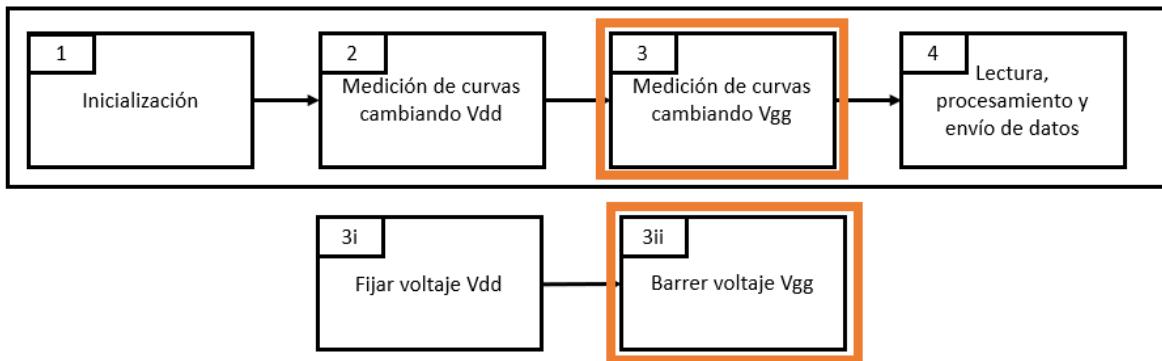


Figura 73. Ubicación del bloque **Barrer voltaje Vgg (3ii)**.

```
/**  
 * Actualiza Vgg con diferentes valores de entre 0V y 5V  
 * Despues envia por el puerto serie las muestras leidas  
 */  
void barrer_voltaje_Vgg(){  
    // --- Transistor MOSFET: valores de Vgg mayores a Vth ---  
    for (int i_muestra=0; i_muestra<150; i_muestra+=3){  
        establecer_voltaje_Vgg(i_muestra);  
        leer_procesar_enviar_muestras();  
    }  
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva  
}
```

Figura 74. Subrutina “barrer_voltaje_Vgg”

Esta subrutina establece el voltaje V_{dd} entre un valor de 0 a 5V. Luego indica que lee, procesa y envía los datos leídos a la computadora.

La subrutina establecer_voltaje_Vdd fue presentada en la sección anterior 6.3.1.2.3.

6.3.1.2.5. Subrutinas para la lectura, procesamiento y envío de las muestras leídas (4)

Nombre del programa: Arduino AR 65 MOSFET

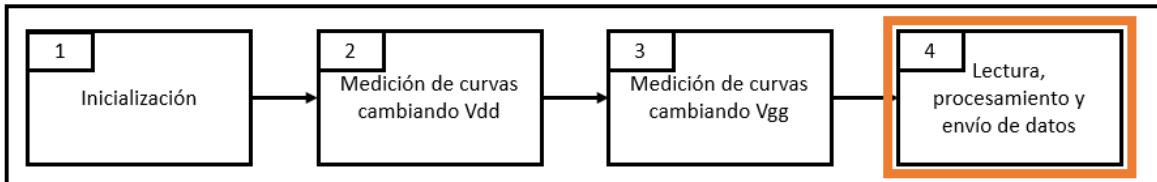


Figura 75. Ubicación del bloque **Lectura, procesamiento y envío de datos (4).**

```
/**  
 * Lee los canales analogicos del circuito Vdd, Vds, Vgg y Vgs (4  
voltajes)  
 * Escala los canales al voltaje de la fuente de alimentacion  
 * Calcula las corrientes a partir de la resta de voltajes  
 * Envia los datos de todos los canales por el puerto serie  
 *  
 */  
void leer_procesar_enviar_muestras(){  
    // --- Lecura de los canales ---  
    leer_canales(); // Lee los 4 voltajes del circuito  
  
    // --- Escalamiento de canales ---  
    escalar_canales(Vdd); // Escala los 4 voltajes leidos al voltaje de  
alimentacion Vdd  
  
    // -- calculo de corrientes  
    calcular_corrientes(); // Calcula las corrientes con los voltajes  
escalados  
  
    // --- Transmision de cada valor al programa en Python ---  
    enviar_datos_serials(); // Envia los 4 voltajes escalados y las 2  
corrientes calculadas  
}
```

Figura 76. Subrutina “leer_procesar_enviar_muestras”

Esta subrutina agrupa cuatro subrutinas, que son ejecutadas después de haber actualizado ambos PCF8591. La siguiente tabla 12 realiza una comparación de las subrutinas agrupadas con el diagrama a bloques de la figura 60.

Diagrama a bloques de la parte Lectura, procesamiento y envío de datos (4)	Código
Lectura de canales (4a)	lectura_canales();
Escalar_canales (4b)	escalar_canales(Vdd);
Calcular corrientes (4c)	calcular_corrientes();
Enviar datos seriales (4d)	enviar_datos_seriales();

Tabla 12. Comparación de las funciones que realiza la parte del programa **Lectura, procesamiento y envío de datos (4)** con su diagrama a bloques.

6.3.1.2.5.1. Subrutina para leer los canales (4a)

Nombre del programa: Arduino AR 65 MOSFET

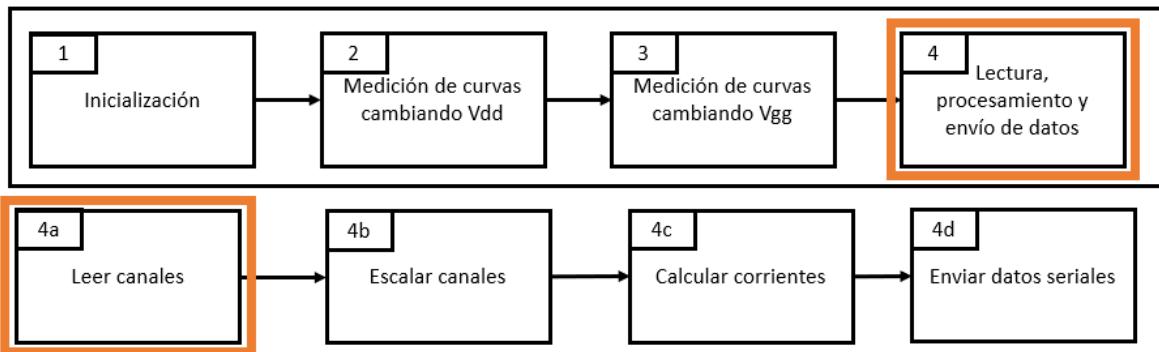


Figura 77. Ubicación del bloque Leer canales (4a).

```

/**
 * Lee los 4 voltajes del circuito. [Vgg,Vgs,Vdd,Vds]
 */
void leer_canales()
{
    if (Filtrado_Circular)
    {
        // --- CON FILTRADO CIRCULAR con buffer circular de 32 muestras ---
        for (int b=0; b<bufferSize; b++)
        {
            canales_medidos[0] = analogRead(PinVgg); // Vgg
            canales_medidos[1] = analogRead(PinVgs); // Vgs
            canales_medidos[2] = analogRead(PinVdd); // Vdd
            canales_medidos[3] = analogRead(PinVds); // Vds

            // Filtrado circular en cada lectura. SIN mezclar muestras en
            tiempos diferentes
            agregar_lecturas_buffer(canales_medidos);
            promediador_lecturas_circular(canales_medidos);
        }
    } else {
        // --- SIN FILTRADO CIRCULAR ---
        canales_medidos[0] = analogRead(PinVgg); // Vgg
        canales_medidos[1] = analogRead(PinVgs); // Vgs
        canales_medidos[2] = analogRead(PinVdd); // Vdd
        canales_medidos[3] = analogRead(PinVds); // Vds

        for(int j=0; j<4; j++)
        {
            canales[j] = canales_medidos[j];
        }
    }
}

```

Figura 78. Subrutina “lectura_canales”.

Esta subrutina lee los cuatro voltajes del circuito V_{gg} , V_{gs} , V_{dd} y V_{ds} . También puede aplicar un filtrado circular a las muestras, dependiendo de la variable booleana Filtrado_Circular.

6.3.1.2.5.2. Subrutina para escalar canales (4b)

Nombre del programa: Arduino AR 65 MOSFET

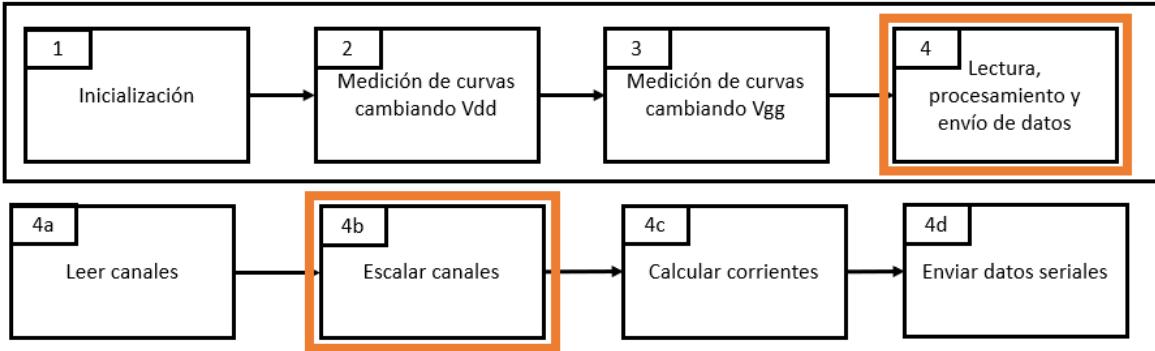


Figura 79. Ubicación del bloque **Escalar canales (4b)**.

```

/**
 * Escala los 4 canales de voltaje, que varian entre [0,1023] a un numero
 que representa el voltaje [0,5]
 */
void escalar_canales(double Vdd_ref)
{
    for(int i=0;i<numero_entradas_analogicas;i++)
    {
        voltajes[i] = scaling(canales[i], 0, 1023, 0, Vdd_ref); // 
Vgg,Vbs,Vdd,Vds
    }
}

```

Figura 80. Subrutina “escalar_canales”.

En caso de que la fuente de alimentación proporcionada por el microcontrolador no generara exactamente 5V, se utiliza una subrutina para escalar todas las lecturas de los cuatro canales al valor medido de alimentación con un multímetro.

Se utiliza una subrutina de escalado que es la presentada en la figura 81:

```
/**  
 * Escala valores que se sabe que varian entre un intervalo definido  
[in_min,in_max] a otro intervalo [out_min,out_max]  
* @param x -> es el valor a escalar  
* @param in_min -> es el valor minimo que puede tomar x  
* @param in_max -> es el valor maximo que puede tomar x  
* @param out_min -> es el valor minimo al cual se escala x  
* @param out_max -> es el valor maximo al cual se escala x  
*  
*/  
float scaling(float x, float in_min, float in_max, float out_min, float  
out_max)  
{  
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;  
}
```

Figura 81. Subrutina “scaling”.

Esta subrutina escala valores entre dos rangos diferentes. Se implementa la siguiente ecuación:

$$x_{escalado} = \frac{(x - in_{min})(out_{max} - out_{min})}{(in_{max} - in_{min})} + out_{min}$$

Donde:

x es el número a escalar.

in_min es el límite mínimo que puede tomar x antes del escalado.

in_max es el límite máximo que puede tomar x antes del escalado.

out_min es el nuevo límite mínimo que tomará x después del escalado.

out_max es el nuevo límite máximo que tomará x después del escalado.

6.3.1.2.5.3. Subrutina para calcular las corrientes del circuito (4c)

Nombre del programa: Arduino AR 65 MOSFET

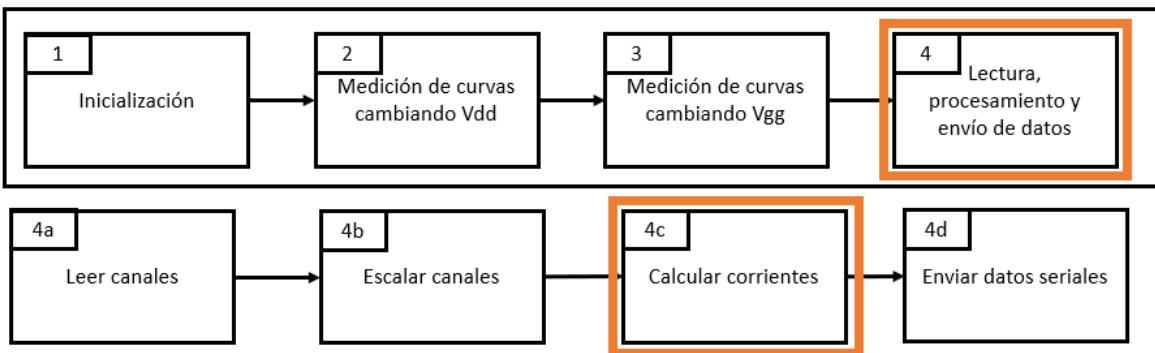


Figura 82. Ubicación del bloque **Calcular corrientes (4c)**.

```

/**
 * Calcula las corrientes con los valores de voltaje escalados [Ig, Id]
 * NOTA: Las corrientes Ig e Id no estan escaladas al valor medido de las
resistencias Rg y Rd.
* Solo son una diferencia de voltajes. Ig = Vgg - Vgs
* Id = Vdd - Vds
*/
void calcular_corrientes()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        corrientes[i] = (voltajes[i*2]-voltajes[i*2+1]);
    }
}

```

Figura 83. Subrutina “calcular_corrientes”.

Se calculan las corrientes I_g e I_d se calculan de la siguiente manera

$$I_g = V_{gg} - V_{gs}$$

$$I_d = V_{dd} - V_{ds}$$

Las corrientes I_g e I_d no se dividen entre la resistencia correspondiente (ni R_g para I_g ni R_d para I_d) para evitar manejar números muy pequeños que sean difíciles de representar en un tipo de dato flotante.

6.3.1.2.5.4. Subrutina para enviar datos seriales (4d)

Nombre del programa: Arduino AR 65 MOSFET

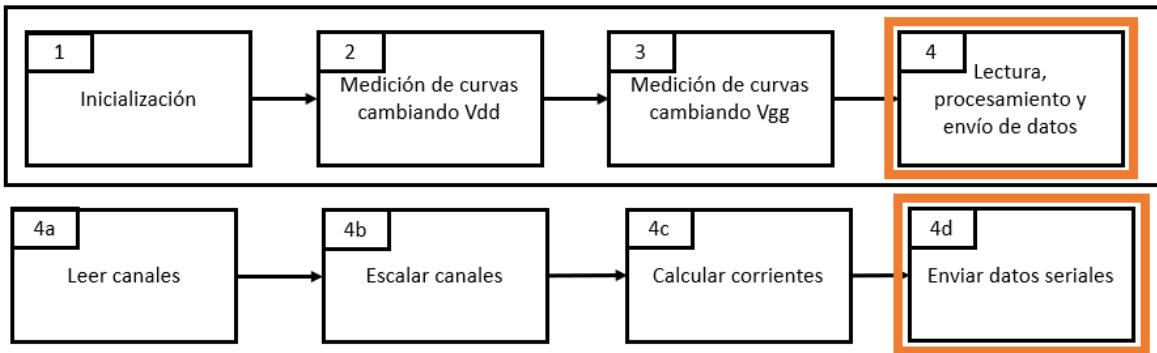


Figura 84. Ubicación del bloque **Enviar datos seriales (4d)**.

```

/*
 * Envía todos los datos por el puerto serial
 * En orden, se envian: [Vgg,Vgs,Ig,Vdd,Vds,Id]
 */
void enviar_datos_seriales()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        Serial.println(voltajes[i*2],4); //Vgg,Vdd
        Serial.println(voltajes[i*2+1],4); //Vgs,Vds
        Serial.println(corrientes[i],4); //Ig,Id
    }
}


```

Figura 85. Subrutina “enviar_datos_seriales”.

Una vez realizados tanto la lectura de canales, escalado de voltajes y cálculo de corrientes, se envían todas las muestras por el puerto serial hacia la computadora

Variable enviada	Orden de envío
V_{gg}	1
V_{gs}	2
I_g	3
V_{dd}	4
V_{ds}	5
I_d	6

Tabla 13. Orden de envío de variables del circuito a la computadora a través del puerto serial.

6.3.1.2.6. Subrutinas de filtrado

Programa de Arduino para el Trazador de curvas
Nombre del programa: Arduino AR 65 MOSFET

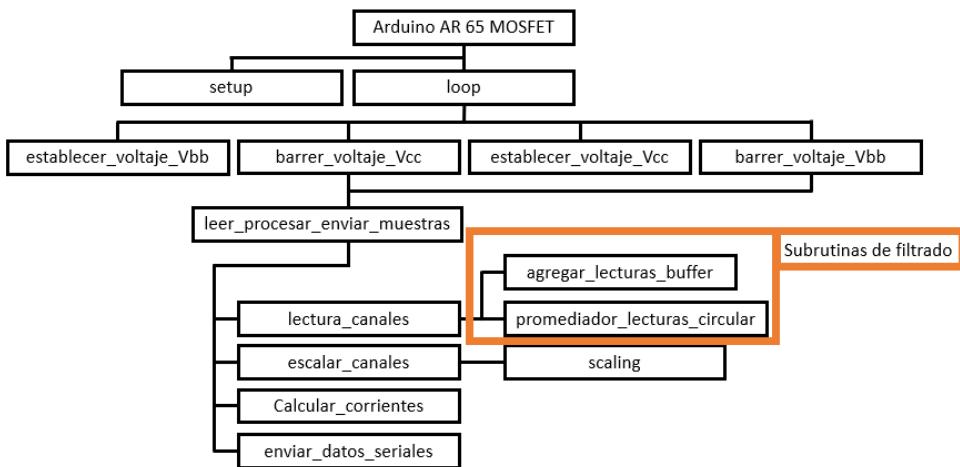


Figura 86. Ubicación de las **subrutinas de filtrado**.

Las subrutinas de filtrado realizan lo siguiente:

- **Subrutina “agregar_lecturas_buffer”:** Lee la misma muestra un determinado número de veces y las almacenan en un buffer.
- **Subrutina “promediador_lecturas_circular”:** Promedia todas las muestras leídas.

```

// ****
// 
// FILTRO CIRCULAR (promediador) con buffer de 32 muestras
// 
// ****
// --- Inicialización del buffer ---
void agregar_lecturas_buffer(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        buffer_promediador[i][index[i]] = canales_medidos[i];
        index[i] += 1;
        if (index[i] >= bufferSize) index[i] = 0;
    }
}

```

Figura 87. Subrutina “agregar_lecturas_buffer”.

Esta subrutina guarda un número determinado de muestras en un buffer.

```
// --- Filtro circular: promediador ---
void promediador_lecturas_circular(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        long sum = 0;
        for (int k=0; k<bufferSize; k++)
        {
            sum += buffer_promediador[i][k];
        }
        canales[i] = (int)(sum/bufferSize);
    }
}
```

Figura 88. Subrutina “promediador_lecturas_circular”.

Esta subrutina suma todas las muestras que hay en el buffer y las promedia.

6.3.2. Circuito generador de señal senoidal (DDS) (b)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
 - b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
 - c. **Amplificador de una etapa (c)** con un transistor MOSFET.
 - d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.3.2.1. Etapa de adquisición de datos utilizado para el circuito Generador de Señal Senoidal

Para adquirir la señal del circuito DDS, se conectó al microcontrolador de la siguiente manera:

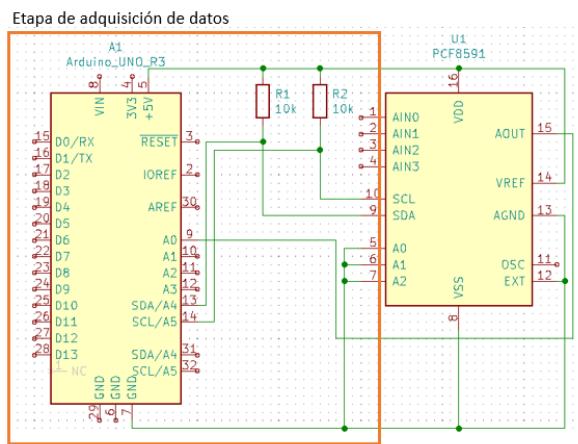


Figura 89. Etapa de adquisición de datos utilizada para la adquisición de datos del circuito generador de señal senoidal (2).

Para el circuito **Generador de Señal Senoidal**, se utilizó un microcontrolador como etapa de adquisición de datos. El valor en binario que envía el microcontrolador al PCF8591 es un número hexadecimal que puede variar entre 0x00 y 0xFF. El microcontrolador se comunica con el PCF8591 mediante I₂C y adquiere la señal analógica del circuito DDS (Pin 15 AOUT del PCF8591) en el pin A_0 .

6.3.2.2. Programa para adquirir señales del circuito DDS

El diagrama a bloques del programa completo es el siguiente:

Programa de Arduino para el Circuito Generador de Señal Senoidal
Nombre del programa: Arduino AR 65 DDS

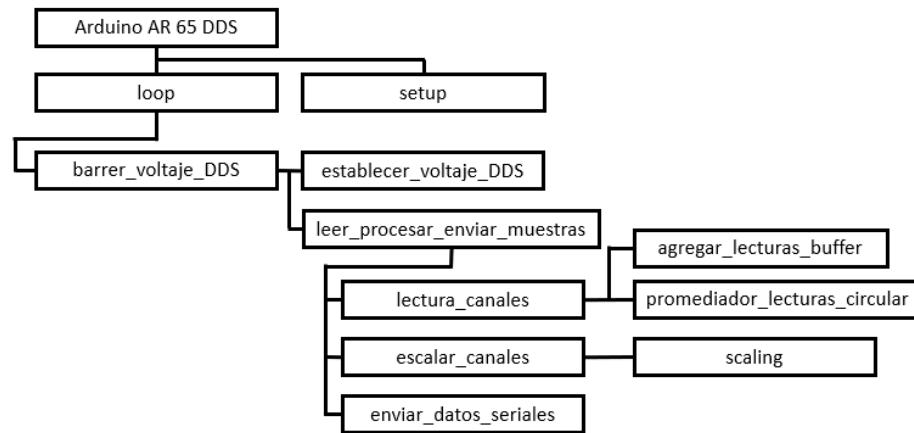


Figura 90. Diagrama a bloques del programa completo para adquirir señales del **circuito Generador de Señal Senoidal (b)**.

Sobre este diagrama, el programa se puede dividir en tres partes, como se indica en la figura 91:

Programa de Arduino para el Circuito Generador de Señal Senoidal
Nombre del programa: Arduino AR 65 DDS

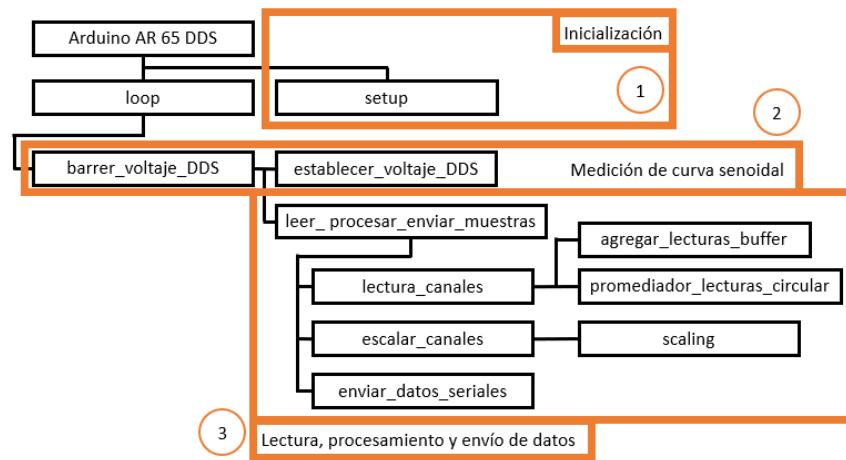


Figura 91. División del diagrama a bloques en tres partes principales.

A continuación, se presenta el funcionamiento del programa en bloques antes de pasar al código.

Nombre del programa: Arduino AR 65 DDS

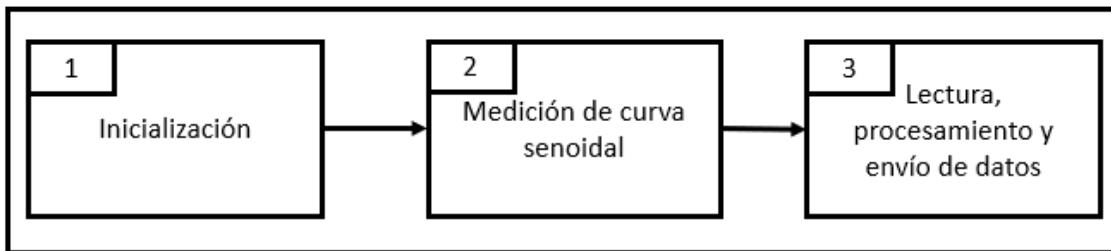


Figura 92. Diagrama a bloques del funcionamiento del programa para el circuito trazador de curvas.

El programa que se puede dividir en dos partes:

1. **Inicialización (1):** Inicia la comunicación serial con la computadora a través de un puerto USB e inicializa la comunicación I2C con los circuitos PCF8591.
2. **Medición de curva senoidal (2):** Manda valores en binario para el PCF8591 se actualice, de tal manera que se genere una señal senoidal a la salida. Para ello se actualiza el PCF8591 con los valores correspondientes para producir una señal senoidal indicados por programa.
3. **Lectura, procesamiento y envío de datos (3):** El microcontrolador el voltaje de salida del circuito DDS, lo procesa y lo envía a la computadora.

Para la **medición de curva senoidal (2)** se realiza un barrido de voltaje para generar la señal senoidal.

Para la parte de **lectura, procesamiento y envío de datos (3)** se realizan 3 funciones:

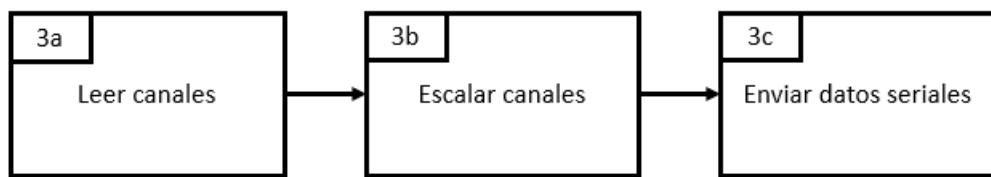


Figura 93. Diagrama a bloques de las funciones que realiza la parte de **lectura, procesamiento y envío de datos (3)**.

Para la señal senoidal medida, se realizan los siguientes pasos:

- 3a. **Leer canales:** Lee el voltaje de la señal senoidal. Es posible aplicar filtros a esta medición.
- 3b. **Escalar voltajes:** Escala este voltaje a el valor de la fuente de alimentación, en caso de que no entregue exactamente 5V.
- 3c. **Enviar datos seriales:** Envía este dato leído y procesado a la computadora.

El programa completo se presenta en la sección 10.2.2.1.

6.3.2.2.1. Subrutina de Inicialización (1)

Nombre del programa: Arduino AR 65 DDS

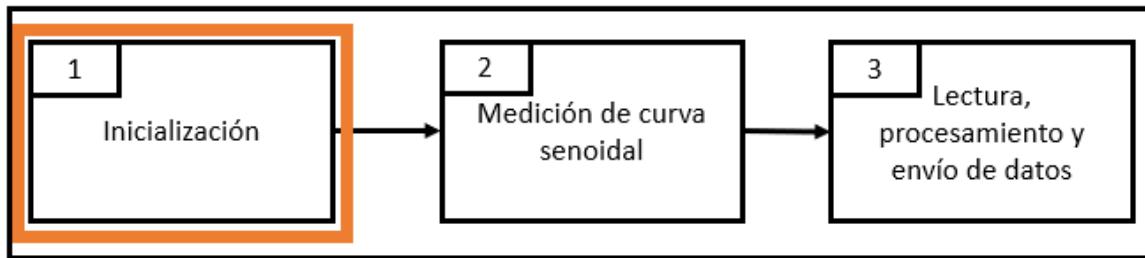


Figura 94. Ubicación del bloque de **Inicialización (1)**.

La subrutina de inicialización es idéntica a la mostrada en la sección 6.3.1.2.1 en la figura 64.

6.3.2.2.2. Subrutinas para la medición de curva senoidal (2)

Nombre del programa: Arduino AR 65 DDS

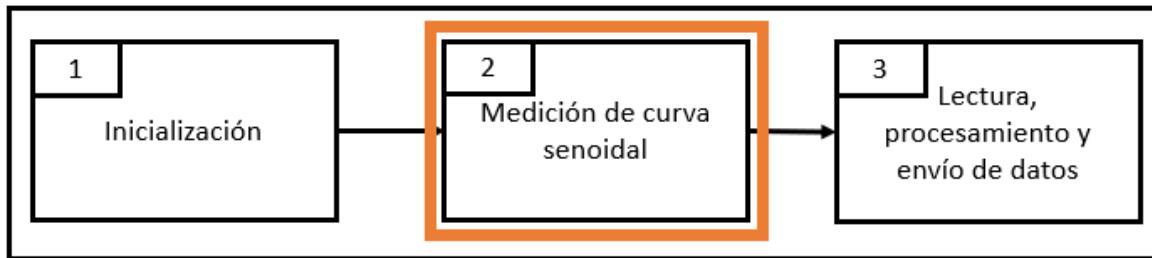


Figura 95. Ubicación del bloque Medición de curva senoidal (2).

```
/**  
 * Se mide al menos una curva para graficar la señal senoidal.  
 */  
  
void loop() {  
    // --- Toma muestras cada sampleTime ms ---  
    if (millis()-lastTime > sampleTime) {  
        lastTime = millis(); // Actualiza el tiempo de la ultima vez que se  
        tomaron muestras  
        barrer_voltaje_senoidal();  
    }  
}
```

Figura 96. Subrutina “loop”.

La subrutina que mide las curvas del circuito DDS primero verifican si ya es tiempo de tomar alguna muestra, después actualiza el PCF8591 del circuito DDS y toma la muestra al llamar a la función barrer_voltaje_DDS.

```
/**  
 * Obtiene el valor binario entre 0 y 255 a partir de la ecuación de la  
 * forma de onda senoidal  
 */  
  
void barrer_voltaje_senoidal(){  
    // --- Barre durante 50 muestras ---  
    for (int i_muestra=0; i_muestra<250; i_muestra+=5){  
        sine =  
byte(127.5*(1+sin((1000/sampleTime)*TWO_PI*i_muestra/(numMuestras-1))));  
        establecer_voltaje_senoidal(sine);  
  
        leer_procesar_enviar_muestras();  
    }  
}
```

Figura 97. Subrutina “barrer_voltaje_senoidal”.

Esta subrutina calcula el valor en binario (0 a 255) que se debe enviar al PCF8591, para ello se implementa la siguiente ecuación:

$$sine = 127.5 \left(1 + \sin \left(\frac{1000}{sampleTime} * \frac{2\pi * i_muestra}{numMuestras - 1} \right) \right)$$

En donde:

i_muestra es la variable iterativa que va desde 0 hasta 245 en incrementos de 5.

sampleTime es el tiempo en el que toma cada muestra en milisegundos.

numMuestras es la cantidad de muestras que hay por cada curva del circuito DDS.

```
/*
 * Establece el valor binario en el PCF8591. Este valor binario sera
convertido a un voltaje
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
 *
 */
void establecer_voltaje_senoidal(int valor){
    Wire.beginTransmission(PCF_DDS); // wake up PCF_Vcc
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}
```

Figura 98. Subrutina “establecer_voltaje_senoidal”.

El valor binario es convertido por el PCF8591, en donde:

Valor binario	Voltaje
0	0V
255	5V

Tabla 14. Voltaje generado a partir de un número binario.

Para los valores binarios entre 0 y 255, se genera un voltaje directamente proporcional a este valor binario.

$$V_{salida} = \frac{valor\ binario}{255} \cdot 5$$

Por ejemplo, para un valor binario de 100, se generaría un voltaje de salida de aproximadamente 1.96V.

6.3.2.2.3. Subrutinas para la lectura, procesamiento y envío de datos (3)

Nombre del programa: Arduino AR 65 DDS

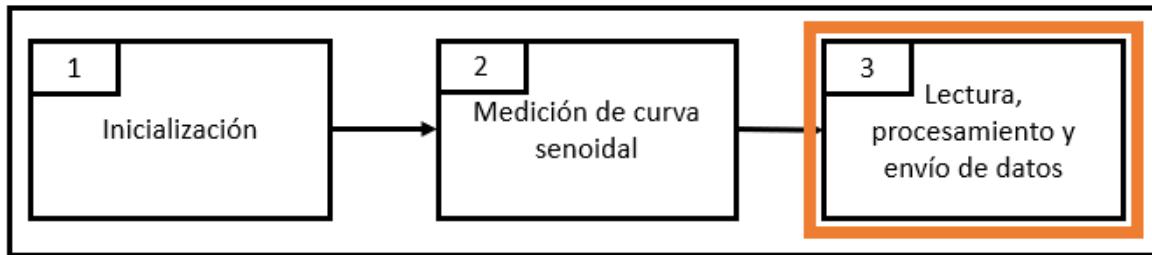


Figura 99. Ubicación del bloque **Lectura, procesamiento y envío de datos (3)**.

```

/**
 * Realiza la lectura, escalado y envio de datos por el puerto serial
 */
void leer_procesar_enviar_muestras(){
    // --- Lecura de los canales ---
    lectura_canales(); // Lee el voltaje senoidal del circuito

    // --- Escalamiento de canales ---
    escalar_canales(Vcc); // Escala el voltaje senoidal leido al voltaje
    de alimentacion Vcc

    // --- Transmision de cada valor al programa en Python ---
    enviar_datos_seriales(); // Envia el voltaje senoidal
}
  
```

Figura 100. Subrutina “leer_procesar_enviar_muestras”.

Esta subrutina agrupa tres subrutinas, que son ejecutadas después de haber actualizado el PCF8591. La siguiente tabla 15 realiza una comparación de las subrutinas agrupadas con el diagrama a bloques de la figura 93.

Diagrama a bloques de la parte Lectura, procesamiento y envío de datos (4)	Código
Lectura de canales (4a)	lectura_canales();
Escalar_canales (4b)	escalar_canales(Vcc);
Enviar datos seriales (4d)	enviar_datos_seriales();

Tabla 15. Comparación de las funciones que realiza la parte del programa **Lectura, procesamiento y envío de datos (3)** con su diagrama a bloques.

6.3.2.2.3.1 Subrutina para leer los canales (3a)

Nombre del programa: Arduino AR 65 DDS

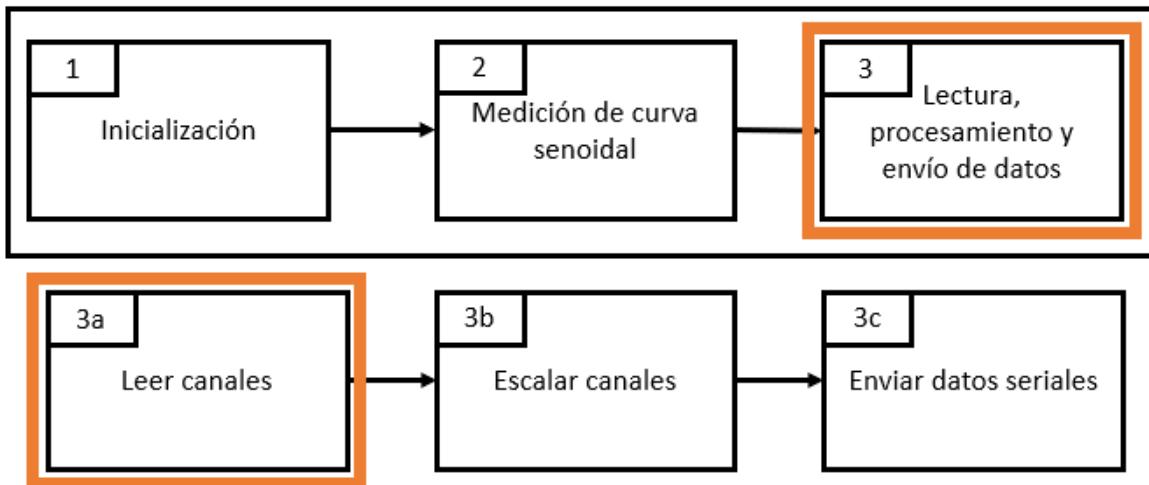


Figura 101. Ubicación del bloque **Leer canales(3a)**.

```
/**  
 * Lee el voltaje senoidal del circuito.  
 */  
  
void lectura_canales()  
{  
    canales_medidos[0] = analogRead(PinDDS);  
    canales_medidos[1] = sine;  
  
    if (Filtrado_Circular)  
    {  
        // --- CON FILTRADO CIRCULAR con buffer circular de 32 muestras ---  
        for (int b=0; b<bufferSize; b++)  
        {  
            // Filtrado circular en cada lectura. SIN mezclar muestras en  
            // tiempos diferentes  
            agregar_lecturas_buffer(canales_medidos);  
            promediador_lecturas_circular(canales_medidos);  
        }  
    } else {  
        // --- SIN FILTRADO CIRCULAR ---  
        for(int j=0; j<numero_entradas_analogicas; j++)  
        {  
            canales[j] = canales_medidos[j];  
        }  
    }  
}
```

Figura 102. Subrutina “lectura_canales”.

Se leen dos variables:

1. La salida analógica del pin 15 AOUT del PCF8591 conectado al pin A_0 del microcontrolador.
2. El valor en binario sine que se envía al PCF8591.

La siguiente tabla 16 compara las líneas de código que leen estas dos variables:

Variable leída	Código
Voltaje analógico en el pin A_0	canales_medidos[0] = analogRead(PinDDS);
Valor binario	canales_medidos[1] = sine;

Tabla 16. Comparación de la lectura de variables con el código

6.3.2.2.3.2 Subrutina para escalar los canales (3b)

Nombre del programa: Arduino AR 65 DDS

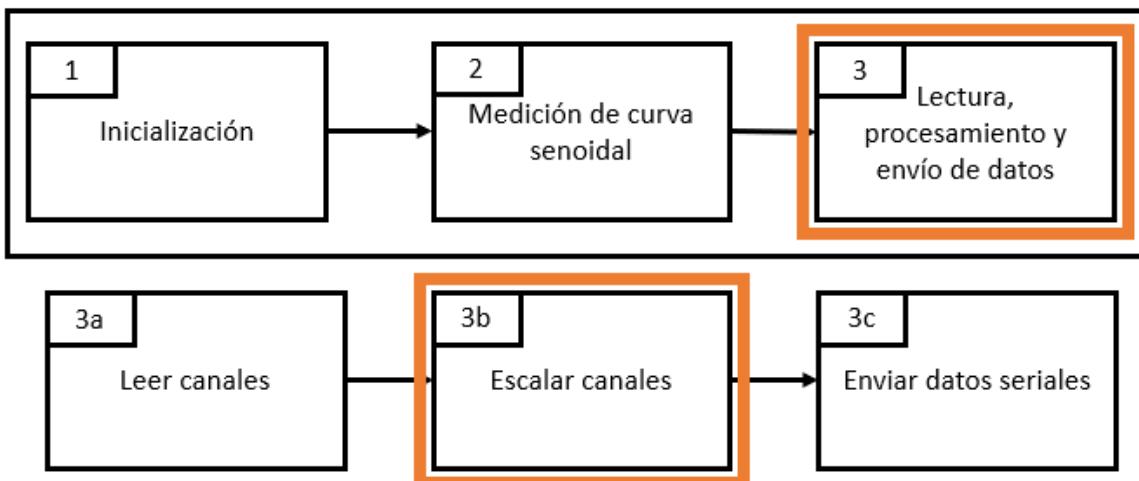


Figura 103. Ubicación del bloque **Escalar canales (3b)**.

```
/**  
 * Escala el canal de voltaje senoidal, que varia entre [0,1023] a un  
 * numero que representa el voltaje [0,5]  
 */  
void escalar_canales(double Vcc_ref)  
{  
    voltajes[0] = scaling(canales[0], 0, 1023, 0, Vcc_ref);  
}
```

Figura 104. Subrutina “escalar_canales”.

En esta subrutina se hace un escalado del voltaje analógico que se lee en el pin A_0 del microcontrolador.

Esta subrutina hace uso de una subrutina de escalado que es la misma que la de la sección 6.3.1.2.5.2 en la figura 81.

6.3.2.2.3.3 Subrutina para enviar datos seriales (3c)

Nombre del programa: Arduino AR 65 DDS

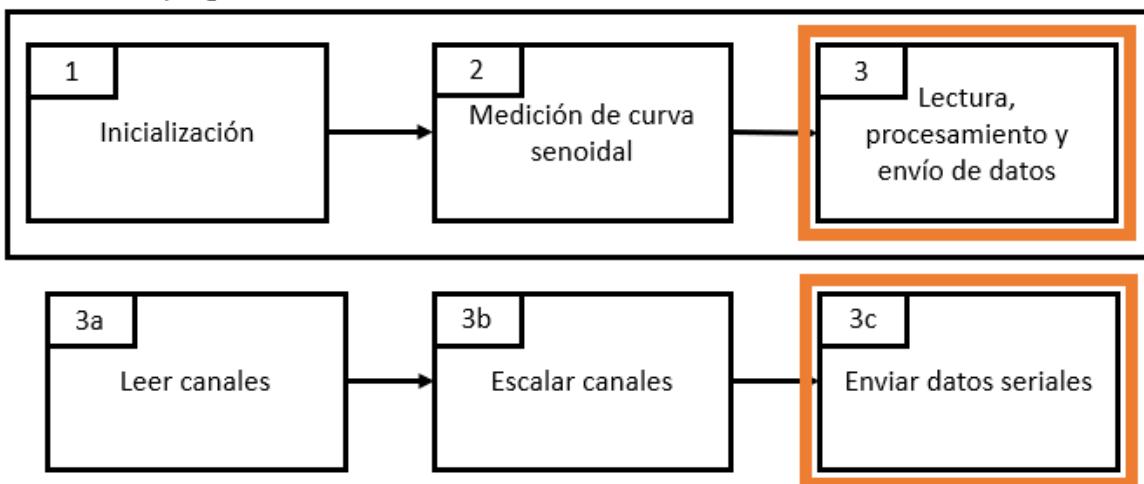


Figura 105. Ubicación del bloque **Enviar datos seriales (3c)**.

```
/**  
 * Envía todos los datos por el puerto serial  
 * En orden, se envia:  
 *   1. Voltaje de la señal senoidal  
 *   2. Número en binario que genera ese voltaje  
 */  
void enviar_datos_seriales()  
{  
    Serial.println(voltajes[0]); // Voltaje  
    Serial.println(canales[1]); // Valor binario  
}
```

Figura 106. Subrutina “enviar_datos_seriales”.

6.3.2.2.4 Subrutinas de filtrado

Programa de Arduino para el Circuito Generador de Señal Senoidal
Nombre del programa: Arduino AR 65 DDS

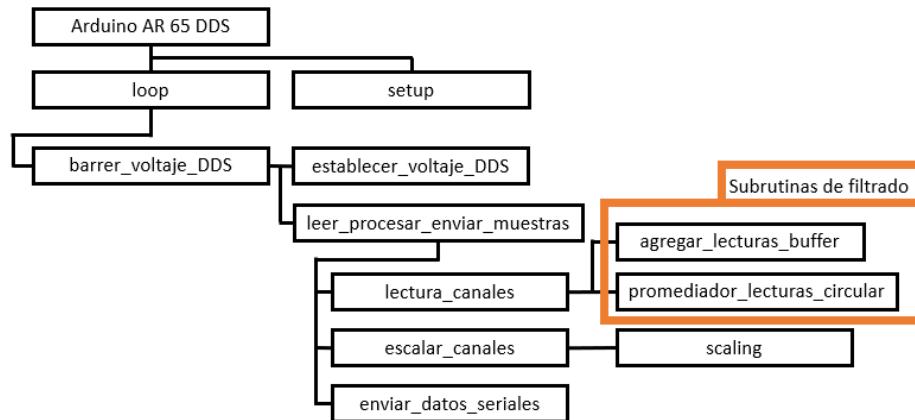


Figura 107. Ubicación de las **subrutinas de filtrado**.

Las **subrutinas de filtrado** son las mismas que las presentadas en la sección 6.3.1.2.6 en las figuras 87 y 88.

6.3.3. Amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Círculo secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.3.3.1. Etapa de adquisición de datos utilizada para el circuito Amplificador de una etapa

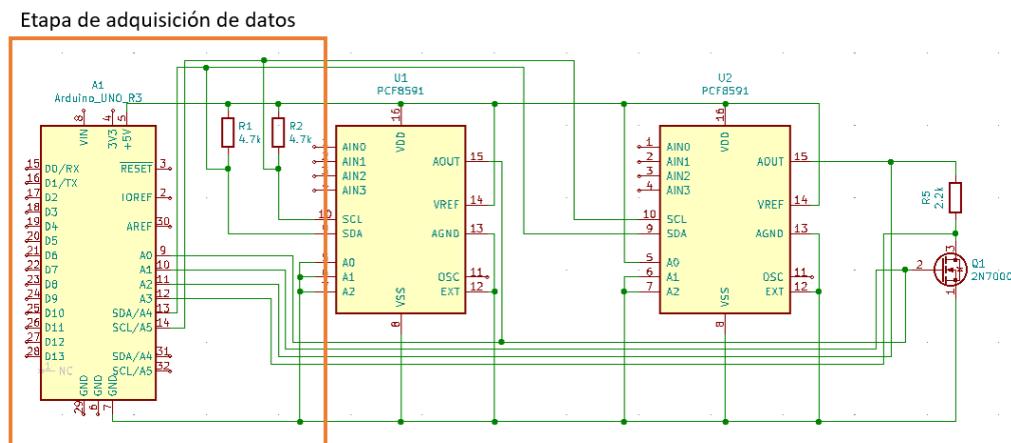


Figura 108. Etapa de adquisición de datos utilizada para la adquisición de datos del circuito Amplificador de una etapa (3).

Para el circuito amplificador de una etapa se utilizó el mismo circuito descrito en la sección 6.3.1.1 en la figura 56, que es el presentado en la figura 108.

6.3.3.2. Programa para adquirir señales del amplificador de una etapa

El diagrama a bloques del programa completo es el siguiente:

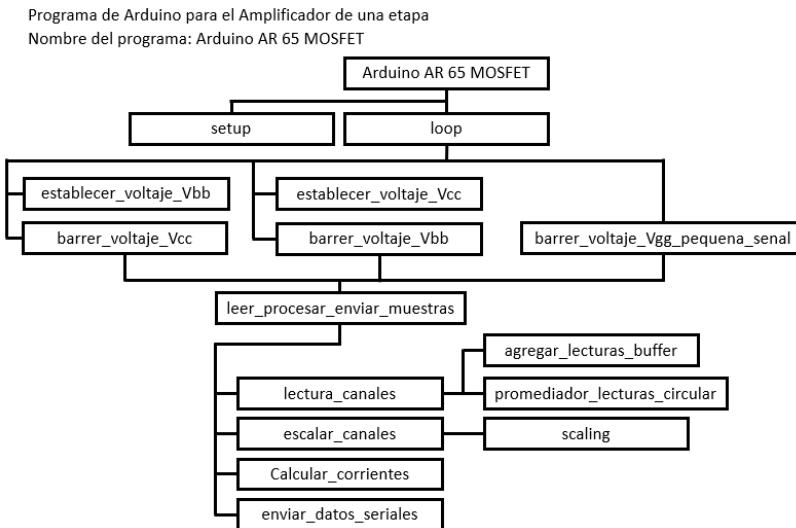


Figura 109. Diagrama a bloques del programa completo en Arduino para adquirir señales del circuito **Amplificador de una Etapa (c)**.

Sobre este diagrama, el programa se puede dividir en cinco partes, como se indica en la figura 110:

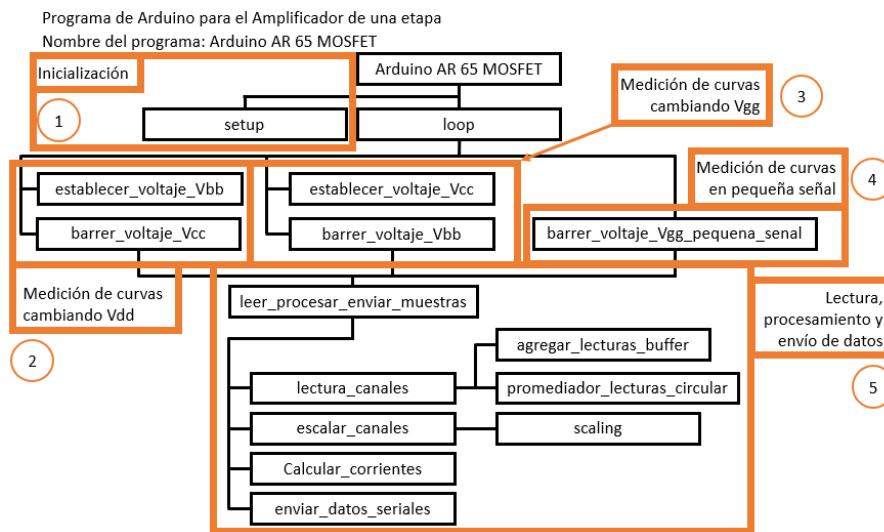


Figura 110. División del diagrama a bloques en cinco partes principales.

Nombre del programa: Arduino AR 65 MOSFET

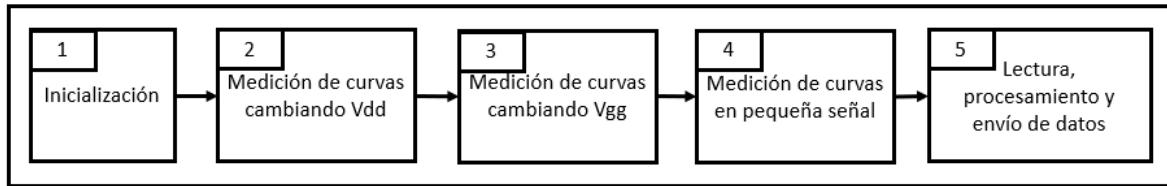


Figura 111. Diagrama a bloques del funcionamiento para el circuito **Amplificador de una etapa (c)**.

El programa que se puede dividir en tres partes:

1. **Inicialización:** Inicia la comunicación serial con la computadora a través de un puerto USB e inicializa la comunicación I2C con los circuitos PCF8591.
2. **Medición de curvas cambiando V_{dd} (2):** El microcontrolador envía valores en binario al PCF8591 que controla el voltaje V_{dd} para barrer este voltaje de 0 a 5V, mientras fija el PCF que controla el voltaje V_{gg} . Con estas mediciones es posible obtener las curvas I_d vs V_{ds} .
3. **Medición de curvas cambiando V_{gg} (3):** El microcontrolador envía valores en binario al PCF8591 que controla el voltaje V_{gg} para barrer este voltaje, mientras fija el PCF8591 que controla el voltaje V_{dd} . Con estas mediciones es posible obtener las demás curvas.
4. **Medición de curvas en pequeña señal (4):** El microcontrolador envía valores en binario al PCF8591 que controla el voltaje V_{gg} para barrer este voltaje en forma de una senoidal de pequeña amplitud con una amplitud pico-pico de alrededor 150mV, mientras fija el PCF8591 que controla el voltaje V_{dd} .
5. **Lectura, procesamiento y envío de datos (5):** Para cada parte **Medición de curvas cambiando V_{dd} (2)**, **Medición de curvas cambiando V_{gg} (3)** y **Medición de curvas en pequeña señal (4)** el microcontrolador lee los diferentes voltajes del circuito Amplificador de una Etapa, los procesa y los envía a la computadora.

Las partes de **Medición de curvas cambiando V_{dd} (2)** y **Medición de curvas cambiando V_{gg} (3)** realizan las mismas funciones que las descritas en la sección 6.3.1.2 en las figuras 60 y 61.

La parte de **Medición de curvas en pequeña señal (4)** realiza las siguientes funciones:

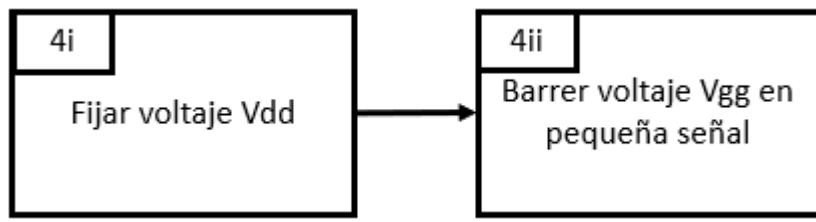


Figura 112. Diagrama a bloques de las funciones que realiza la parte del programa **Medición de curvas en pequeña señal (4)**.

- 4i. **Fijar el voltaje V_{dd} (3ii):** Se actualiza el valor de V_{dd} al enviar el número binario correspondiente al PCF8591 que controla este voltaje.
- 4ii. **Barrer voltaje V_{gg} (3iii):** Se actualiza el voltaje V_{gg} con la forma de onda de una señal senoidal con un voltaje de offset y una amplitud pequeña de 150mV pico-pico. Se envían números binarios al PCF8591 del voltaje V_{gg} . Cada vez que se actualiza, se toman muestras de los diferentes voltajes de la respuesta del transistor.

El programa completo se presenta en la sección 10.2.3.1.

6.3.3.2.1. Subrutina de Inicialización (1)

Nombre del programa: Arduino AR 65 MOSFET

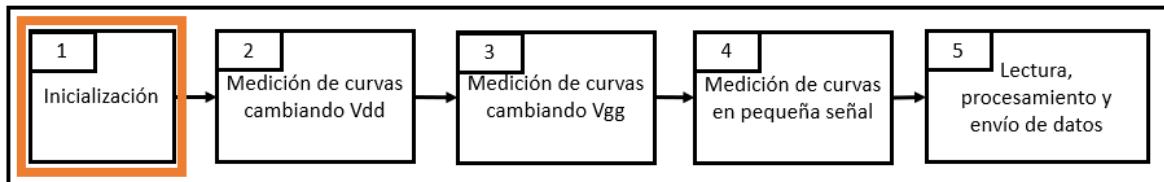


Figura 113. Ubicación del bloque de **Inicialización (1)**.

La subrutina de inicialización es la misma que la descrita en la sección 6.3.1.2.1 en la figura 64.

6.3.3.2.2. Subrutina principal loop (2,3,4,5)

Nombre del programa: Arduino AR 65 MOSFET



Figura 114. Ubicación de los bloques **Medición de curvas cambiando Vdd**, **Medición de curvas cambiando Vgg** y **Medición de curvas en pequeña señal** (4).

```

/*
 * Se miden cinco curvas para graficar Id vs vds. Posteriormente se toma
una curva para id vs vgs y una ultima curva para el amplificador en
pequeña señal
 */
void loop() {
// --- Toma muestras cada sampleTime ms ---
if (millis()-lastTime > sampleTime) {

    lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
tomaron muestras

    if(numero_curva<num_curvas_id_vds) // ¿Se toman muestras de las
primeras "num_curvas_id_vds" curvas para Id vs Vce?
    {
        int valor;
        if (Transistor)
        {
            // --- 5 Curvas para Ic vs Vce [BJT] ---
            valor = map(numero_curva,0,5,50,250);
        } else {
            // --- 5 Curvas para Id vs Vds [MOSFET] ---
            valor = map(numero_curva,0,5,102,120); //116 - 132 a 220 ohm
        }

        establecer_voltaje_Vbb(valor);
        barrer_voltaje_Vcc();
    }
    else if(numero_curva<num_curvas_id_vds+num_curvas_parametros)
    {
        // --- 1 Curva para Ib vs Vbe , Ic vs Vbe , entre otras ---
        // --- 1 Curva para Ig vs Vgs , Id vs Vgs ---
        establecer_voltaje_Vcc(255);
        barrer_voltaje_Vbb();
    }
    else
if(numero_curva<num_curvas_id_vds+num_curvas_parametros+num_curvas_id_vds
+num_curvas_pequena_senal)
    {

```

```

    // --- 1 Curva para pequena senal ---
    establecer_voltaje_Vcc(255);
    barrer_voltaje_Vbb_pequena_senal(2.2,0.075);
}
else
{
    numero_curva = 0;
}
}
}

```

Figura 115. Subrutina principal “loop”.

Primero se verifica que ha pasado el tiempo para tomar una muestra. Después se toman mediciones para 5 curvas utilizando el bloque **Medición de curvas cambiando el voltaje V_{dd} (2)** y después se toman mediciones para 1 curva utilizando el bloque **Medición de curvas cambiando el voltaje V_{gg} (3)**. Por último, se toma 1 curva utilizando el bloque **Medición de curvas en pequeña señal (4)**.

Comparando el código de la figura 115 con el diagrama para la parte de **Medición de curvas cambiando el voltaje V_{dd} (2)** de la figura 60, cada una de las siguientes tres líneas de código corresponde a una función del bloque.

Diagrama a bloques de la parte Medición de curvas cambiando el voltaje V_{dd} (2)	Código
Calcular rango de voltaje V_{gg} (2i)	valor = map(numero_curva,0,5,102,122); //116 - 132 anterior
Fijar el voltaje V_{gg} (2ii)	establecer_voltaje_Vgg(valor);
Barrer voltaje V_{dd} (2iii)	barrer_voltaje_Vdd();

Tabla 17. Comparación de las funciones que realiza la parte del programa **Medición de curvas cambiando el voltaje V_{dd} (2)** con su diagrama a bloques.

Comparando el código de la figura 115 con el diagrama para la parte de **Medición de curvas cambiando el voltaje V_{gg} (3)** de la figura 61, cada una de las siguientes dos líneas de código corresponde a una función del bloque.

Diagrama a bloques de la parte Medición de curvas cambiando el voltaje V_{gg} (3)	Código
Fijar el voltaje V_{dd} (3i)	establecer_voltaje_Vdd(255);
Barrer voltaje V_{gg} (3ii)	barrer_voltaje_Vgg();

Tabla 18. Comparación de las funciones que realiza la parte del programa **Medición de curvas cambiando el voltaje V_{gg} (3)** con su diagrama a bloques.

Comparando el código de la figura 115 con el diagrama para la parte de **Medición de curvas en pequeña señal (4)** de la figura 112, cada una de las siguientes dos líneas de código corresponde a una función del bloque.

Diagrama a bloques de la parte Medición de curvas en pequeña señal (4)	Código
Fijar el voltaje V_{dd} (4i)	establecer_voltaje_Vdd(255);
Barrer voltaje V_{gg} en pequeña señal (4ii)	barrer_voltaje_Vgg_pequena_senal(2.4, 0.075);

Tabla 19. Comparación de las funciones que realiza la parte del programa **Medición de curvas en pequeña señal (4)** con su diagrama a bloques.

En la subrutina para barrer voltaje V_{gg} , se actualiza con una señal senoidal con 2.2V de offset y 150mV de amplitud pico-pico.

6.3.3.2.3. Subrutinas para la medición de curvas barriendo V_{gg} (2)

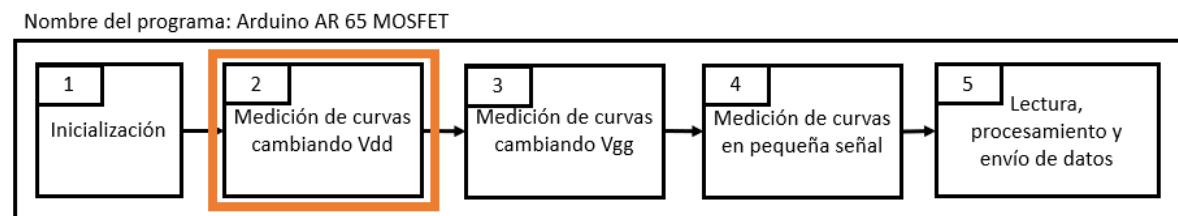


Figura 116. Ubicación del bloque **Medición de curvas cambiando Vdd (2)**.

Las subrutinas para la parte de **Medición de curvas cambiando Vdd (2)** son las mismas que la descrita en la sección 6.2.1.2.3, en las figuras 68 y 70.

6.3.3.2.4. Subrutinas para la medición de curvas barriendo Vdd (3)

Nombre del programa: Arduino AR 65 MOSFET

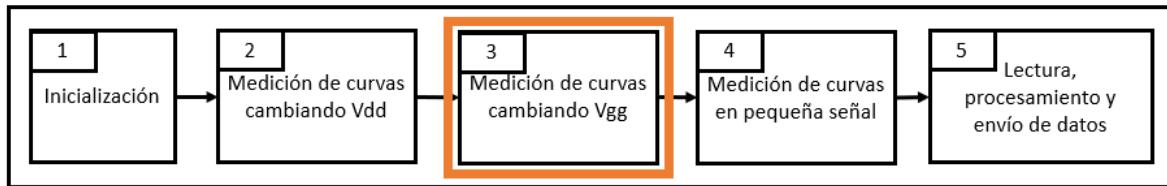


Figura 117. Ubicación del bloque **Medición de curvas cambiando Vgg (3)**.

Las subrutinas para la parte de **Medición de curvas cambiando Vgg (3)** son las mismas que la descrita en la sección 6.2.1.2.4, en las figuras 72 y 74.

6.3.3.2.5. Subrutinas para la medición de curvas en pequeña señal (4)

Nombre del programa: Arduino AR 65 MOSFET

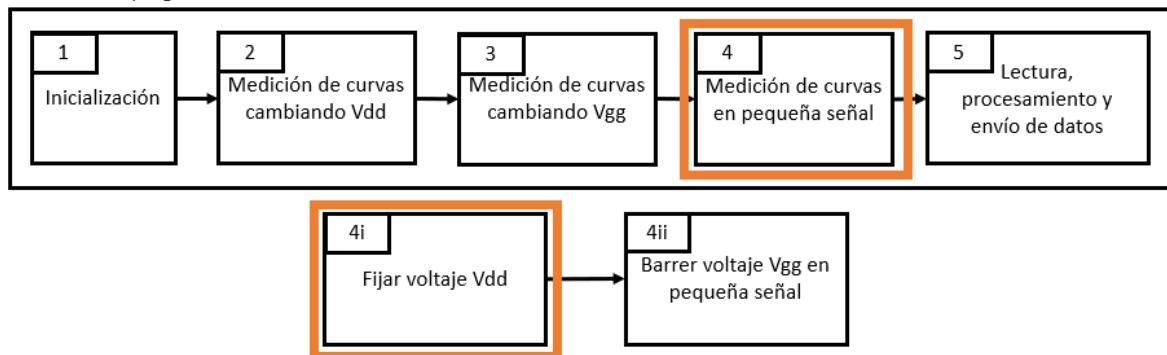


Figura 118. Ubicación del bloque **Fijar voltaje Vdd (4i)**.

Este bloque es el mismo que el descrito en la sección 6.3.1.2.4 en la figura 72.

Nombre del programa: Arduino AR 65 MOSFET

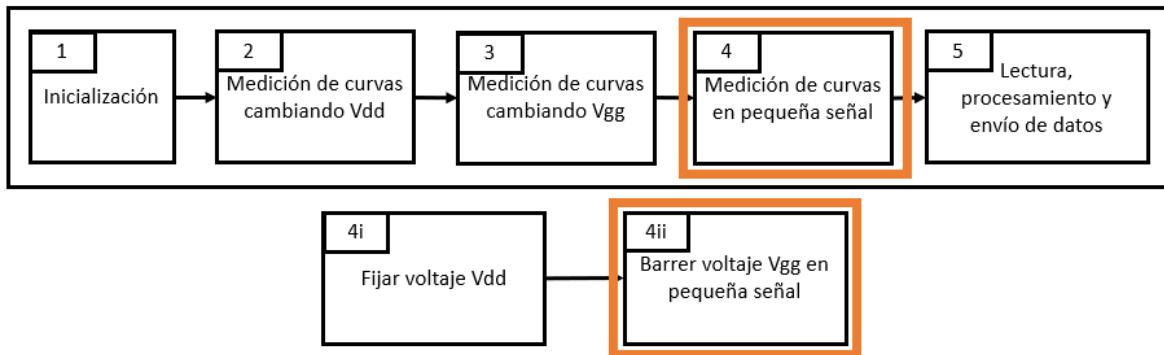


Figura 119. Ubicación del bloque **Barrer voltaje Vgg en pequeña señal (4ii)**.

```

/**
 * Actualiza Vgg con diferentes valores de una senoidal con amplitud
pequeña
 * @param Voffset -> Es el voltaje de offset de la señal senoidal
 * @param Amplitud_pico -> Es el valor de amplitud pico de la senoidal
 *
 */
void barrer_voltaje_Vgg_pequena_senal(double Voffset, double
Amplitud_pico){
    for (int i_muestra=0; i_muestra<250; i_muestra+=5) {
        sine =
byte(255*((Voffset/Vdd)+((Amplitud_pico/(Vdd))*sin((1000/sampleTime)*TWO_
PI*i_muestra/(numMuestras-1)))));
        establecer_voltaje_Vgg(sine);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

```

Figura 120. Subrutina “barrer_voltaje_Vgg_pequena_senal”.

Esta función calcula un número binario correspondiente a la amplitud de la forma de onda de una señal senoidal. Esta señal tiene una amplitud pequeña y contiene una componente de offset. Implementa la siguiente ecuación:

$$sine = byte \left(255 * \left(\frac{Voffset}{Vcc} + \left(\frac{Amplitud_pico}{Vcc} \right) \sin \left(\frac{1000}{sampleTime} * \frac{2\pi * i_muestra}{numMuestras - 1} \right) \right) \right)$$

Donde:

Voffset es el voltaje deseado de offset de la señal.

Amplitud_pico es la amplitud deseada de la senoidal.

Vcc es el valor medido con un multímetro de la fuente de voltaje que entrega el microcontrolador.

i_muestra es la variable iterativa que va desde 0 hasta 250 en incrementos de 5.

sampleTime es el tiempo en el que toma cada muestra en milisegundos.

numMuestras es la cantidad de muestras que hay por cada curva del circuito DDS.

Puesto que el valor de toda la ecuación es convertido a un tipo de dato byte, hay 256 valores binarios para la amplitud de la señal senoidal.

6.3.3.2.6. Subrutinas para la lectura, procesamiento y envío de las muestras leídas (5)

Nombre del programa: Arduino AR 65 MOSFET

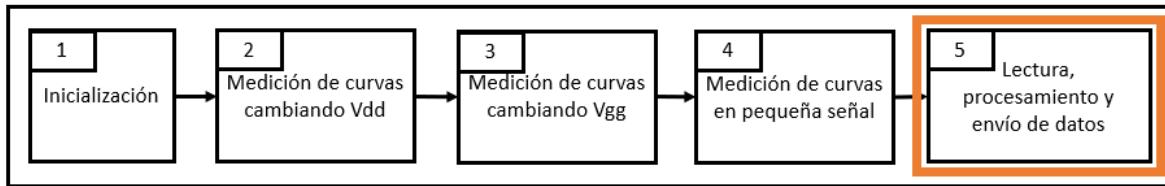


Figura 121. Ubicación del bloque **Lectura, procesamiento y envío de datos (5)**.

Las subrutinas para la parte de **Lectura, procesamiento y envío de datos (5)** son las mismas que las descritas en la sección 6.2.1.2.5, en las figuras 76, 78, 80, 81, 83 y 85.

6.3.3.2.7 Subrutinas de filtrado

Programa de Arduino para el Amplificador de una etapa

Nombre del programa: Arduino AR 65 MOSFET

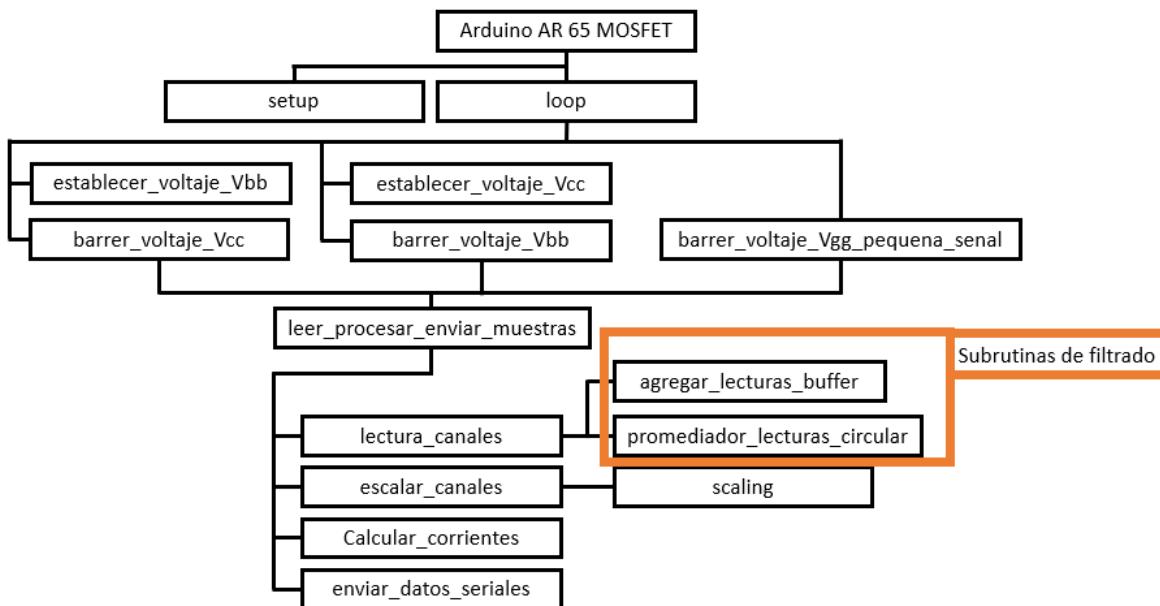


Figura 122. Ubicación de las **subrutinas de filtrado**.

Las **subrutinas de filtrado** son las mismas que las presentadas en la sección 6.3.1.2.6 en las figuras 87 y 88.

6.3.4. Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
 - b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
 - c. **Amplificador de una etapa (c)** con un transistor MOSFET.
 - d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.3.4.1 Etapa de adquisición de datos utilizada para el circuito secuencial

Para ambos circuitos, ya sea el **circuito secuencial implementado por hardware (i)** o el **circuito secuencial implementado por software (ii)**, se utiliza la misma etapa de adquisición de datos y el mismo programa para adquirir las señales del **circuito secuencial (d)**.

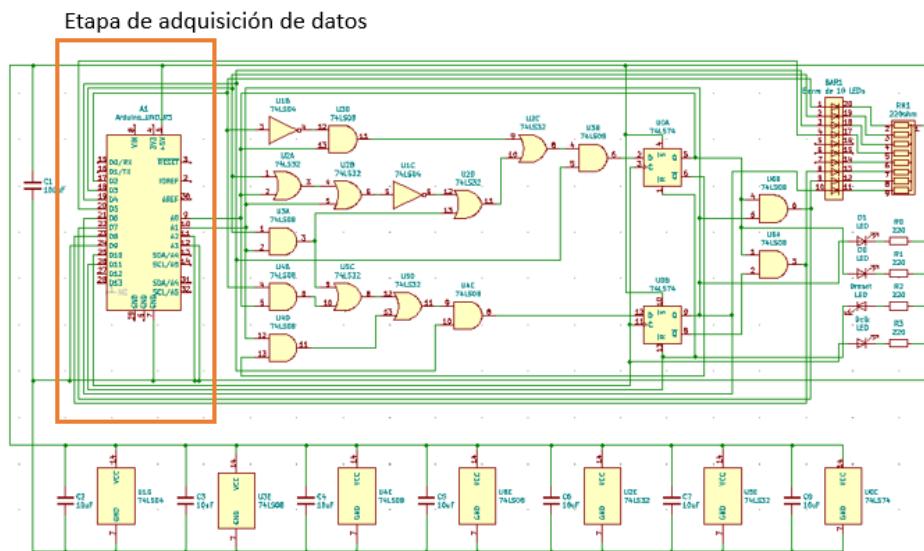


Figura 123. Etapa de adquisición de datos para el circuito secuencial (d) implementado por hardware (i).

Etapa de adquisición de datos

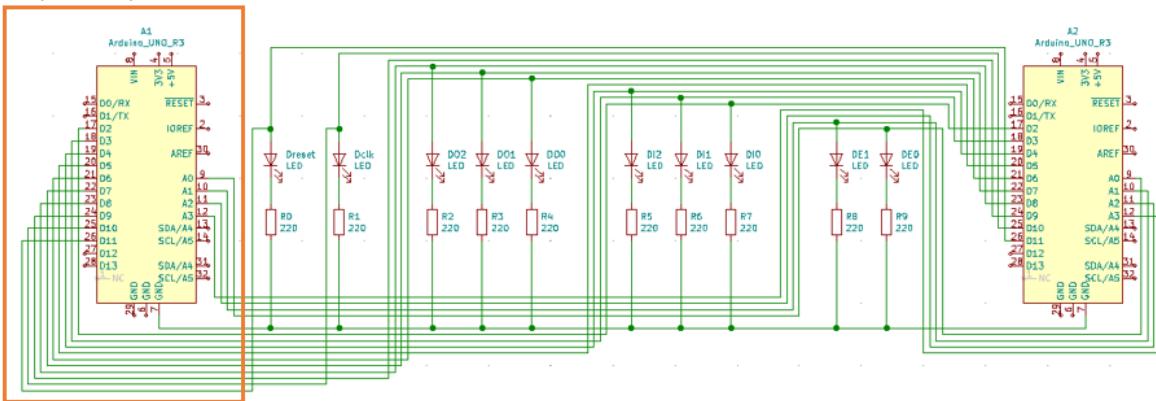


Figura 124. Etapa de adquisición de datos para el **circuito secuencial (d)** implementado por software (ii).

El microcontrolador de la etapa de adquisición de datos realiza lo siguiente:

- En los pines D2, D3 y D4 **escribe la entrada** de la máquina de estados finitos.
- En los pines D6, D7 y D8 **lee la salida** de la máquina de estados finitos.
- En los pines A0 y A0 **lee el estado** de la máquina de estados finitos.
- En los pines D10 y D11 **escribe** respectivamente la **señal de reloj** y de **señal dero**set a la máquina de estados finitos.

6.3.4.2. Programa para adquirir señales del circuito secuencial

El diagrama a bloques del programa completo es el siguiente:

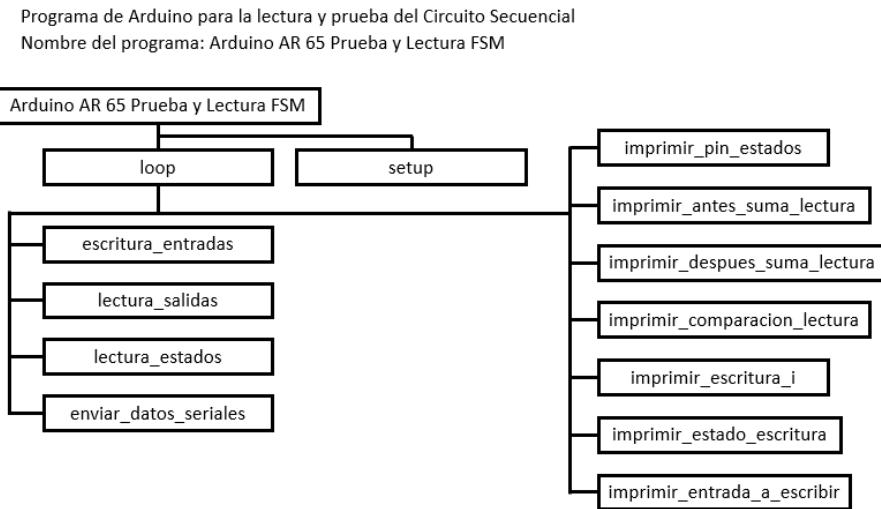


Figura 125. Diagrama a bloques del programa completo en Arduino para adquirir señales del **Circuito secuencial (d)**.

Sobre este diagrama, el programa se puede dividir en tres partes, como se indica en la figura 126:

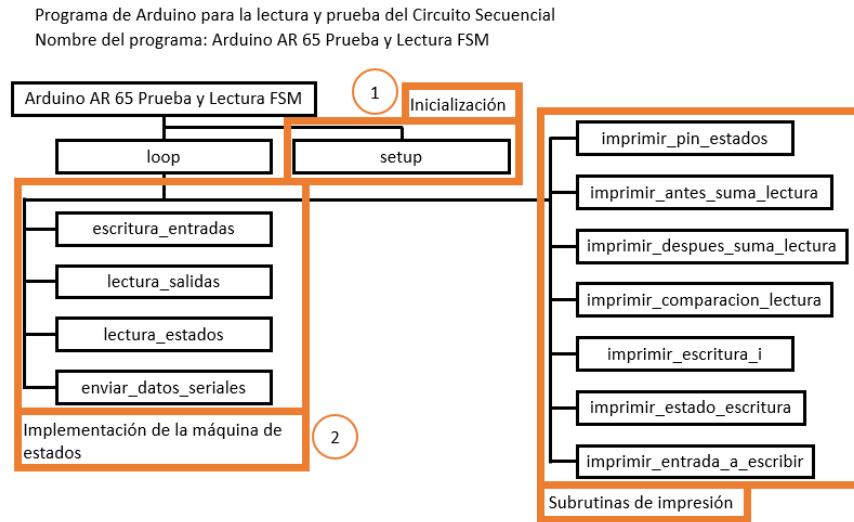


Figura 126. División del diagrama a bloques en dos partes principales.

Para poder adquirir los datos de la máquina de estados, el programa correspondiente realiza las siguientes funciones:

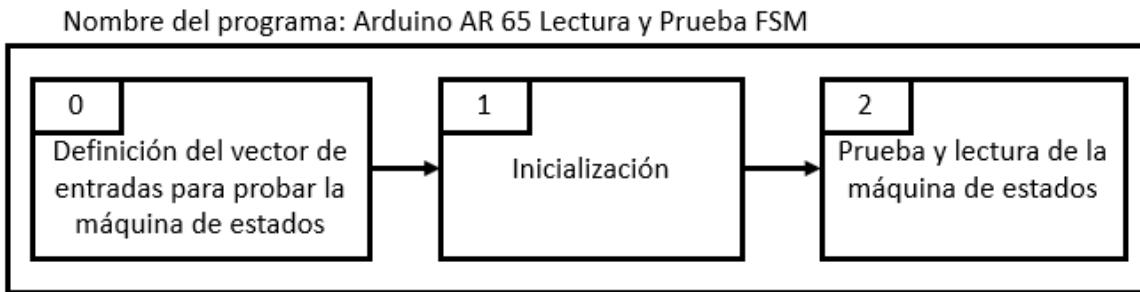


Figura 127. Diagrama a bloques del funcionamiento del programa para probar y leer la máquina de estados finitos.

Para que pueda funcionar este programa, se realiza la **Definición del vector de entradas para probar la máquina de estados (0)**, en la cual se describe una serie de valores digitales binarios que se enviarán como entrada a la máquina de estados.

El programa se puede dividir en dos partes:

1. **Inicialización (1):** Inicializa los pines de entrada, salida, estados y la comunicación serial.
2. **Prueba y lectura de la máquina de estados (2):** Envía valores binarios de entrada a la máquina de estados finitos y lee tanto la salida como el estado de la misma.

La subrutina de **prueba y lectura de la máquina de estados (2)** realiza las siguientes funciones:

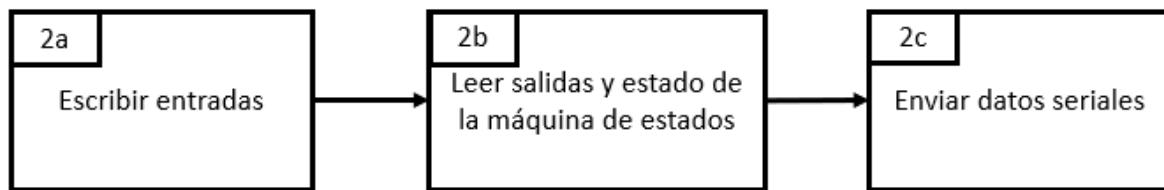


Figura 128. Diagrama a bloques de las funciones que realiza la parte del programa **Prueba y lectura de la máquina de estados (2)**.

- 2a. **Escribir de entradas (2a):** El microcontrolador de la etapa de adquisición de datos envía valores digitales binarios a la entrada de la máquina de estados finitos.
- 2b. **Leer salidas y estado de la máquina de estados (2b):** El microcontrolador de la etapa de adquisición de datos recibe la salida y el estado en el que se encuentra la máquina de estados.
- 2c. **Enviar datos seriales (2c):** El microcontrolador de la etapa de adquisición de datos envía los datos leídos y procesados a la computadora.

El programa completo se presenta en la sección 10.2.4.2.

6.3.4.2.1 Definición del vector de entradas para probar la máquina de estados (0)

```
// Ejemplo de un control para proteccion contra corto circuito para
// fuente de alimentacion
/**
 * tabla_vectores_escritura_prueba[] -> contiene las entradas a escribir
 * a la maquina de estados finitos
 */
const int tabla_vectores_escritura_prueba[] =
{0B000,0B100,0B101,0B111,0B101,0B100,0B110,0B100,0B100,0B000};
```

Figura 129. Fragmento de código que define al vector de entradas para probar la máquina de estados.

Esta tabla contiene los valores de entrada que se le aplicarán al circuito con flip flops que implementa el circuito secuencial. Están en formato binario, iniciando con el prefijo “0B” seguidos de un número binario de 3 dígitos.

Según la definición de la máquina de estados en la sección 5.4.2, cada uno de estos valores significa:

Entrada $I_2 I_1 I_0$	Significado			Salida esperada $O_2 O_1 O_0$
	Botón de alimentación Entrada I_2	Botón de reset Entrada I_1	Sensor de cortocircuito Entrada I_0	
0B000	No presionado	No presionado	Sin cortocircuito	0B000
0B100	Presionado	No presionado	Sin cortocircuito	0B100
0B101	Presionado	No presionado	Cortocircuito	0B001
0B111	Presionado	Presionado	Cortocircuito	0B011
0B101	Presionado	No presionado	Cortocircuito	0B001
0B100	Presionado	No presionado	Sin cortocircuito	0B001
0B110	Presionado	Presionado	Sin cortocircuito	0B011
0B100	Presionado	No presionado	Sin cortocircuito	0B100
0B100	Presionado	No presionado	Sin cortocircuito	0B100
0B000	No presionado	No presionado	Sin cortocircuito	0B000

Tabla 20. Significado de los valores binarios de entrada del vector de entradas para probar la máquina de estados.

Con estos valores se deben de observar las salidas de la máquina de estados:

1. La **fuente de alimentación (O_2)** funciona cuando el **botón de la fuente de alimentación (I_2)** está prendido y cuando no hay **corto circuito (I_0)**.
2. De lo contrario, deshabilita la salida del **relevador de alimentación (O_2)** al haber un **corto circuito (I_0)**.
3. Si sigue habiendo **corto circuito (I_0)** incluso con el **botón de reset (I_1)** presionado, no se habilita el **relevador de alimentación (O_2)**.
4. Se tiene que remover el **corto circuito (I_0)** para después presionar el **botón de reset (I_1)** y ahora sí se habilita el **relevador de alimentación (O_2)**.
5. La **fuente de alimentación (O_2)** deja de funcionar cuando se deja de activar el **botón de la fuente de alimentación (I_2)**.

6.3.4.2.2 Subrutina de inicialización (1)

Nombre del programa: Arduino AR 65 Lectura y Prueba FSM

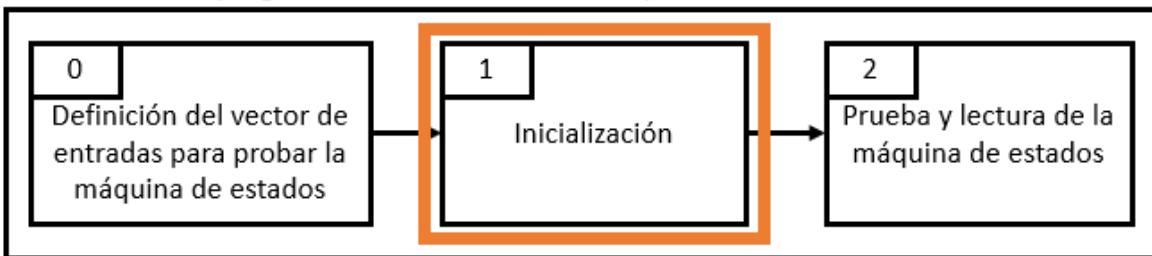


Figura 130. Ubicación del bloque de **Inicialización (1)**.

```
/**  
 * Inicializa todos los pines a utilizar, inicializa el reloj, la  
frecuencia de operacion, la comunicacion serial y la comunicacion I2C  
para comunicarse con los dos PCF8591  
* Para cambiar la frecuenacia del PWM de pines en Arduino Mega:  
* https://forum.arduino.cc/t/mega-2560-pwm-frequency/71434/2  
*/  
void setup() {  
  
    // Se limpian los registros que definen la frecuencia de los pines PWM  
    int myEraser = 7;  
    TCCR0B &= ~myEraser;  
    TCCR1B &= ~myEraser;  
    TCCR2B &= ~myEraser;  
    TCCR3B &= ~myEraser;  
    TCCR4B &= ~myEraser;  
  
    // Se configuran los registros para la frecuencia de los pines PWM  
    int myPrescaler = 3;  
    TCCR0B |= myPrescaler;  
    TCCR1B |= myPrescaler;  
    TCCR2B |= myPrescaler;  
    TCCR3B |= myPrescaler;  
    TCCR4B |= myPrescaler;  
  
    // Inicializa los pines en donde se leera el estado de la maquina  
    for(int i=0;i<sizeof(pin_estados)/sizeof(pin_estados[0]);i++)  
    {  
        pinMode(pin_estados[i], INPUT);  
    }  
  
    // Inicializa los pines en donde se escribiran las entradas de la  
maquina  
    for(int i=0;i<sizeof(pin_escrituras)/sizeof(pin_escrituras[0]);i++)  
    {  
        pinMode(pin_escrituras[i], OUTPUT);  
    }  
}
```

```

}

// Inicializa los pines en donde se leeran las salidas de la maquina
for(int i=0;i<sizeof(pin_lecturas)/sizeof(pin_lecturas[0]);i++)
{
    pinMode(pin_lecturas[i],INPUT);
}

// Se inicializan los pines del reloj y de reset
pinMode(pinCLK,OUTPUT);
pinMode(pinReset,OUTPUT);

// Se inicializa la señal de reloj, escribiendo un voltaje PWM
analogWrite(pinCLK,127);

// --- Inicializa velocidad de transmision serial ---
Serial.begin(9600);

// Manda un pulso de reset para inicializar la maquina de estados
finitos
digitalWrite(pinReset,~HIGH);
delay(sampleTime/50);
digitalWrite(pinReset,~LOW);
delay(sampleTime-sampleTime/50);

firstTime = millis();

// --- Sincronizacion de comunicacion serial ---
Serial.println("inicio");
leer_salidas();
leer_estados();
enviar_datos_seriales();

}

```

Figura 131. Subrutina “setup”.

En esta función inicializa 3 conjuntos de pines:

- Pines para **escribir la entrada** a la máquina de estados finitos como salida.
- Pines para **leer el estado** de la máquina de estados finitos como entrada.
- Pines para **leer la salida** de la máquina de estados finitos como entrada.

Adicionalmente inicializa los siguientes pines:

- Pin de **reset** como salida para inicializar la máquina
- Pin de **reloj** para sincronizar todas las operaciones que realiza la máquina.

Se configura la velocidad del reloj modificando el valor de la variable myPrescaler para obtener una frecuencia de PWM baja y observar el comportamiento de la máquina de estados.

6.3.4.2.3 Subrutina principal loop (2)

Nombre del programa: Arduino AR 65 Lectura y Prueba FSM



Figura 132. Ubicación del bloque **Prueba y lectura de la máquina de estados (2)**.

```
void loop() {  
    // Obtiene el siguiente valor binario a escribir a la entrada de la  
    // maquina de estados finitos  
    if(i<(sizeof(tabla_vectores_escritura_prueba)/2)-1) { i++; } else {  
        i=0; }  
  
    // Escribe a la entrada de la maquina uno de los valores de la tabla de  
    // vectores de prueba  
    escritura = tabla_vectores_escritura_prueba[i];  
  
    //imprimir_entrada_a_escribir();  
  
    // Escribe un valor a la entrada de la maquina de estados finitos  
    escribir_entradas();  
  
    delay(20);  
    // Lee la salida y el estado en el que se encuentra la maquina de  
    // estados finitos  
    leer_salidas();  
    leer_estados();  
  
    // Envia datos por el puerto serial  
    enviar_datos_seriales();  
  
    delay(sampleTime);  
}
```

Figura 133. Subrutina principal “loop”.

Esta función realiza las siguientes funciones:

- Obtiene la entrada a escribir en la máquina de estados a partir de la tabla que define el **vector de entradas para probar la máquina de estados (0)**.
- **escribir_entradas**: Escribe la entrada a la máquina de estados.
- **leer_salidas**: Lee las salidas de la máquina de estados.
- **leer_estado**: Lee el estado de la máquina de estados.
- **enviar_datos_seriales**: Envía los datos por el puerto serial.

6.3.4.2.3.1 Subrutina para escribir entradas (2a)

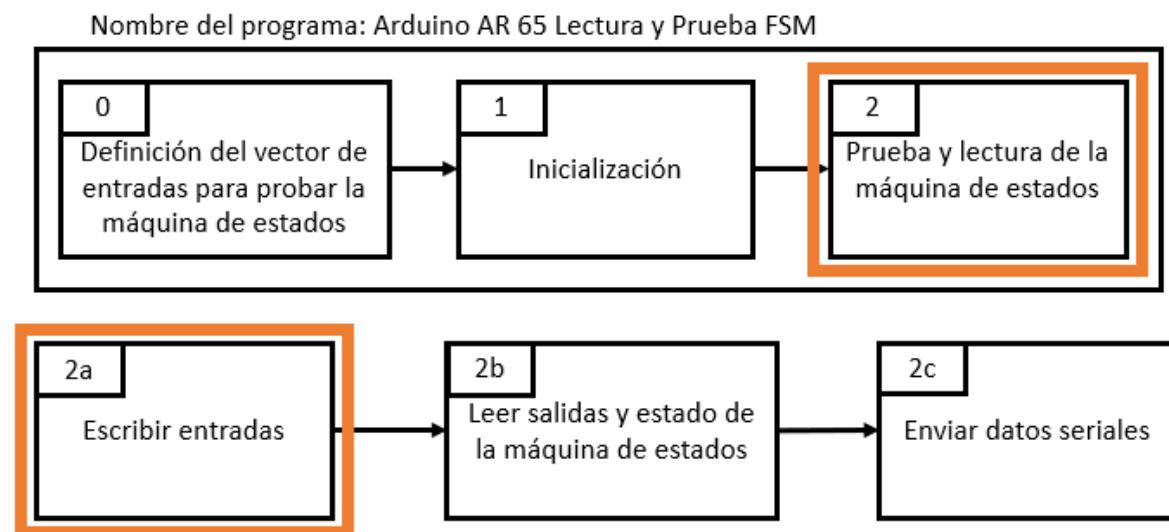


Figura 134. Ubicación del bloque **Escribir entradas (2a)**.

```
/**  
 * Escribe un valor en binario a la entrada de la maquina de estados finitos  
 * Convierte el valor en decimal a escribir a binario  
 */  
void escribir_entradas()  
{  
    int temp = escritura;  
    for(int i=(sizeof(pin_escrituras)/2)-1;i>=0;i--) {  
        //imprimir_estado_escritura(i,escritura);  
        if(escritura>=(pow(2,i)))  
        {  
            escritura -= round(pow(2,i));  
            digitalWrite(pin_escrituras[i],HIGH);  
            //imprimir_escritura_i(i,1);  
        }  
        else  
        {
```

```

        digitalWrite(pin_escrituras[i], LOW);
        //imprimir_escritura_i(i,0);
    }
}
escritura = temp;
}

```

Figura 135. Subrutina “escribir_entradas”.

Esta función escribe una entrada en la máquina de estados, en los pines de escritura. Por ejemplo, si se tiene que escribir un 5, en los pines de entrada de la máquina de estados finitos se escribe un 101.

6.3.4.2.3.2 Subrutina para leer salida y estado (2b)

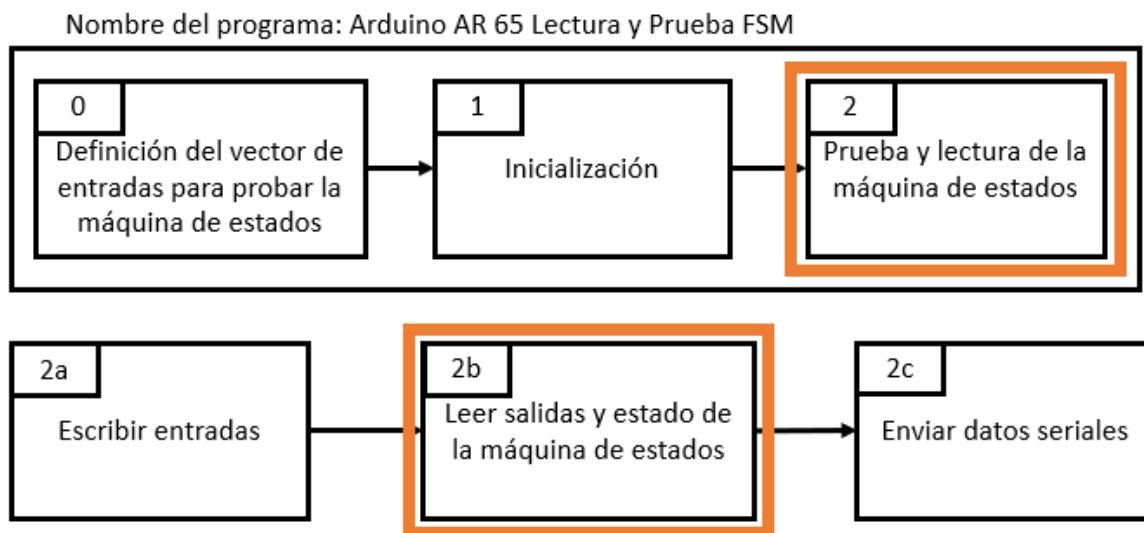


Figura 136. Ubicación del bloque **Ler salidas y estado de la máquina de estados (2b)** (parte de lectura de la salida de la máquina de estados finitos).

```

/**
 * Lee un valor binario a la salida de la maquina de estados finitos
 * Convierte este valor en binario leido a un valor decimal
 */
void leer_salidas()
{
    lectura = 0x00;

    for(int i=0;i<sizeof(pin_lecturas)/2;i++) {
        //imprimir_anter_suma_lectura();
        lectura += (byte)(digitalRead(pin_lecturas[i])*pow(2,i)+0.5);
        //imprimir_comparacion_estado(i);
    }
}

```

```

    //lectura = byte(lectura);
    //imprimir_despues_suma_lectura();
}

```

Figura 137. Subrutina “leer_salidas”.

Esta función realiza la lectura de la salida de la máquina de estados finitos. Cada pin leído es un bit, por lo que el conjunto de pines de salida se convierte a un número decimal.

Nombre del programa: Arduino AR 65 Lectura y Prueba FSM

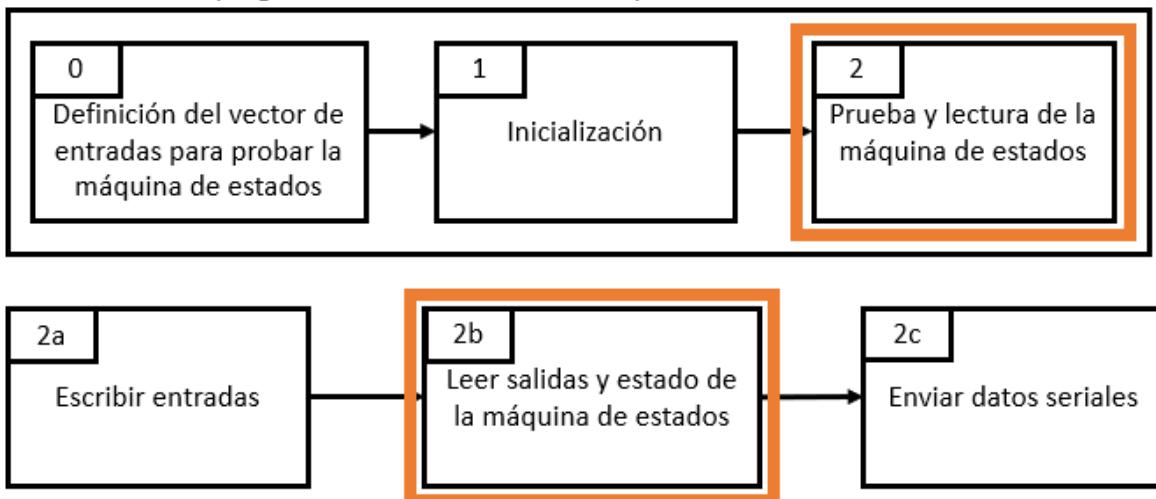


Figura 138. Ubicación del bloque **Leyendo salidas y estado de la máquina de estados (2b)** (parte de la lectura del estado de la máquina de estados finitos).

```

/**
 * Lee un valor binario correspondiente al estado de la maquina de estados finitos
 * Convierte este valor en binario leido a un valor decimal
 */
void leer_estados()
{
    estado = 0x00;
    for(int i=0;i<sizeof(pin_estados)/2;i++){

        //imprimir_pin_estados(i);
        estado += (byte)(digitalRead(pin_estados[i])*pow(2,i)+0.5);
    }
}

```

Figura 139. Subrutina “leer_estados”.

Esta función realiza la lectura del estado de la máquina de estados finitos. Cada pin leído es un bit, por lo que el conjunto de pines de estado se convierte a un número decimal.

6.3.4.2.3.3 Subrutina enviar_datos_seriales (2c)

Nombre del programa: Arduino AR 65 Lectura y Prueba FSM

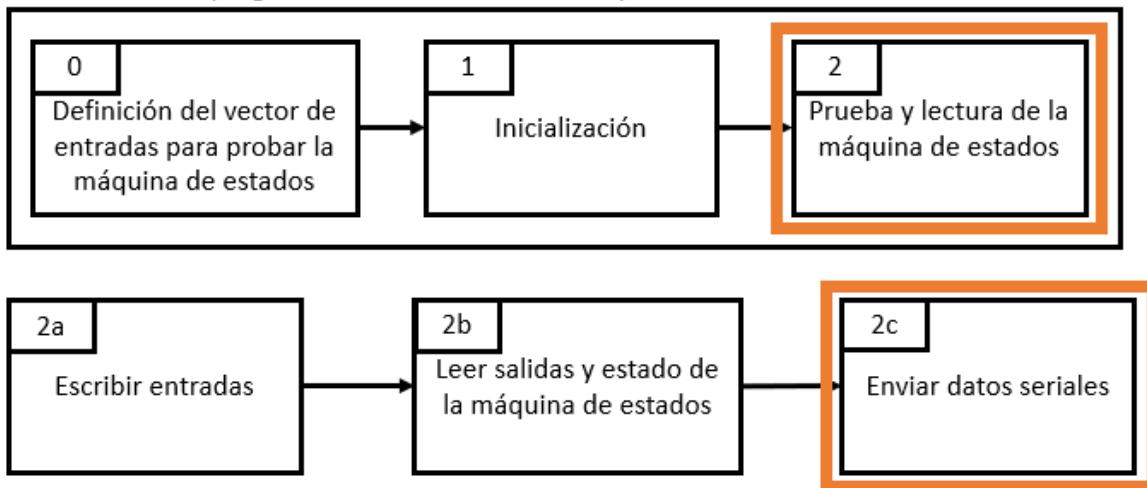


Figura 140. Ubicación del bloque Enviar datos seriales (2c).

```
/**  
 * Envía el valor que se le envio a la maquina, la salida y el estado de  
 misma  
 */  
void enviar_datos_seriales(){  
 //Serial.print("Escritura: ");  
 Serial.println(escritura);  
 //Serial.print(" , Estado: ");  
 Serial.println(estado);  
 //Serial.print(" , Lectura: ");  
 Serial.println(lectura);  
}
```

Figura 141. Subrutina “enviar_datos_seriales”.

Esta función envía los datos en el siguiente orden:

1. Variable **escritura**, que es la **entrada** escrita a la máquina de estados
2. Variable **estado** de la máquina de estados
3. Variable **lectura** que es la lectura de la salida de la máquina de estados

6.3.4.2.4 Subrutinas adicionales de impresión a monitor serial

Programa de Arduino para la lectura y prueba del Circuito Secuencial
Nombre del programa: Arduino AR 65 Prueba y Lectura FSM

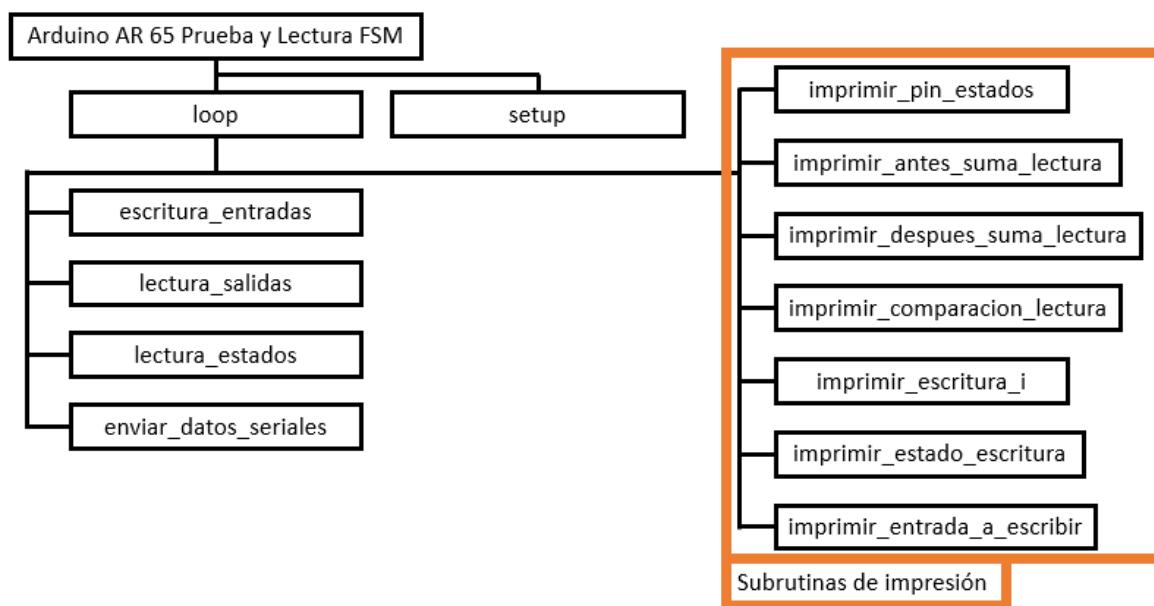


Figura 142. Ubicación de las **subrutinas de impresión**.

Este conjunto de subrutinas sirve para observar el comportamiento de la etapa de adquisición de datos de la máquina de estados finitos en el monitor serial. Se pueden observar diferentes partes de la etapa de adquisición de datos de la máquina de estados, como la entrada escrita, la salida y el estado leído de la máquina de estados. Cualquiera de estas subrutinas se puede comentar sin afectar el comportamiento del programa, ya que es solo para depuración.

6.4 Computadora y Cámara de Video (3)

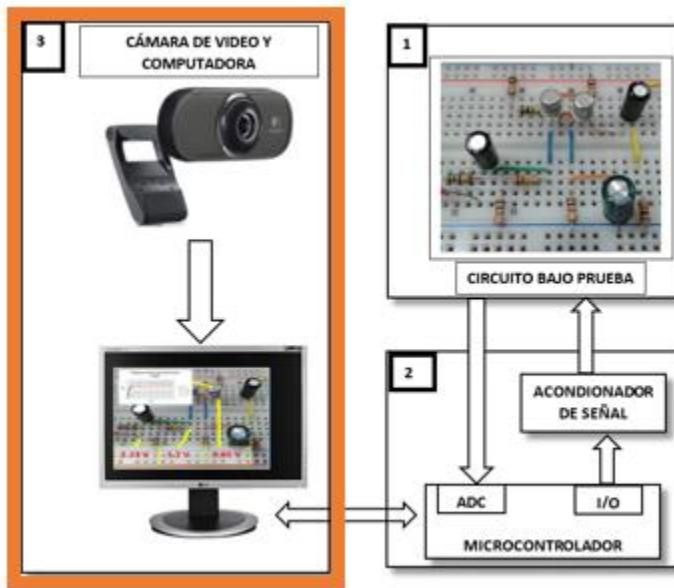


Figura 143. Ubicación del bloque Cámara de video y computadora (3).

Para implementar el uso de la realidad aumentada se utilizó el **lenguaje de programación Python** y algunos **paquetes de OpenCV** para el manejo de la imagen del circuito. Se trabajó sobre el sistema operativo **Windows 10**.

Se aplicará la **realidad aumentada** a la **imagen del circuito** capturada por una **cámara de video**, la cual estará conectada a la computadora.

Esta cámara deberá estar bien orientada al circuito para identificar los marcadores que se encuentran (orientada de frente a unos 10 a 20 cm del circuito). La cámara debe tener una buena resolución (recomendable HD o FullHD) para que se puedan identificar los marcadores por medio de los programas. La imagen del circuito capturada debe contener exclusivamente al protoboard, procurando que no salgan más objetos dentro de la imagen (ruido). Cabe señalar que es necesario tomar la imagen del circuito de prueba con una buena iluminación, es decir, poder distinguir en la imagen del circuito las conexiones (alambrado) del mismo en el protoboard.

Las señales provenientes de la etapa de adquisición de datos se envían a la computadora por **comunicación serial** a través de un puerto USB. Los datos, como se observará a continuación, fueron enviados uno por uno a través del **puerto serial**.

Con estos datos, se **calcularán** por software (programa en python) los **parámetros eléctricos** necesarios que permitan generar el despliegue visual, tanto de dichos datos, como del funcionamiento de los dispositivos y/o circuito, así como **gráficas informativas** del mismo, conectando visualmente dichos parámetros, comportamientos y/o gráficas con la imagen del circuito bajo prueba. Igualmente, con los datos obtenidos se podrá realizar el

análisis de éstos y comprobar si se tienen errores de funcionamiento en el mismo circuito, o simplemente observar cómo varían los parámetros eléctricos de éste al variar los valores de los componentes ó la interconexión entre éstos.

Se presentarán tanto las imágenes utilizadas como los códigos en Python de los 4 circuitos desarrollados:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

6.4.1. Imágenes y marcadores ARUCO utilizados

Para desplegar gráficas e información sobre la imagen del circuito, se utilizaron los marcadores ARUCO, los cuales son los siguientes:

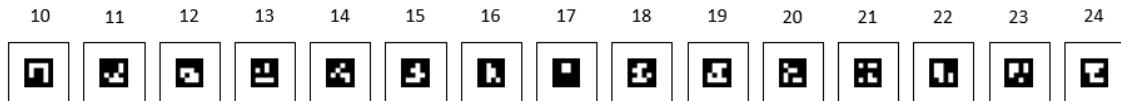


Figura 144. Marcadores ARUCO utilizados.

Hay 15 marcadores diferentes. Las dimensiones utilizadas para identificar cada marcador son las siguientes:

- Medidas del marcador: 0.4 cm x 0.4 cm
- Medidas del cuadro: 0.8 cm x 0.8 cm
- Línea del recuadro: 0.25 puntos.

Es importante que haya parte en blanco entre el marcador y el recuadro para que el programa en Python pueda identificar el marcador en cuestión.

En cada programa se utilizan los siguientes marcadores:

1. **Trazador de curvas (a):** Se utilizan los marcadores 10 a 16 para mostrar 7 gráficas.
2. **Circuito generador de señal senoidal (b):** Se utilizan los marcadores 10 a 13 para mostrar 4 gráficas:
3. **Amplificador de una etapa (c):** Se utilizan los marcadores 10 a 17 para mostrar 8 gráficas:
4. **Circuito secuencial (d):** Se utilizan los marcadores 10 a 14 para mostrar 5 gráficas:

6.4.1.1 Imagen utilizada para el circuito Trazador de curvas (a)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

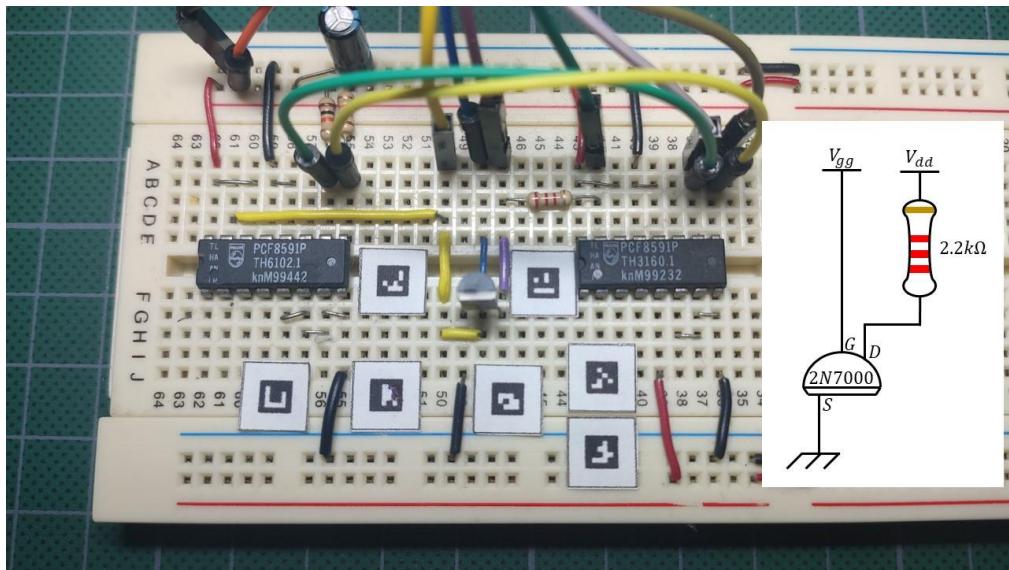


Figura 145. Imagen utilizada para el circuito **Trazador de curvas (a)**.

Para este circuito se utilizaron 7 marcadores. En estos 7 marcadores se despliega lo siguiente:

1. **Voltajes en el tiempo** usados para generar las diferentes señales del circuito.
2. **Gráfica I_g vs V_{gs}** . Curva característica de entrada del transistor MOSFET.
3. **Gráfica I_d vs V_{gs}** . Curva característica de transferencia del transistor MOSFET.
4. **Gráfica I_d vs V_{ds}** . Familia de curvas características de salida del transistor MOSFET.
5. **Gráfica g_m vs I_d** . Comportamiento de la transconductancia g_m en función de la corriente I_d con la que opera el transistor MOSFET.
6. **Gráfica r_d vs I_d** . Comportamiento de la resistencia r_d del modelo híbrido pi en función de la corriente I_d con la que opera el transistor MOSFET.
7. **Gráfica de parámetros híbridos**. Se presentan los valores de λ , K_n , voltaje de umbral V_{th} , resistencia híbrida r_g del transistor MOSFET.

6.4.1.2 Imagen utilizada para el circuito generador de señal senoidal (b)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

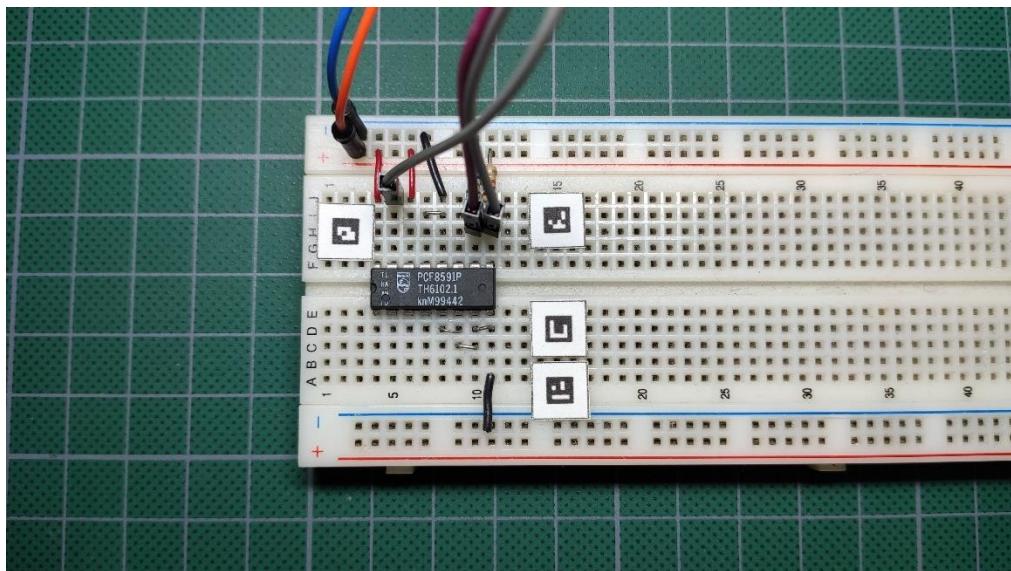


Figura 146. Imagen utilizada para el **circuito generador de señal senoidal (b)**.

Para este circuito se utilizaron 4 marcadores. En estos 4 marcadores se despliega lo siguiente:

1. **Comparación entre voltaje y binario (1)** en forma de una gráfica comparativa entre voltaje analógico de salida y número digital en binario que genera ese voltaje.
2. **Comparación entre voltaje y binario (2)** en forma de tabla comparativa entre voltaje analógico de salida y número digital en binario que genera ese voltaje.
3. **Comparación entre voltaje y binario (3)** mostrando los números binarios sobre la gráfica de voltaje analógico.
4. **Comparación entre voltaje y binario (4)** en forma de diagrama de estados.

6.4.1.3 Imagen utilizada para el circuito Amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

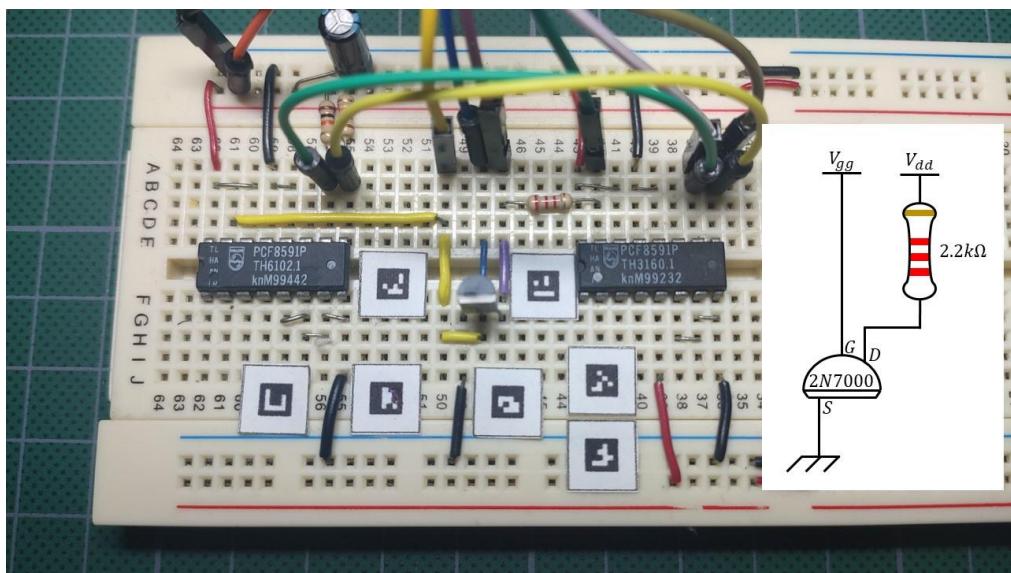


Figura 147. Imagen utilizada para el circuito Amplificador de una etapa (c).

Para este circuito se utilizaron 8 marcadores. Se despliegan las mismas graficas descritas que para el trazador de curvas descritas en la sección 6.4.1.1, modificando y agregando lo siguiente:

1. **Voltajes en el tiempo** usados para generar las diferentes señales del circuito.
2. **Gráfica I_g vs V_{gs}** . Curva característica de entrada del transistor MOSFET.
3. **Gráfica I_d vs V_{gs}** . Curva característica de transferencia del transistor MOSFET.
4. **Gráfica I_d vs V_{ds}** . Familia de curvas características de salida del transistor MOSFET.
5. **Gráfica g_m vs I_d** . Comportamiento de la transconductancia g_m en función de la corriente I_d con la que opera el transistor MOSFET.
6. **Gráfica r_d vs I_d** . Comportamiento de la resistencia r_d del modelo híbrido pi en función de la corriente I_d con la que opera el transistor MOSFET.

7. **Voltajes del amplificador en fuente común** en donde se observa el comportamiento del transistor como amplificador de pequeña señal.
8. **Gráfica de parámetros híbridos y punto de operación.** Se presentan los valores de λ , K_n , voltaje de umbral V_{th} , resistencia híbrida r_g del transistor. Adicionalmente, se presentarán parámetros del punto de operación, como V_{dsq} , I_{dq} , transconductancia g_{mq} , resistencias híbridas r_d y r_g , resistencia equivalente R_{lac} y ganancia.

6.4.1.4 Imagen utilizada para el circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

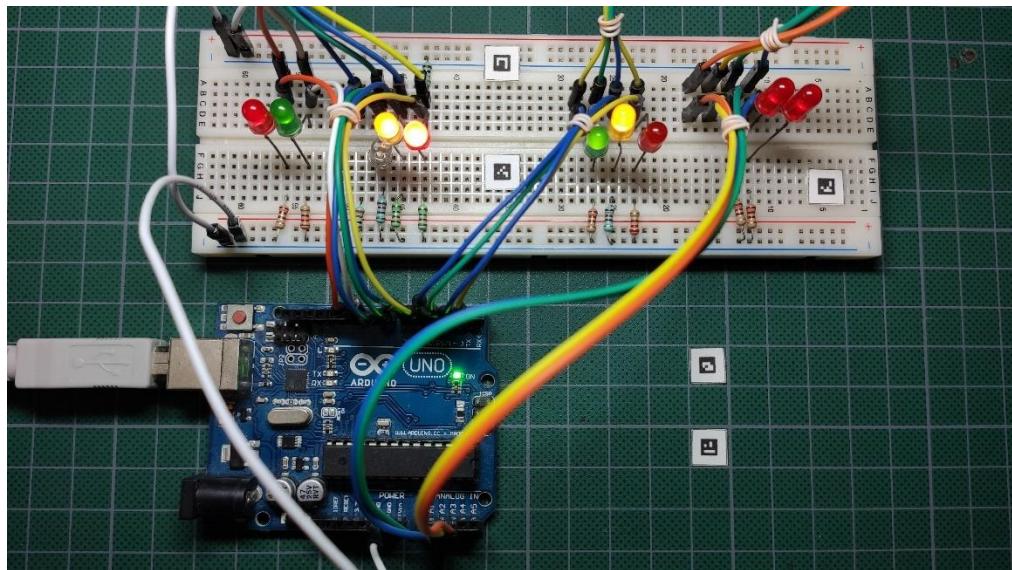


Figura 148. Imagen utilizada para el **circuito secuencial (d) implementado por hardware (i).**

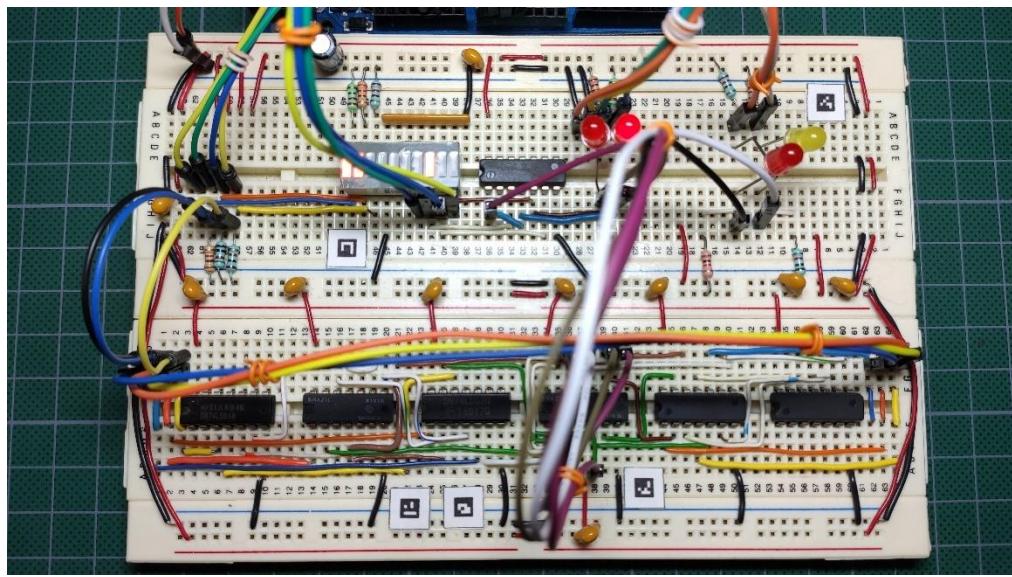


Figura 149. Imagen utilizada para el **circuito secuencial (d) implementado por software (ii).**

Para ambos circuitos se utilizaron 5 marcadores. En estos 5 marcadores se despliega lo siguiente:

1. **Señales digitales binarias en el tiempo** en donde se observan la entrada, estado y salida de la máquina de estados finitos.
2. **Tabla de estados binarios** en donde se muestran los diferentes estados de la máquina de estados, la salida correspondiente a cada estado y las transiciones hacia otros estados, según la sea la entrada que tenga la máquina de estados.
3. **Diagrama de estados (1)** de la máquina de estados finitos
4. **Diagrama de estados (2)** de la máquina de estados finitos
5. **Diagrama de transiciones de la máquina de estados finitos.**

6.4.2. Programas en Python

Para poder ejecutar los programas en Python, es necesario tener instalado los siguientes paquetes:

- engineering-notation
- graphviz
- imutils
- matplotlib
- numpy
- pyserial
- serial
- cv2 (opencv2)

Para todos los programas implementados en Python que realizan tanto la comunicación con Arduino, los cálculos y despliegue de gráficas e información, se utilizó como base un programa proporcionado por el asesor presentado al final del documento en la sección 10.1.2. Este programa está basado en la estructura modelo-vista-controlador. El diagrama completo de clases es el siguiente:

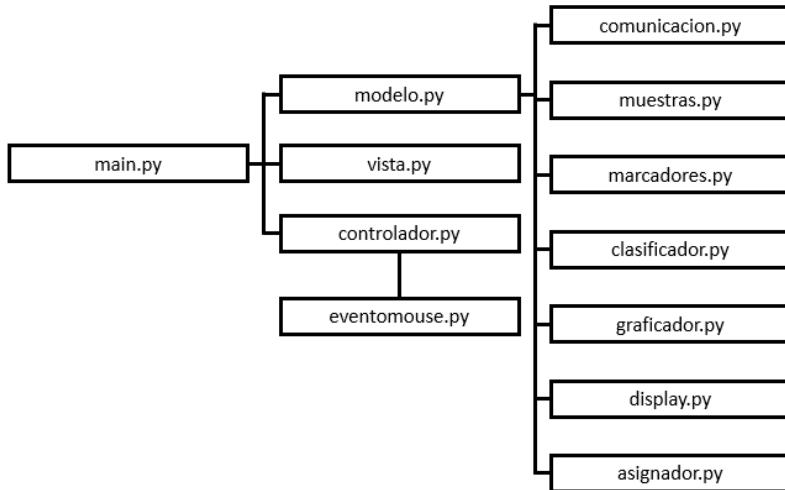


Figura 150. Diagrama de clases del programa base.

- **main.py:** Es el programa principal. Ejecuta los programas modelo, vista y controlador
- **modelo.py:** Implementa el comportamiento del programa.
- **vista.py:** Se encarga de la parte gráfica de la aplicación.
- **controlador.py:** Se encarga de todos los eventos del mouse sobre la aplicación.

La clase modelo.py se apoya en las siguientes clases:

- **comunicacion.py**: Se encarga de la comunicación entre el Arduino y el programa de Python.
- **muestras.py**: Sigue el envío de datos desde Arduino y clasifica las muestras por los canales definidos (ej. separa las muestras asociadas a la medición de la corriente I_d y del voltaje V_{ds}).
- **marcadores.py**: Identifica los marcadores colocados sobre la imagen utilizada y obtiene las coordenadas de cada uno de ellos.
- **clasificador.py**: Clasifica los marcadores de acuerdo con sus coordenadas x y y.
- **graficador.py**: Grafica, dibuja y muestra información del circuito.
- **display.py**: Genera sobre la ventana principal del programa los cuadros de texto y áreas para mostrar las gráficas e información.
- **asignador.py**: Asigna a cada marcador una ventana en donde se puede colocar una gráfica o información.

La clase controlador se apoya en la siguiente clase:

- **evetomouse.py**: Maneja los siguientes eventos del mouse:
 - Botón izquierdo abajo.
 - Mover mouse.
 - Botón izquierdo arriba.

Sobre la base de estos programas se realizó este proyecto. Se trabajó principalmente sobre la **clase graficador.py** para trazar las gráficas necesarias para cada circuito.

En algunos casos, se agregó una clase extra que es **calculadora_datos_graficas.py** que ayuda a la clase graficador.py para principalmente realizar cálculos y obtener coordenadas de los elementos a graficar. El diagrama de clases queda de la siguiente manera:

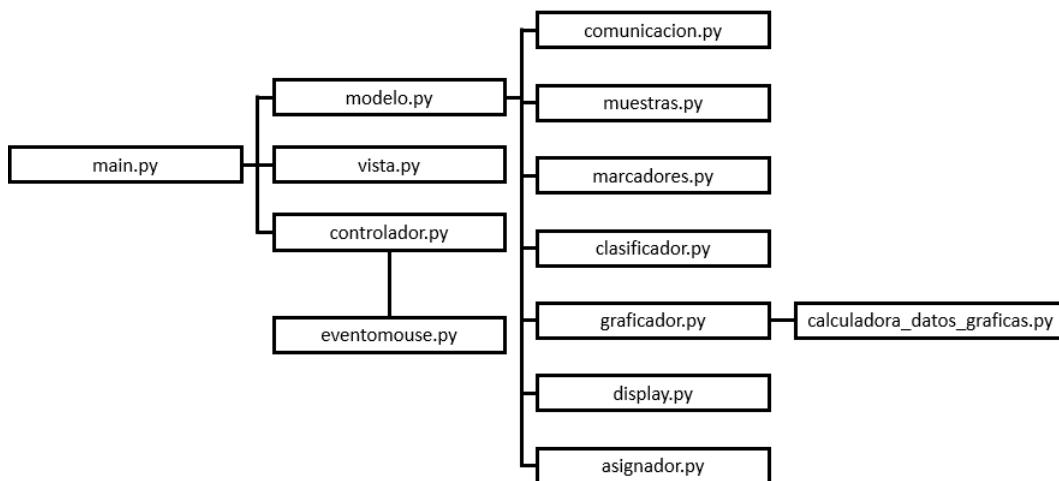


Figura 151. Diagrama de clases con la clase `claculadora_datos_graficas.py`.

A continuación, se detallarán más las partes que cambiaron del programa base, así también se detallará más sobre la clase graficador.py y la clase calculadora_datos_graficas.py, las cuales fueron las clases en las que se trabajaron en este proyecto.

En el programa **main.py** se modificó la línea correspondiente a la imagen que se utilizará en el programa:

En el programa **modelo.py** en el método “`inicializar_objetos_modelo`” se modificó el arreglo de llaves que se utiliza en el programa, es decir, el nombre de los canales de datos que recibe el programa. Se modificó también el número y nombre de las gráficas a presentar.

Se detallará sobre todo el método “`graficar_cuvas`” de la clase **graficador.py**, que es en donde más se trabajó para obtener las gráficas.

6.4.2.1 Trazador de Curvas (a)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Círcuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Círcuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el programa de Python del circuito **Trazador de Curvas (a)**, se agregó la clase `calculadora_datos_graficas.py`, quedando el diagrama de clases de la siguiente manera:

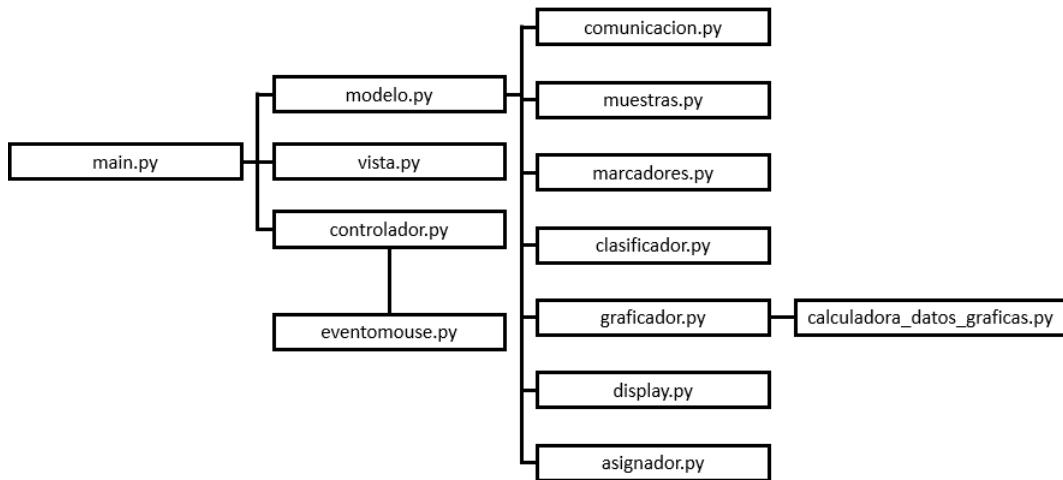


Figura 152. Diagrama de clases para el **círcuito trazador de curvas (a)**.

6.4.2.1.1 Modificaciones en la clase modelo.py

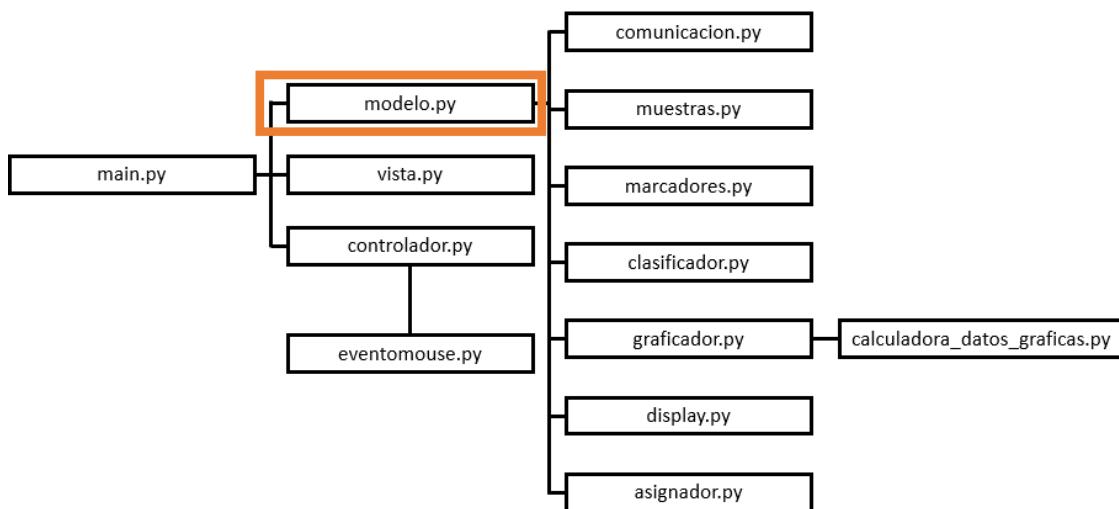


Figura 153. Ubicación de la clase modelo.py.

Esta clase contiene los siguientes métodos y atributos:



Figura 154. Métodos, atributos y propiedades del método modelo.py.

Donde:

- (c) indica que es una **clase**.
- (m) indica que es un **método**.
- (p+) indica que es una **propiedad**.
- (f) indica que es un **atributo**.

Las modificaciones realizadas en la clase **modelo.py** se realizaron sobre el método “**inicializar_objetos_modelo**”.

En la clase modelo en el método “**inicializar_obterjos_modelo**” se modifica lo siguiente:

```
titulos_vectores_muestras = ["Vgg", "Vgs", "Ig", "Vdd", "Vds", "Id"]
param_muestras = {'numMuestras': 300, 'numCanales':
len(titulos_vectores_muestras),
'Illave': titulos_vectores_muestras}
```

Figura 155. Definición de los títulos de las llaves para los vectores de muestras y número de muestras.

Se toman **300 muestras** para cada uno de los **6 canales de muestras** V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Hay que recordar que tanto I_g como I_d son la diferencia de voltajes V_{gg} con V_{gs} y V_{dd} con V_{ds} respectivamente.

De las 300 muestras:

- 250 muestras se tomaron para graficar las **curvas I_d vs V_{ds}** (5 series de 50 muestras para 5 **curvas I_d vs V_{ds}**).
- 50 muestras para las demás curvas (1 serie de 50 muestras para las **curvas I_g vs V_{gs} , I_d vs V_{gs} , g_m vs I_d , r_d vs I_d** y cálculo de **parámetros híbridos**).

```
titulos_marcadores_display = {'m0': "Voltajes",
'm1': "Ig vs Vgs",
'm2': "Id vs Vgs",
'm3': "Id vs Vds",
'm4': "gm vs Id",
'm5': "rd vs Id",
'm6': "Parametros hibridos"}
```

Figura 156. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.

Se presentarán 7 gráficas, descritas en la sección 6.4.1.1.

6.4.2.1.2 Descripción de las variables de los datos recibidos desde el microcontrolador

La variable **vectores_muestras** contiene los 6 canales de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Es un diccionario de tres dimensiones. La variable **lim_max** contiene el número de muestras por canal. En este caso hay 300 muestras por canal.

Para acceder a cada uno de los datos se tiene que especificar:

- **Llave del canal de muestras:** Se indica entre comillas el nombre del canal de muestras a utilizar. Por ejemplo, para acceder las muestras del canal V_{gg} con filtro circular, se coloca 'Vgg_pc'.
- **Serie de datos:** Cada canal de muestras tiene varias series de 50 muestras. Para acceder a cada serie de datos, se indica con un número entero entre 0 y el valor especificado por la expresión $\frac{\text{lim_max}}{50 \text{ muestras}}$. En este caso hay 6 series de datos de 50 muestras. Por ejemplo, para seleccionar la serie 3 que contiene las muestras 101 a 150, se colocaría un 3.
- **Número de la muestra o conjunto de muestras:** Se accede a una muestra específica, colocando un número de entre 0 y 50. También es posible acceder a un conjunto de muestras, colocando la muestra inicial, seguido de dos puntos e indicando la muestra final. Por ejemplo, para acceder de la muestra 4 a 25 de la serie de datos 3, se coloca el rango 4:25.

Por ejemplo, para acceder a las muestras del canal V_{gg} con filtro circular, serie de datos 3 y muestras 4:25, se indica de la siguiente manera:

```
vectores_muestras['Vgg_pc'][3][4:25]
```

Para el programa del **círculo Trazador de curvas (a)**, se toman los 6 canales de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Por cada canal se toman 6 series de datos de 50 muestras cada uno.

Tomando como ejemplo el canal V_{gg} , se tienen las siguientes series de muestras:

Serie de muestras	V_{gg}
Serie 0 (muestras 0 a 49)	vectores_muestras['Vgg'][0]
Serie 1 (muestras 50 a 99)	vectores_muestras['Vgg'][1]
Serie 2 (muestras 100 a 149)	vectores_muestras['Vgg'][2]
Serie 3 (muestras 150 a 199)	vectores_muestras['Vgg'][3]
Serie 4 (muestras 200 a 249)	vectores_muestras['Vgg'][4]
Serie 5 (muestras 250 a 299)	vectores_muestras['Vgg'][5]

Tabla 21. Descripción de los canales de muestras.

A cada canal de muestras se le aplican dos filtros diferentes, un filtro circular y un filtro pasa bajas, guardando cada canal con filtro con un nombre diferente, mostrado en la tabla 22:

Canal de muestras	Nombre del canal de muestras		
	Sin filtro	Con filtro circular	Con filtro pasa bajas
V_{gg}	Vgg	Vgg_pc	Vgg_lp
V_{gs}	Vgs	Vgs_pc	Vgs_lp
I_g	Ig	Ig_pc	Ig_lp
V_{dd}	Vdd	Vdd_pc	Vdd_lp
V_{ds}	Vds	Vds_pc	Vds_lp
I_d	Id	Id_pc	Id_lp

Tabla 22. Descripción de los canales con filtro circular y pasa bajas.

La variable **muestras_llaves** contiene los títulos de las llaves de la variable **vector_muestras** mostrados en la tabla 22. El orden en que se guardan es el siguiente:

Orden	Canal de datos
0	Vgg
1	Vgs
2	Ig
3	Vdd
4	Vds
5	Id
6	Vgg_pc
7	Vgs_pc
8	Ig_pc
9	Vdd_pc
10	Vds_pc
11	Id_pc
12	Vgg_lp
13	Vgs_lp
14	Ig_lp
15	Vdd_lp
16	Vds_lp
17	Id_lp

Tabla 23. Orden de los títulos en la variable `muestras_llaves`.

Cada uno de los canales expuestos de la tabla 23 contienen 300 muestras cada uno. Se observa la variable `muestras_llaves` guarda los títulos de las llaves en el siguiente orden:

0. Títulos de las llaves de los canales **sin filtro**.
1. Títulos de las llaves de los canales **con filtro circular**.
2. Títulos de las llaves de los canales **con filtro pasa bajas**.

Usando la variable **sel_filtro** permite acceder de manera sencilla a estos canales, tomando valores de 0 a 2 en donde

0. Accede a los canales **sin filtro**.
1. Accede a los canales **con filtro circular**.
2. Accede a los canales **con filtro pasa bajas**.

La variable **ind_corrientes** e **ind_voltajes** son arreglos de enteros que permiten acceder a los canales de voltajes y corrientes de manera sencilla, ya que contienen sus índices (con respecto al orden en que se reciben: V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d):

Orden en la variable ind_voltajes	0	1	2	3
Canal de datos	V_{gg}	V_{gs}	V_{dd}	V_{ds}
índice	0	1	3	4

Tabla 24. Orden de los índices de voltajes para la variable **ind_voltajes**.

Orden en la variable ind_corrientes	0	1
Canal de datos	I_g	I_d
índice	2	5

Tabla 25. Orden de los índices de corrientes para la variable **ind_corrientes**.

Por ejemplo, para acceder a las muestras del canal V_{gg} con filtro circular, serie de datos 3 y muestras 4:25, se indica de la siguiente manera (utilizando las variables **ind_voltajes**, **lim_max**, **sel_filtro**, **muestras_llaves**):

```
vectores_muestras[muestras_llaves[muestras[ind_voltajes[0]+sel_filtro*(lim_max/50)]]][3][4:25]
```

En donde

```
sel_filtro = 1
lim_max = 300
```

6.4.2.1.3. Clase calculadora_datos_graficas.py

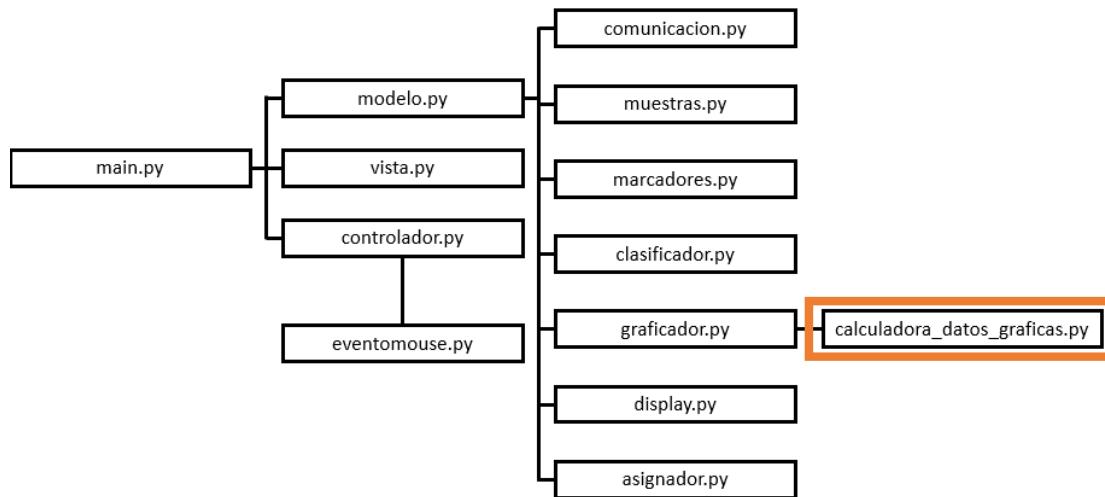


Figura 157. Ubicación de la clase calculadora_datos_graficas.py.

Esta clase contiene los siguientes métodos y atributos:

```
© m.calculadora_datos_graficas.Calculadora_datos_graficas
m __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes)
m estimate_coef(self, x, y)
m plot_regression_line(self, x, y, b)
m construir_series_datos(self)
m obtener_ind_min(self,ind_serie_datos)
m obtener_ind_max(self,ind_serie_datos,ind_min)
m regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max)
m inicializar_excel_datos(self)
m guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId, voltajeVds)
m finalizar_excel_datos(self,writer)
m leer_datos_excel(self,ind_variables, i)
m calcular_lambda(self, linea_lim_y)
m calcular_Vth_Kn_Idss(self)
m calcular_puntos_límite_grafica_ids_vds(self, Vth)
f muestras_llaves
f sel_filtro
f ind_voltajes
f ind_corrientes
f muestras
f modelo
f lim_max
```

Figura 158. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.

Donde:

-  indica que es una **clase**.
-  indica que es un **método**.
-  indica que es una **propiedad**.
-  indica que es un **atributo**.

Se presentan los siguientes métodos:

1. **Método constructor (1)**. Se presenta el método “`__init__`”
2. **Método para calcular V_{th} , K_n e I_{dss} (2)**. Se presenta el método “`calcular_Vth_Kn_Idss`”.
3. **Método para calcular puntos límite en la gráfica I_d vs V_{ds} (3)**. Se presenta el método “`calcular_puntos_limite_grafica_ids_vds`”.
4. **Métodos para calcular lambda (4)**. Se presentan los siguientes métodos:
 - 4a. **Método “`calcular_lambda`” (4a)**.
 - 4b. **Métodos “`obtener_ind_min`” y método “`obtener_ind_max`” (4b)**.
 - 4c. **Método “`regresión_lineal_lambda`” (4c)**.
 - 4d. **Método “`estimate_coef`” (4d)**.

6.4.2.1.3.1 Constructor (1)

```
def __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes):  
  
    print("")  
    print(" CONSTRUCTOR: Clase: Calculadora_datos_graficas")  
    self.modelo = modelo  
    self.muestras = modelo.vectores_muestras  
    self.muestras_llaves = muestras_llaves  
    self.lim_max = lim_max  
    self.sel_filtro = sel_filtro  
    self.ind_corrientes = ind_corrientes  
    self.ind_voltajes = ind_voltajes
```

Figura 159. Método constructor.

En el constructor se traen todas las variables importantes para los cálculos. Estas son:

- **muestras:** Es una copia del diccionario **vectores_muestras** que contiene el diccionario de las muestras tomadas de los 6 canales correspondientes a V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **muestras_llaves.** Contiene un arreglo de los nombres de las llaves del diccionario “**vectores_muestras**”. Es decir, contiene los nombres V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d para identificar cada canal de muestras tomadas.
- **lim_max:** Contiene el número total de muestras tomadas por canal que es de **300 muestras**. En este caso, se tomaron muestras para 6 series de datos para cada canal de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **sel_filtro.** Indica cuál conjunto de datos se usará para todos los canales de muestras (**sin filtro (0)**, con **filtro circular (1)**, con **filtro pasa bajas (2)**).
- **ind_corrientes.** Indica para los 6 canales de muestras, cuáles son canales de muestras de corrientes.
- **ind_voltajes.** Indica para los 6 canales de muestras, cuáles son canales de muestras de voltajes.

6.4.2.1.3.2 Método para calcular Vth, Kn e Idss (2)

```

# -----
# Calcula los parametros Vth, Kn e Idss del transistor MOSFETT
# Ademas, calcula los indices de ids-vgs que se utilizaron para calcular estos parametros
(Vth,Kn,Idss)
# -----
def calcular_Vth_Kn_Idss(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    lim_max = self.lim_max
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    # --- Se calcula la maxima variacion de ids, entre muestras adyacentes ---
    ind_ids = []
    delta_ids = [0]*50
    for i in range(1,50-1):
        delta_ids[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][i]
        - muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][i-1]
        #delta_vgs = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][i] -
        muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][i-1]
        #print(['+str(i)+' 'delta_ids: '+str(delta_ids[i])#+', delta_vgs: '+str(delta_vgs))]

    # --- Se identifica el indice de la maxima variacion y el indice anterior a esta muestra -
    --
    ids_max = np.max(delta_ids)
    ind_ids.append(delta_ids.index(ids_max)-1)
    ind_ids.append(delta_ids.index(ids_max))
    #print('ind_ids: '+str(ind_ids))

    # --- Se calcula la diferencia de vgs y de ids, con los indices anteriores ---
    vgs_1 = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][ind_ids[0]]
    vgs_2      =      muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-
    1][ind_ids[len(ind_ids)-1]]
    ids_1      =      muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-
    1][ind_ids[0]]
    ids_2      =      muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-
    1][ind_ids[len(ind_ids)-1]]

    #print('ids_1: '+str(ids_1))
    #print('ids_2: '+str(ids_2))
    #print('vgs_1: '+str(vgs_1))
    #print('vgs_2: '+str(vgs_2))

    # --- Se calcula vth, kn e idss ---
    Vth = (vgs_1-vgs_2*math.sqrt((ids_1)/(ids_2)))/(1-math.sqrt((ids_1)/(ids_2)))
    Kn = (ids_1)/((vgs_1-Vth)**2)

```

```

Idss = Kn * Vth**2

print('Vth_calculado: '+str(Vth))
print('Kn_calculada: '+str(Kn))
print('Idss_calculada: '+str(Idss))

return (Vth, Kn, Idss, ind_ids)

```

Figura 160. Método “calcular_Vth_Kn_Idss”.

Este método realiza lo siguiente:

1. Calcula sobre el conjunto de datos para la curva I_d vs V_{gs} los **índices** correspondientes a las muestras **con la diferencia máxima** entre valores adyacentes de **corriente I_d** .
2. Para el par de muestras con mayor diferencia, se obtienen **los voltajes V_{gs1} y V_{gs2}** y **corrientes I_{d1} e I_{d2}** correspondientes.
3. Se calcula V_{th} , K_n e I_{dss} .

Para calcular el **voltaje de umbral V_{th}** se utiliza la siguiente ecuación:

$$V_{th} = \frac{V_{gs1} - V_{gs2} \sqrt{\frac{I_{d1}}{I_{d2}}}}{1 - \sqrt{\frac{I_{d1}}{I_{d2}}}}$$

Para calcular K_n se utiliza la siguiente ecuación:

$$K_n = \frac{I_{d1}}{(V_{gs1} - V_{th})^2}$$

Para calcular I_{dss} se utiliza la siguiente ecuación:

$$I_{dss} = K_n V_{th}^2$$

6.4.2.1.3.3 Método para calcular puntos límite en la gráfica id vs vds (3)

```

# -----
# Calcula los puntos a graficar sobre las curvas vds-ids
# Se calcula la posicion de la curva que delimita las regiones ohmica y de saturacion de
un transistor MOSFET
# Regresa tambien el valor promedio vgs al cual se grafico esa curva
# -----
def calcular_puntos_limite_grafica_ids_vds(self, Vth):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_id_vds = [0,1,2,3,4]

    linea_lim_x = [0] * (len(num_curvas_id_vds)+1)
    linea_lim_y = [0] * (len(num_curvas_id_vds)+1)
    Vgs_prom = [0] * (len(num_curvas_id_vds))

    # --- Se calcula la posicion de los puntos que delimitan las regiones de operacion sobre
el conjunto de curvas ids-vds ---
    for i in num_curvas_id_vds:
        Vgs_prom[i] =
        round(np.average(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][i]),3)
        # --- Se obtiene el voltaje de saturacion vds_sat ---
        linea_lim_x[i+1] = Vgs_prom[i]-Vth
        for j in range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i])):
            if
                abs((Vgs_prom[i]-Vth) -
muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i][j]) < 0.05:
                # --- Se obtiene la muestra mas cercana de ids correspondiente al voltaje
vds_sat ---
                    linea_lim_y[i+1] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][j]
                    #print('lim_curva_'+str(i)+'
(temp): '+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+'A,     dif='+str(abs((Vgs_prom-Vth) -
muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i][j])/220))
                    print('lim_curva_'+str(i)+': '+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+'A')

    return (linea_lim_x,linea_lim_y,Vgs_prom)

```

Figura 161. Método “calcular_puntos_limite_grafica_ids_vds”.

Este método realiza lo siguiente sobre cada conjunto de muestras para las curvas I_d vs V_{ds} :

1. Obtiene el **voltaje promedio** V_{gs} usado para esa curva I_d vs V_{ds} . Esto se realiza porque puede haber ligeras variaciones del voltaje V_{gs} al tomar las muestras.
2. Se calcula el valor del **voltaje** $V_{ds(sat)}$ en el que el transistor MOSFET cambia de la región óhmica a la de saturación.
3. Se encuentra el **índice** de la muestra más cercana al valor de **voltaje** $V_{ds(sat)}$ calculado en el paso anterior.
4. Se obtiene tanto el valor de **voltaje** $V_{ds(sat)}$ como de como el de **corriente** $I_{d(sat)}$ correspondientes a ese índice.

6.4.2.1.3.4 Métodos para calcular lambda (4)

6.4.2.1.3.4.1 Método “*calcular_lambda*” (4a)

```
#-----
# -----
# FUNCION PRINCIPAL que calcula lambda de un transistor mos
# Obtiene primero los índices mínimos y máximos en donde se considera "recta" a la
# curva Id vs Vds
# Calcula lambda para esa curva
# Obtiene el promedio de todas las lambdas calculadas
# -----
def calcular_lambda(self,linea_lim_y):

    MOSFET_lambdas = []
    muestras = self.muestras
    lim_max = self.lim_max
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    #valores_id_vds = self.construir_series_datos()
    ind_min = 0
    ind_max = 0
    #print(valores)
    #print(valores['Id1'][0])

    for [ind_serie_datos,i] in zip(range(0,lim_max-1),range(1,lim_max)):
        #print('Id'+str(ind_serie_datos)+', '+Vds'+str(ind_serie_datos))
        #print('muestras[Id][ind_serie_datos]:')
        '+str(len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))+',
        '+muestras[Vds][ind_serie_datos]):'
        '+str(len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos]))')
        #print('Id'+str(ind_serie_datos)+': ')
```

```

#print(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos])
#print('Vds'+str(ind_serie_datos)+': ')
#print(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos])

    ind_min = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos].index(linea_lim_y[i])
    #ind_min = self.obtener_ind_min(ind_serie_datos)
    print('ind_min:           '+str(ind_min)+':           '+Id'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][ind_min]))

    ind_max = self.obtener_ind_max(ind_serie_datos,ind_min)
    print('ind_max:           '+str(ind_max)+':           '+Vds'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos][ind_max]))

    MOSFET_lambdas.append(self.regresion_lineal_lambda(ind_serie_datos,
ind_min, ind_max))
    print('lambda: '+str(MOSFET_lambdas[ind_serie_datos])+'\n')

print('lambda_promedio: '+str(statistics.mean(MOSFET_lambdas)))
#return statistics.mean(MOSFET_lambdas)
return MOSFET_lambdas

```

Figura 162. Método “calcular_lambda”.

Este método realiza para cada conjunto de datos para las curvas I_d vs V_{ds} lo siguiente:

1. Se obtienen los **índices** de dos muestras en donde se considera que la curva es ya una línea recta.
2. Se realiza la **regresión lineal** con estos dos puntos para obtener su intersección con el eje x correspondiente al voltaje.
3. El valor obtenido es el **inverso del parámetro lambda**.

Este método puede regresar el promedio todos los valores obtenidos o en cambio, los valores obtenidos de todas las curvas sin promediarlas.

6.4.2.1.3.4.2 Métodos “obtener_ind_min” y “obtener_ind_max”

(4b)

```
# -----
# Obtiene el límite mínimo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de corriente Id
# Esto es la primera muestra cuya diferencia con la anterior es menor a 0.06 y que el
índice de esa muestra sea mayor a la muestra número 2
# -----
def obtener_ind_min(self,ind_serie_datos):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes

    ind_min = 0;
    id_cambio = [x for x in
range(0,len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))]

    for i in range(1,30):
        id_cambio[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i]
        muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1] =
        #print('i=' +str(i)+':'
        '+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i])+'
        '+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1])+'
        '+str(id_cambio[i]))
        #print('i: ' +str(i)+', ' +str(id_cambio[i])+': ' +str(id_cambio[i] < 0.06))
        if id_cambio[i] < (0.06)/1000 :
            #print('i: ' +str(i)+', ' +str(id_cambio[i])+': ' +str(id_cambio[i] < 0.06))
            if i > (2+int(ind_serie_datos/2)+1):
                ind_min = i
                break

    #print('ind_min: id_cambio['+str(ind_min)+']: ' +str(id_cambio[ind_min]))

    return ind_min

# -----
# Obtiene el límite máximo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de voltaje Vds
# Esto es la primera muestra cuya diferencia con la anterior es mayor a 0.05 y que el
índice de esa muestra sea mayor a la muestra número 20
# -----
def obtener_ind_max(self,ind_serie_datos,ind_min):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes
```

```

ind_max
len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos])-2
    vds_cambio = [x for x in
range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos))])
    #print('Vds_cambio len: '+str(len(vds_cambio)))
    for i in
range(20,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos])):
        vds_cambio[i] = -
        muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i]
        muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i-1]
        #print('i='+str(i)+:
        '+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i])+
        '+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i-1])+'
        '+str(vds_cambio[i]))
        #print('i: '+str(i)+', '+str(vds_cambio[i])+': '+str(vds_cambio[i] < 0.06))
        if vds_cambio[i] < 0.05 :
            #print('i: '+str(i)+', '+str(vds_cambio[i])+': '+str(vds_cambio[i] < 0.05))
            if i > ind_min :
                ind_max = i-1
                break
#printf('ind_max: vds_cambio['+str(ind_max)+']: '+str(vds_cambio[ind_max]))

return ind_max

```

Figura 163. Métodos “obtener_ind_min” y “obtener_ind_max”.

Estos dos métodos **obtener_ind_min** y **obtener_ind_max** de la figura 163 calculan el **índice mínimo** e **índice máximo** de dos muestras en donde se considera que la curva I_d vs V_{ds} es una línea recta.

Para calcular el **índice mínimo** se compara que la diferencia de los valores de corriente adyacentes no sea mayor que un determinado valor. En el momento que sea menor, se considera que inicia la línea recta.

Para calcular el **índice máximo** se revisa que la diferencia de voltaje de las últimas muestras no sea menor que un determinado valor. Este es el valor en donde la fuente de alimentación ya no puede suministrar lo suficiente y las lecturas de voltaje ya no representan el comportamiento del transistor.

6.4.2.1.3.4.3 Método “regresión_lineal_lambda” (4a)

```
# -----
# Funcion que obtiene lambda para un transistor mosfet
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    #print(valores_id_vds[ind_variables[ind_serie_datos+1]][[ind_min:ind_max]])
    #print(valores_id_vds[ind_variables[ind_serie_datos]][[ind_min:ind_max]])

    x = np.array(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][[ind_min:ind_max]]).reshape(-1)
    y = np.array(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][[ind_serie_datos]][[ind_min:ind_max]]).reshape(-1)

    #print(x)
    #print(y)

    b = self.estimate_coef(x,y)
    #print("Estimated coefficients: b_0 = {} b_1 = {}".format(b[0], b[1]))
    #plot_regression_line(x, y, b)

    return -b[0] / b[1]
```

Figura 164. Método “regresión_lineal_lambda”.

El método de la figura 164 regresa el valor de la intersección con el eje x que es del voltaje V_{ds} . Esto se obtiene porque:

$$y = mx + b$$

Pero $y = 0$, por lo tanto, despejando x:

$$x = \frac{-b}{m}$$

Lo que devuelve la función es el **inverso del parámetro lambda**.

6.4.2.1.3.4.4 Método “estimate_coef” (4a)

```
# -----
# Calcula los coeficientes b0 y b1 del método de regresión lineal simple
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def estimate_coef(self, x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)
```

Figura 165. Método “estimate_coef”.

Este método realiza los cálculos de la regresión lineal simple. Regresa los valores de la **ordenada al origen** y de la **pendiente**.

6.4.2.1.4 Modificaciones en la clase graficador.py

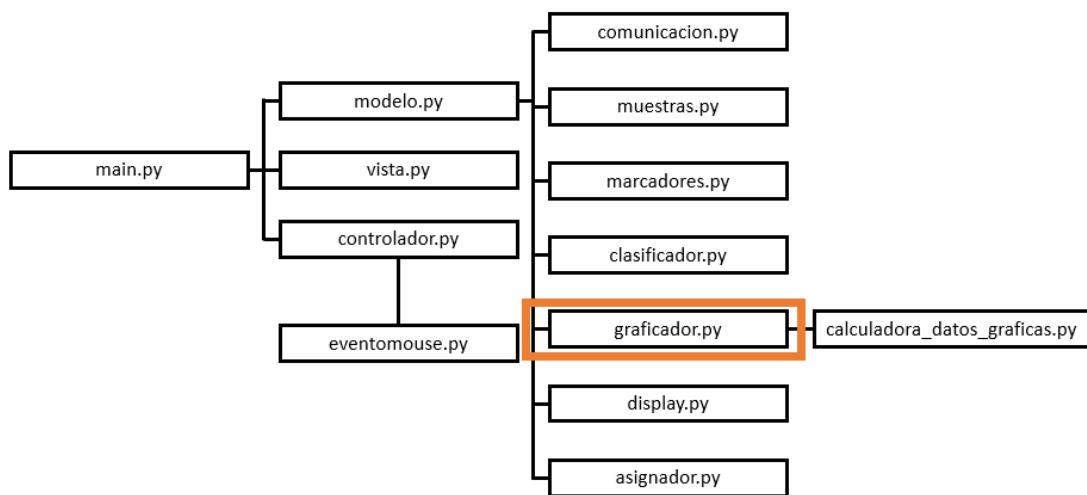


Figura 166. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

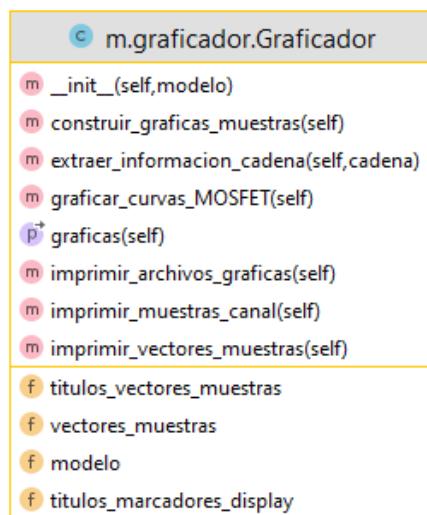


Figura 167. Métodos, atributos y propiedades de la clase graficador.py.

Donde:

-  indica que es una **clase**.
-  indica que es un **método**.
-  indica que es una **propiedad**.
-  indica que es un **atributo**.

De esta clase se presenta el método “graficar_curvas_MOSFET”. Este método se puede dividir en 8 partes principales:

- i. **Cálculo de datos.** Se calculan parámetros del transistor y datos necesarios para la obtención de las gráficas.
- ii. Código para la obtención de la **Gráfica 0: Voltajes en el tiempo**.
- iii. Código para la obtención de la **Gráfica 1: I_g vs V_{gs}** .
- iv. Código para la obtención de la **Gráfica 2: I_d vs V_{gs}** .
- v. Código para la obtención de la **Gráfica 3: I_d vs V_{ds}** .
- vi. Código para la obtención de la **Gráfica 4: g_m vs I_d** .
- vii. Código para la obtención de la **Gráfica 5: r_d vs I_d** .
- viii. Código para la obtención de la **Gráfica 6: Parámetros híbridos**.

6.4.2.1.4.1 Cálculo de datos (i)

```
def graficar_curvas_MOSFET(self):  
    global graficas  
    graficas = []  
  
    muestras = self.vectores_muestras  
    muestras_llaves = list(muestras)  
  
    # Son los indices de los canales de muestras recibidos como se difnio en la clase  
    modelo  
    ind_voltajes = [0,1,3,4] #Vgg, Vgs, Vdd, Vds  
    ind_corrientes = [2,5] # Ig, Id  
  
    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras  
    hay 6 curvas)  
    lim_max = len(muestras['Id'])  
    #print('lim_max: '+str(lim_max))  
    #print('muestras_llaves: '+str(len(muestras_llaves)))  
  
    # Seleccion entre las series de muestras sin filtro, con filtro circular o con filtro pasa-  
    bajas  
    # 0 = sin filtro  
    # 1 = con filtro circular cp  
    # 2 = con filtro pasa bajas lp  
  
    sel_filtro = 1
```

Figura 168. Fragmento de código para definir las variables a utilizar.

Se definen algunas variables útiles como:

- **muestras:** Es una copia del diccionario **vectores_muestras** que contiene el diccionario de las muestras tomadas de los 6 canales correspondientes a V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **muestras_llaves:** Contiene una lista de las llaves del diccionario **vectores_muestras**. Es decir, contiene los nombres V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d para identificar cada canal de muestras tomadas.
- **lim_max:** Contiene el número total de muestras tomadas por canal que es de **300 muestras**. En este caso, se tomaron muestras para 6 series de datos para cada canal de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **sel_filtro.** Indica cuál conjunto de datos se usará para todos los canales de muestras (**sin filtro (0)**, **con filtro circular (1)**, **con filtro pasa bajas (2)**).
- **ind_corrientes.** Indica para los 6 canales de muestras, cuáles son los **índices de canales de muestras de corrientes**.
- **ind_voltajes.** Indica de los 6 canales de muestras, cuáles son los **índices de los canales de muestras de voltajes**.

```

# Ajuste de Id con Rd = 220 ohm y de Ig con Rg = 100 Kohm

for i in range(ind_corrientes[1],lim_max*3,6):
    #print('Serie '+str(muestras_llaves[i])+' por ajustar')
    for j in range(0,lim_max):
        muestras[muestras_llaves[i]][j] = [x / 2200 for x in muestras[muestras_llaves[i]][j]]
        muestras[muestras_llaves[i-3]][j] = [x / 100000 for x in muestras[muestras_llaves[i-3]][j]]
    #print('Serie '+str(muestras_llaves[i])+' ajustada')

# Se obtienen los parametros lambda, Vth, Kn y gm.
calc = Calculadora_datos_graficas(self.modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes)

Vth, Kn, Idss, ind_ids = calc.calcular_Vth_Kn_Idss()
linea_lim_x,linea_lim_y,Vgs_prom = calc.calcular_puntos_límite_grafica_ids_vds(Vth)
MOS_lambdas = calc.calcular_lambda(linea_lim_y)
MOS_avg_lambda = -(1/statistics.mean(MOS_lambdas))
#print(MOS_avg_lambda)

gm = [0] * (lim_max-1)
corrientes_gm = [0] * (lim_max-1)

for i in range(0,lim_max-1):
    corrientes_gm[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][25]
    gm[i] = 2*math.sqrt(Kn*corrientes_gm[i])

#print('Parametros Híbridos')
# --- Parametros adicionales: rg, rd -----
-----
rg = float('inf')
#print(rg)

rd = [0] * (lim_max-1)

for i in range(0,lim_max-1):
    rd[i] = (1/(MOS_avg_lambda*corrientes_gm[i]))
    #print(rd[i])

```

Figura 169. Fragmento de código para el **cálculo los datos (i)** necesarios para las gráficas.

En esta parte del método se hace un ajuste de las corrientes I_d e I_g , puesto que son diferencias de voltajes. Ambas corrientes se dividen entre la resistencia correspondiente utilizada.

Se crea un **objeto** de la clase **calculadora_datos_graficas.py**, con la cual se realizan los cálculos de los parámetros del transistor como V_{th} , K_n , I_{dss} , g_m y **lambda**. Estos parámetros se calculan en la clase **calculadora_datos_graficas.py** detallada en las secciones 6.4.2.1.3.2 y 6.4.2.1.3.4.

6.4.2.1.4.2 Gráfica 0: Voltajes en el tiempo (ii)

En esta **gráfica 0** muestra una comparación de los **voltajes V_{gg} , V_{gs} , V_{dd} y V_{ds}** .

```
print('Grafica 0')
# --- Grafica 0 : Voltajes en el tiempo -----
-----
# --- Esta grafica es para observar el comportamiento de los voltajes en el tiempo
# --- Se observan los voltajes Vgg, Vgs, Vdd, Vds
plt.figure(0)

ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)

for i in ind_voltajes:
    arr = []
    for j in range(0,lim_max):
        arr = arr+muestras[muestras_llaves[i+6*sel_filtro]][j]
    plt.plot(arr, label=muestras_llaves[i+6*sel_filtro])

plt.xlabel('Muestras')
plt.ylabel('Voltaje')
plt.title("Voltajes en el tiempo")
plt.legend()
grafica = 'g/00_Voltajes' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 170. Fragmento de código para la **Gráfica 0: Voltajes en el tiempo**.

Muestra los **voltajes V_{gg} , V_{gs} , V_{dd} y V_{ds}** sobre una misma gráfica. Se seleccionan las muestras del conjunto con un filtro específico con la variable **sel_filtro**, como se detalla en la sección 6.4.2.1.2.

El resultado de esta parte es la siguiente gráfica:

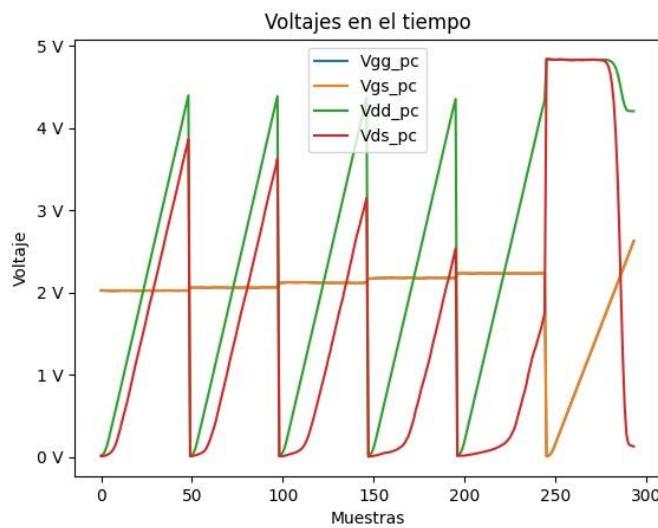


Figura 171. Gráfica 0: Voltajes en el tiempo.

Se observan los voltajes V_{gg} (azul), V_{gs} (naranja), V_{dd} (verde) y V_{ds} (rojo). Se tomaron **300 muestras** divididas en **6 series** de 50 muestras. Para este caso, se eligió el conjunto de muestras con filtro circular.

6.4.2.1.4.3 Gráfica 1: I_g vs V_{gs} (iii)

En esta **gráfica 1** se muestra una comparación entre la **corriente de compuerta I_g** y el **voltaje entre compuerta y fuente V_{gs}** .

```
print('Grafica 1')
# --- Grafica 1: Ig vs Vgs -----
-----
plt.figure(1)
plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1], muestras[muestras_llaves[ind_corrientes[0]+6*sel_filtro]][lim_max-1], label='Ig vs Vgs')

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

x1,x2,y1,y2=plt.axis()
plt.xlim(0,4)
plt.ylim(y1-y2*10,y2*1000)
plt.xlabel('Vgs')
plt.ylabel('Ig')
plt.subplots_adjust(left=0.15)
plt.title("Ig vs Vgs")
plt.legend()
grafica = 'g/01_Ig_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 172. Fragmento de código para la **Gráfica 1: I_g vs V_{gs}** .

El resultado de esta parte es la siguiente gráfica:

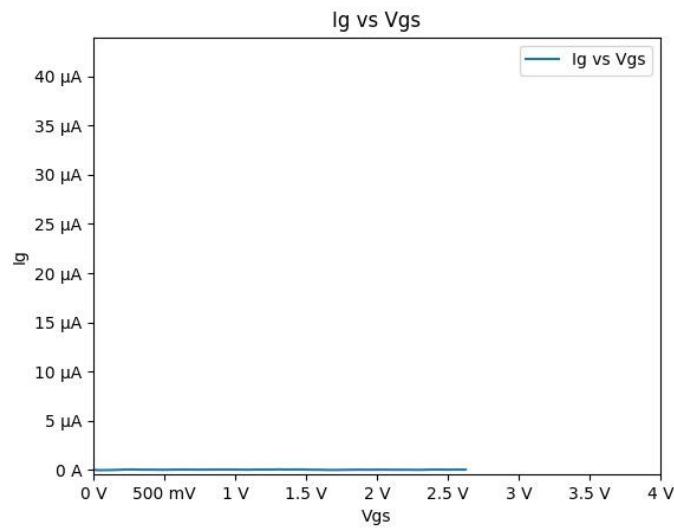


Figura 173. Gráfica 1: I_g vs V_{gs} .

Se muestra en azul la **curva I_g vs V_{gs}** .

6.4.2.1.4.4 Gráfica 2: I_d vs V_{gs} (iv)

En esta **gráfica 2** se muestra una comparación entre la **corriente de drenaje I_d** y el **voltaje entre compuerta y fuente V_{gs}** .

```
print('Grafica 2')
# --- Grafica 2: Id vs Vgs -----
-----
plt.figure(2)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][0:ind_ids[len(ind_ids)-1]], muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][0:ind_ids[len(ind_ids)-1]], label='Id vs Vgs')
x1,x2,y1,y2=plt.axis()
plt.axis([0,x2,y1,y2])
plt.xlabel('Vgs')
plt.ylabel('Id')
plt.title("Id vs Vgs")
plt.legend()
grafica = 'g/02_Id_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 174. Fragmento de código para la **Gráfica 2: I_d vs V_{gs}** .

El resultado de esta parte es la siguiente gráfica:

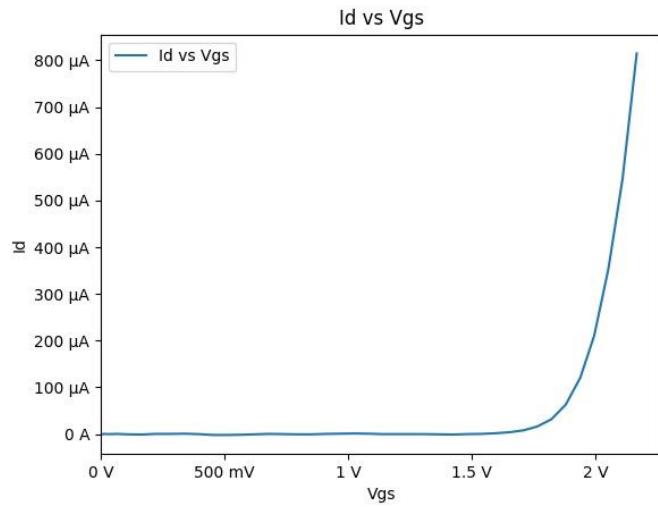


Figura 175. Gráfica 2: Id vs Vds.

En azul se observa la **curva I_g vs V_{gs}** .

6.4.2.1.4.5 Gráfica 3: I_d vs V_{ds} (v)

En esta **gráfica 3** se muestra una comparación entre la **corriente de drenaje I_d** y el **voltaje entre drenaje y fuente V_{ds}** . Para esta gráfica se tomaron **5 series** de datos para **5 curvas**, variando el **voltaje V_{gs}** para cada barrido del **voltaje V_{ds}** .

```
print('Grafica 3')
# --- Grafica 3: Id vs Vds (Cinco curvas) -----
-----
plt.figure(3)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

for i in range(0,lim_max-1):

    plt.plot(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i],muestras[muestras_llaves
    [ind_corrientes[1]+6*sel_filtro]][i],label='Vgs' =
    '+str(round(np.average(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][i]),3)))'

    plt.xlabel('Vds')
    plt.ylabel('Id')
    plt.title("Id vs Vds")
    plt.legend()
    grafica = 'g/03_Id_vs_Vds' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)
```

Figura 176. Fragmento de código para la **Gráfica 3: I_d vs V_{ds}** .

El resultado de esta parte es la siguiente gráfica:

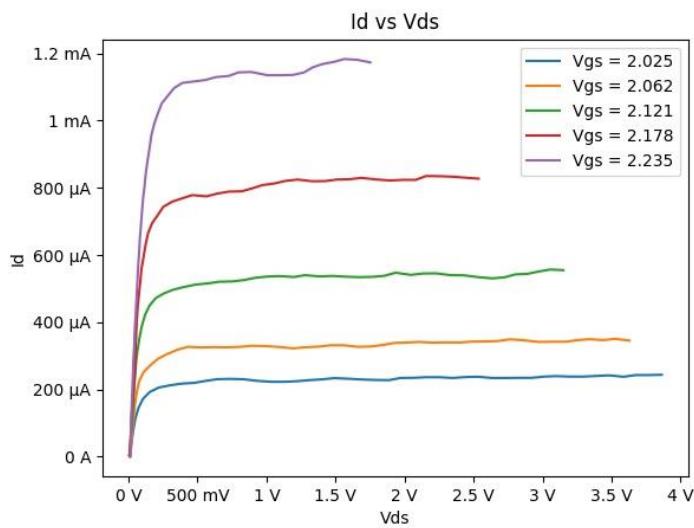


Figura 177. Gráfica 3: I_d vs V_{ds} .

Se observan las **5 curvas I_d vs V_{ds}** en azul, naranja, verde, rojo y morado. Para cada curva se especifica el voltaje V_{gs} utilizado.

6.4.2.1.4.6 Gráfica 4: gm vs Id (vi)

En esta **gráfica 4** se compara el parámetro de **transconductancia** g_m calculado con el valor correspondiente de **corriente de drenaje** I_d .

```
print('Grafica 4')
# --- Grafica 4: gm vs Id -----
-----
plt.figure(4)

ax = plt.gca()
formatter0 = EngFormatter(unit='\'\u03c3C3')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm, gm, 'o-', label='gm vs Id')

for i in range(0, len(gm)):

    plt.annotate(str(EngNumber(gm[i])) + '\u03c3C3', (corrientes_gm[i], gm[i]), textcoords="offset
    points", xytext=(0, 10), ha='center')
    x1, x2, y1, y2 = plt.axis()
    plt.axis([x1 - x1 / 8, x2 + x2 / 8, y1, y2 + y2 / 16])
    plt.xlabel('Id')
    plt.ylabel('gm')
    plt.title("gm vs Id")
    plt.legend()
    grafica = 'g/04_gm_vs_Id' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)
```

Figura 178. Fragmento de código para la **Gráfica 4: gm vs Id** .

El resultado de esta parte es la siguiente gráfica:

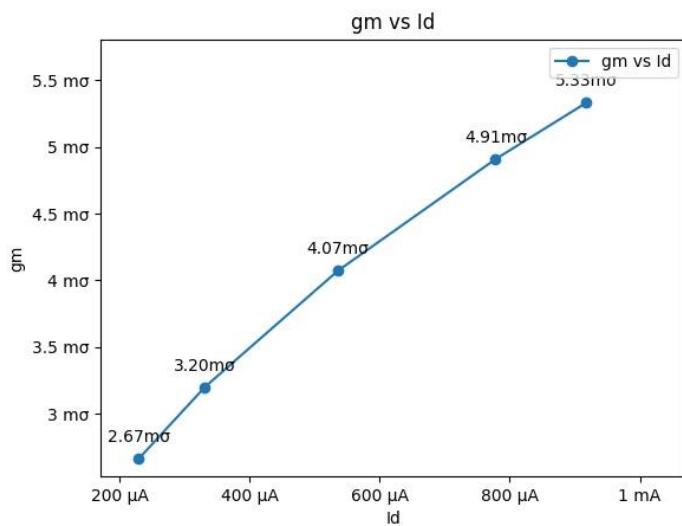


Figura 179. Gráfica 4: gm vs Id

Se observa en azul la **curva g_m vs I_d** y se marcan las cinco muestras correspondientes a las cinco curvas I_d vs V_{ds} .

6.4.2.1.4.7 Gráfica 5: rd vs Id (vii)

En esta **gráfica 5** se compara la **resistencia r_d** del modelo híbrido pi previamente calculada con el valor correspondiente de **corriente de drenaje I_d** .

```
print('Grafica 5')
# --- Grafica 5: rd vs Id -----
-----
plt.figure(5)

ax = plt.gca()
formatter0 = EngFormatter(unit='\u03a9')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm,rd,'o-', label='rd vs Id')
for i in range(0,len(rd)):

    plt.annotate(str(EngNumber(rd[i]))+'\u03a9',(corrientes_gm[i],rd[i]),textcoords="offset
    points",xytext=(0,10),ha='left')
    x1,x2,y1,y2 = plt.axis()
    plt.axis([x1,x2+x2/4,y1,y2+y2/4])
    plt.xlabel('Id')
    plt.ylabel('rd')
    plt.title("rd vs Id")
    plt.legend()
    grafica = 'g/05_rd_vs_Id' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)
```

Figura 180. Fragmento de código para la **Gráfica 5: rd vs Id** .

El resultado de esta parte es la siguiente gráfica:

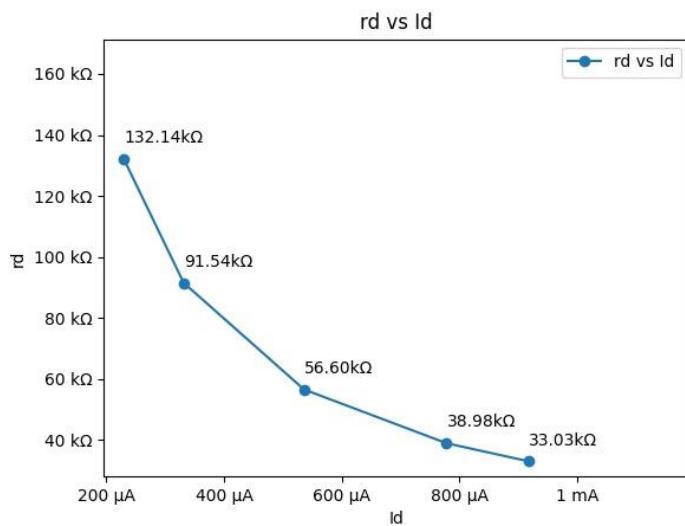


Figura 181. Gráfica 5: rd vs Id.

En azul se muestra la **curva r_d vs I_d** y se marcan las cinco muestras correspondientes a las cinco curvas I_d vs V_{ds} .

6.4.2.1.4.8 Gráfica 6: Parámetros híbridos (viii)

En esta **grafica 6** se muestra de forma condensada todos los **parámetros híbridos** calculados del transistor MOSFET. Estos parámetros incluyen:

- **Lambda**
- El **voltaje de umbral de encendido** V_{th} del transistor MOSFET
- Los parámetros I_{dss} , K_n y la **resistencia** r_d del modelo híbrido pi.

```
print('Grafica 6')
# --- Grafica 6 Resumen parametros hibridos-----
-----
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- RESUMEN PARAMETROS HIBRIDOS ---

cv2.putText(canvas2,"Transistor      MOSFET      2N7000", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"PARAMETROS      HIBRIDOS", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Lambda : " + str(EngNumber(MOS_avg_lambda)) + 'V^-1', (30,
130),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Voltaje de umbral de encendido : " + str(EngNumber(Vth)) + 'V',
(30, 150),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Idss : " + str(EngNumber(Idss)) + 'A', (30,
170),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Kn : " + str(EngNumber(Kn)) + 'A/V^2', (30,
190),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"rg : " + str(rg), (30, 210),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)

filename = "g/06_parametros_hibridos" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)
```

Figura 182. Fragmento de código para la **Gráfica 6: Parámetros híbridos**

El resultado de esta parte es la siguiente gráfica:

Transistor MOSFET 2N7000
PARAMETROS HIBRIDOS
Lambda : 32.97mV⁻¹
Voltaje de umbral de encendido : 1.84V
Idss : 26.30mA
Kn : 7.74mA/V²
rg : inf

Figura 183. Gráfica 6: Parámetros híbridos

Se muestra un resumen de los **parámetros híbridos** calculados del transistor.

6.4.2.2 Circuito generador de señal senoidal (b)

Ubicación de los circuitos bajo prueba propuestos:

- e. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- f. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- g. **Amplificador de una etapa (c)** con un transistor MOSFET.
- h. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

El diagrama de clases del programa en Python para el **circuito generador de señal senoidal (b)** es el siguiente:

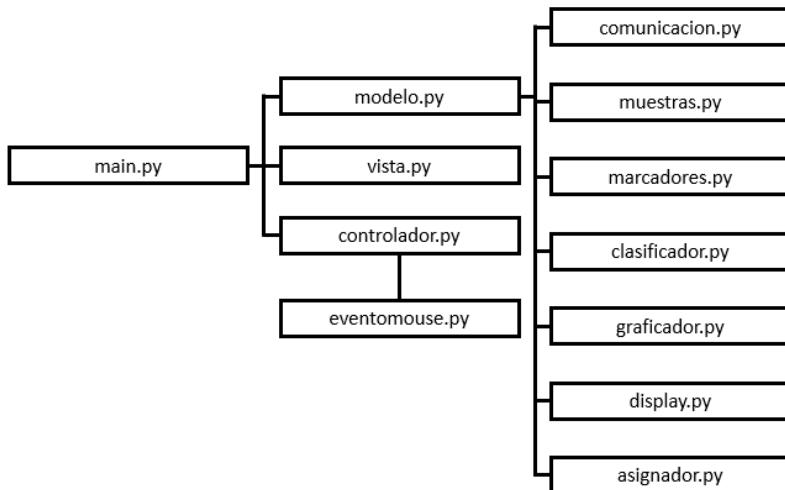


Figura 184. Diagrama de clases para el **circuito generador de señal senoidal (b)**.

6.4.2.2.1 Modificaciones en clase modelo.py

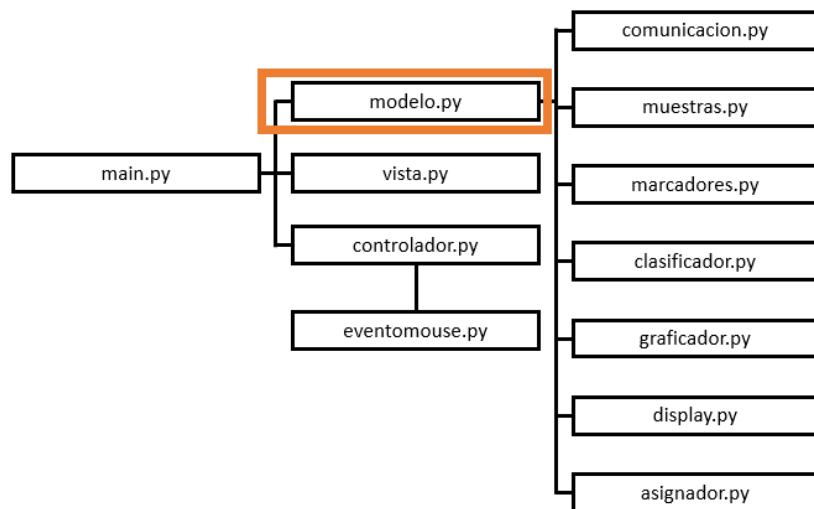


Figura 185. Ubicación de la clase modelo.py.

Esta clase contiene los siguientes métodos y atributos:



Figura 186. Métodos, atributos y propiedades del método modelo.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p⁺ indica que es una **propiedad**.
- f indica que es un **atributo**.

Las modificaciones realizadas en la clase **modelo.py** se realizaron sobre el método “**inicializar_objetos_modelo**”.

En la clase modelo en el método “**inicializar_obterjos_modelo**” se modifica lo siguiente:

```
titulos_vectores_muestras = ["Voltaje", "Binario"]
param_muestras = {'numMuestras': 100, 'numCanales':
len(titulos_vectores_muestras),
'llave': titulos_vectores_muestras}
```

Figura 187. Definición de los títulos de las llaves para los vectores de muestras y número de muestras

Se toman **100 muestras** para los **2 canales de muestras Voltaje y Binario**. Esto es que cada muestra de **voltaje a la salida** del circuito PCF8591 tiene un **valor en binario** asociado con el que fue generado.

```
titulos_marcadores_display = {'m0': "Voltaje en el tiempo",
'm1': "Tabla estados binarios",
'm2': "Voltaje-Binario (1)",
'm3': "Voltaje_binario (2)"}
```

Figura 188. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.

Se presentarán 4 gráficas, descritas en la sección 6.4.1.2.

6.4.2.2.2 Descripción de las variables de los datos recibidos desde el microcontrolador

La variable **vectores_muestras** contiene los 2 canales de muestras Voltaje y Binario. Es un diccionario de tres dimensiones. La variable **lim_max** contiene el número de muestras por canal. En este caso hay 100 muestras por canal.

Para acceder a cada uno de los datos se tiene que especificar:

- **Llave del canal de muestras:** Se indica entre comillas el nombre del canal de muestras a utilizar.
- **Serie de datos:** Cada canal de muestras tiene varias series de 50 muestras. Para acceder a cada serie de datos, se indica con un número entero entre 0 y el valor especificado por la expresión $\frac{\text{lim_max}}{50 \text{ muestras}}$. En este caso hay 2 series de datos de 50 muestras.
- **Número de la muestra o conjunto de muestras:** Se accede a una muestra específica, colocando un número de entre 0 y 50. También es posible acceder a un conjunto de muestras, colocando la muestra inicial, seguido de dos puntos e indicando la muestra final.

Por ejemplo, para acceder a las muestras del canal Voltaje, serie de datos 0 y muestras 4:25, se indica de la siguiente manera:

```
vectores_muestras['Voltaje'][0][4:25]
```

Para el programa del **circuito generador de señal senoidal (b)**, se toman los 2 canales de muestras Voltaje y Binario. Por cada canal se toman 2 series de datos de 50 muestras cada uno.

Tomando como ejemplo el canal V_{gg} , se tienen las siguientes series de muestras:

Serie de muestras	Voltaje
Serie 0 (muestras 0 a 49)	<code>vectores_muestras['Voltaje'][0]</code>
Serie 1 (muestras 50 a 99)	<code>vectores_muestras['Voltaje'][1]</code>

Tabla 26. Descripción de los canales de muestras.

A cada canal de muestras se le aplican dos filtros diferentes, un filtro circular y un filtro pasa bajas, guardando cada canal con filtro con un nombre diferente, mostrado en la tabla 27:

Canal de muestras	Nombre del canal de muestras		
	Sin filtro	Con filtro circular	Con filtro pasa bajas
Voltaje	Voltaje	Voltaje_pc	Voltaje_lp
Binario	Binario	Binario_pc	Binario_lp

Tabla 27. Descripción de los canales con filtro circular y pasa bajas.

La variable **muestras_llaves** contiene los títulos de las llaves de la variable **vector_muestras** mostrados en la tabla 27. El orden en que se guardan es el siguiente:

Orden	Canal de datos
0	Voltaje
1	Binario
2	Voltaje_pc
3	Binario_pc
4	Voltaje_lp
5	Binario_lp

Tabla 28. Orden de los títulos en la variable **muestras_llaves**.

Cada uno de los canales expuestos de la tabla 28 contienen 100 muestras cada uno. Se observa la variable **muestras_llaves** guarda los títulos de las llaves en el siguiente orden:

0. Títulos de las llaves de los canales **sin filtro**.
1. Títulos de las llaves de los canales **con filtro circular**.
2. Títulos de las llaves de los canales **con filtro pasa bajas**.

Usando la variable **sel_filtro** permite acceder de manera sencilla a estos canales, tomando valores de 0 a 2 en donde

0. Accede a los canales **sin filtro**.
1. Accede a los canales **con filtro circular**.
2. Accede a los canales **con filtro pasa bajas**.

6.4.2.2.3 Modificaciones en la clase graficador.py

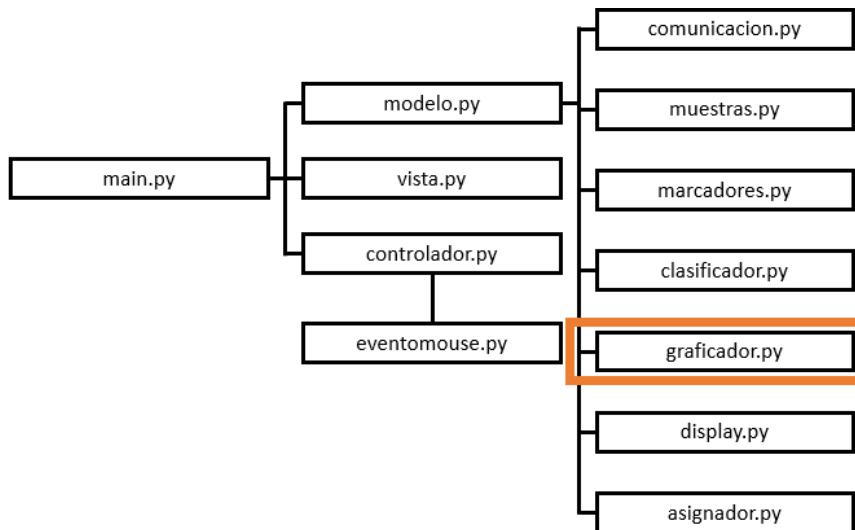


Figura 189. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

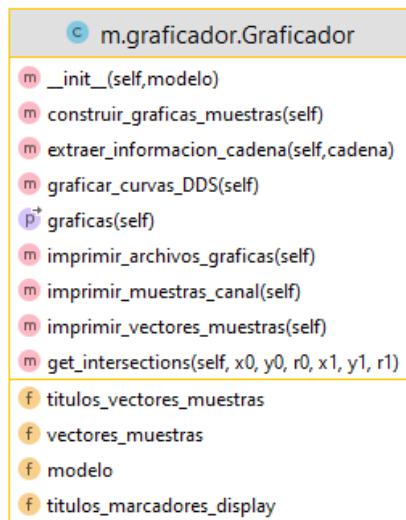


Figura 190. Métodos, atributos y propiedades de la clase graficador.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p+ indica que es una **propiedad**.
- f indica que es un **atributo**.

De esta clase se presenta el método “graficar_curvas_DDS”. Este método se puede dividir en 5 partes principales:

- i. **Definición de variables** útiles para la presentación de las gráficas.
- ii. Código para la obtención de la **Gráfica 0: Comparación entre voltaje y binario 1**.
- iii. Código para la obtención de la **Gráfica 1: Comparación entre voltaje y binario 2**.
- iv. Código para la obtención de la **Gráfica 2: Comparación entre voltaje y binario 3**.
- v. Código para la obtención de la **Gráfica 3: Comparación entre voltaje y binario 4**.

6.4.2.2.3.1 Definición de variables (i)

```
def graficar_curvas_DDS(self):  
    global graficas  
    graficas = []  
  
    muestras = self.vectores_muestras  
    titulos_vectores_muestras = self.titulos_vectores_muestras  
    muestras_llaves = list(muestras)  
    lim_max = len(muestras[muestras_llaves[0]])  
    sel_filtro = 1
```

Figura 191. Fragmento de código para definir las variables a utilizar.

Se definen algunas variables útiles como:

- **muestras:** Es una copia del diccionario **vectores_muestras** de las muestras tomadas de Arduino.
- **muestras_llaves:** Contiene una lista de las llaves del diccionario **vectores_muestras**.
- **lim_max:** Es la longitud de cada canal de muestras. En este caso es de 100 muestras.
- **sel_filtro:** Puesto que cada canal tiene tres posibles conjuntos de valores calculados que son **sin filtro (0)**, con **filtro circular (1)** y con **filtro pasa bajas (2)**.

6.4.2.2.3.2 Gráfica 0: Comparación entre voltaje y binario 1 (ii)

La **gráfica 0** muestra una gráfica comparativa entre el **valores en binario** enviado al PCF8591 y el **voltaje a la salida** de este circuito:

```
# --- Grafica 0 : Voltaje y binario 1 ---
plt.figure(0)

ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)

plt.subplot(2, 1, 1)
plt.title("Voltaje en el tiempo")
plt.plot(muestras['Voltaje'][0], label='Voltaje')
plt.ylabel('Voltaje')

plt.subplot(2, 1, 2)
plt.plot(muestras['Binario'][0], label='Binario')
plt.xlabel('Muestras')
plt.ylabel('Binario')

plt.legend()
grafica = 'g/00_Voltaje_Binario_1' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 192. Fragmento de código para la **Gráfica 0: Comparación entre voltaje y binario (1)**.

Muestra tanto el conjunto de muestras de voltaje como el de números binarios en dos gráficas.

El resultado de esta parte es la siguiente gráfica:

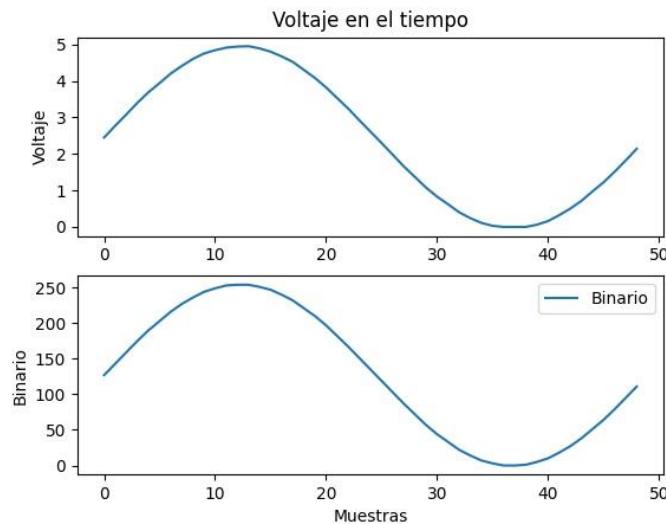


Figura 193. Gráfica 0: Comparación entre voltaje y binario (1).

Se muestra en azul la curva del **voltaje a la salida** del circuito PCF8591 en la parte superior de la gráfica y en azul la curva de **valores en binario** en la parte inferior.

6.4.2.2.3.3 Gráfica 1: Comparación entre voltaje y binario 2 (iii)

La **gráfica 1** muestra una tabla comparativa de algunos **valores en binario** enviados al PCF8591 y los **voltajes a la salida** de este circuito:

```
# --- Grafica 1 : Voltaje y binario 2 ---
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

cv2.putText(canvas2,"Tabla de estados binarios. Circuito DDS con PCF8591", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Numero          binario          (0-255)", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Voltaje ", (250, 100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0,
0), 1, cv2.LINE_AA)

# Generar tabla de valores voltaje-binario
for i in range(0,50,4):
    cv2.putText(canvas2,hex(int(muestras['Binario'][[0][i]])), (30,
100+(int(i/4)+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(muestras['Voltaje'][[0][i]]), (250,
100+(int(i/4)+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/01_Voltaje_Binario_2" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)
```

Figura 194. Fragmento de código para la **Gráfica 1: Comparación entre voltaje y binario (2).**

Se eligen algunos valores del conjunto de muestras y se coloca tanto el voltaje como el número en binario en la tabla.

El resultado de esta parte es la siguiente gráfica:

Tabla de estados binarios. Circuito DDS con PCF8591

Numero binario (0-255)	Voltaje
0x7f	2.45
0xbe	3.69
0xec	4.6
0xfe	4.94
0xf0	4.67
0xc5	3.82
0x87	2.61
0x48	1.37
0x16	0.4
0x0	0.0
0xa	0.16
0x33	0.97
0x6f	2.14

Figura 195. Gráfica 1: Comparación entre voltaje y binario (2).

Se muestra una tabla con algunos **valores en binario** a la izquierda y los valores de **voltaje a la salida** del circuito PCF8591 a la derecha.

6.4.2.2.3.3 Gráfica 2: Comparación entre voltaje y binario 3 (iv)

En la **gráfica 2** se colocan etiquetas con algunos de los **valores en binario** sobre la gráfica de **voltaje a la salida** del circuito PCF8591.

```
# --- Grafica 2 : Voltaje y binario 3 ---
plt.figure(2)
ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)
plt.title("Voltajes en el tiempo")

# Generar arreglo con indices de muestras a marcar
markers_on = []
for i in range(0,len(muestras['Voltaje'][0]),4):
    markers_on.append( i )
#print('markers_on=' +str(markers_on))
plt.plot(muestras['Voltaje'][0],'o-', markevery=markers_on, label='Voltaje')

# Colocar la etiqueta correspondiente segun se comporte la grafica
for i in range(1,len(muestras['Voltaje'][0]),4):
    position = ""
    xoffset = 0
    yoffset = 0
    if muestras['Voltaje'][0][i] > 2.5:
        if (muestras['Voltaje'][0][i] - muestras['Voltaje'][0][i-1]) < 0:
            #print('mod1')
            position = 'left'
            xoffset = 0
            yoffset = 0
        else:
            #print('mod2')
            position = 'right'
            xoffset = -12
            yoffset = 0
    else:
        if (muestras['Voltaje'][0][i-1] - muestras['Voltaje'][0][i]) < 0:
            #print('mod3')
            position = 'left'
            xoffset = 0
            yoffset = 0
        else:
            #print('mod4')
            position = 'right'
            xoffset = -10
            yoffset = -10
```

```

plt.annotate(str(muestras['Voltaje'][0][i-1])+'V,
'+hex(int(muestras['Binario'][0][i])),(i,muestras['Voltaje'][0][i-1]),textcoords="offset
points",xytext=(xoffset,yoffset),ha=position)

x1,x2,y1,y2=plt.axis()
plt.axis([-10,x2+3*x2/8,y1,5.5])
plt.ylabel('Voltaje')
plt.xlabel('Muestras')
plt.legend(['Voltaje, Binario'])
grafica = 'g/02_Voltaje_Binario_3' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

```

Figura 196. Fragmento de código para la **Gráfica 2: Comparación entre voltaje y binario (3)**.

Este método obtiene las coordenadas de la etiqueta a colocar dependiendo de la curva. Esto es para los casos en que se debe colocar la etiqueta abajo o arriba y a la derecha o izquierda del punto.

El resultado de esta parte es la siguiente gráfica:

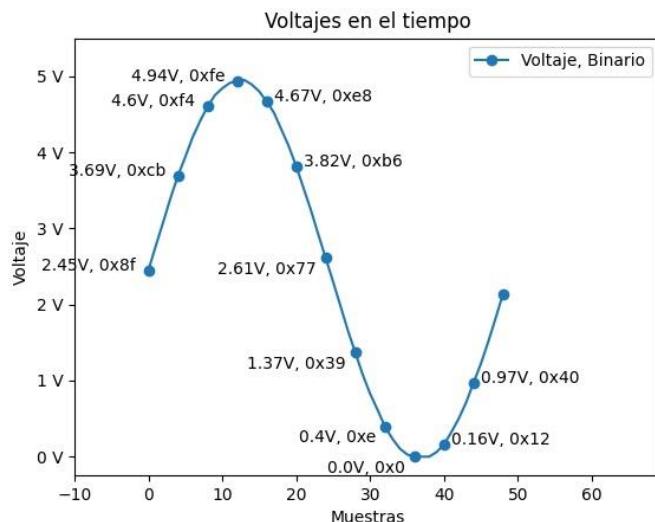


Figura 197. Gráfica 2: Comparación entre voltaje y binario (3).

Se muestra en azul la gráfica de **voltaje a la salida** del circuito PCF8591. Se seleccionan algunas muestras, las cuales se muestra tanto el **valor en binario** como el **voltaje a la salida** del circuito PCF8591.

6.4.2.2.3.4 Gráfica 3: Comparación entre voltaje y binario 4 (v)

En la gráfica 3 se comparan los **voltajes a la salida** del circuito PCF8591y el **valor en binario** en un diagrama.

```
# --- Grafica 3 : Voltaje y binario 4 ---
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

cv2.putText(canvas2,"Diagrama de numeros binarios con valores de voltaje analogicos", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)
#cv2.circle(canvas2, (320,260), 215,(255,0,0),2)
#cv2.circle(canvas2, (320,260), 185,(255,0,0),2)

angle = np.linspace(0,2*math.pi,16)

# Dibuja cada uno de los circulos y la informacion voltaje-binario que va dentro de ella.
Tambien dibuja las flechas entre circulos
for i in range(0,15):
    # Dibuja el circulo
    xcoor = int(320+185*math.cos(angle[i]))
    ycoor = int(260+185*math.sin(angle[i]))
    cv2.circle(canvas2, (xcoor,ycoor), 30, (0,0,0), 2)
    cv2.putText(canvas2,hex(int((muestras['Binario'][0][int((50/16)*i)]))), (xcoor-22,ycoor-5),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(muestras['Voltaje'][0][int((50/16)*i)])+'V', (xcoor-22,ycoor+15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    # Encuentra las coordenadas de la flecha entre dos circulos
    i1 = self.get_intersections(320,260,185,xcoor,ycoor,30)
    i2 = self.get_intersections(320,260,185,int(320+185*math.cos(angle[i+1])),int(260+185*math.sin(angle[i+1])),30)
    dist = 100
    flecha_coords = [0]*4
    ""for j in range(0,1):
        if math.dist([i1[j]*2,i1[j]*2+1],[i2[j]*2,i2[j]*2+1]) < dist:
            print("1, j="+str(j))
            dist = math.dist([i1[j]*2,i1[j]*2+1],[i2[j]*2,i2[j]*2+1])
            flecha_coords[0] = i1[j]*2
            flecha_coords[1] = i1[j]*2+1
            flecha_coords[2] = i2[j]*2
            flecha_coords[3] = i2[j]*2+1
```

```

if math.dist([i1[(2-j*2)],i1[(2-j*2)+1]],[i2[j*2],i2[j*2+1]]) < dist:
    print("2, j="+str(j))
    dist = math.dist([i1[j*2],i1[j*2+1]],[i2[j*2],i2[j*2+1]])
    flecha_coords[0] = i1[(2-j*2)]
    flecha_coords[1] = i1[(2-j*2)+1]
    flecha_coords[2] = i2[j*2]
    flecha_coords[3] = i2[j*2+1]"

# Se puede comprobar con el código comentado que esta es la solución que
proporciona las coordenadas correctas
flecha_coords[0] = i1[2]
flecha_coords[1] = i1[3]
flecha_coords[2] = i2[0]
flecha_coords[3] = i2[1]

#print('coordenadas de la flecha: '+str(flecha_coords))
#print('distancia: '+str(dist))

# Dibuja la flecha
canvas2 =
cv2.arrowedLine(canvas2,int(flecha_coords[0]),int(flecha_coords[1])),(int(flecha_coords[2])
),int(flecha_coords[3])),(0,0,0), 3, tipLength = 0.5)

filename = "g/03_Voltaje_Binario_4" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

```

Figura 198. Fragmento de código para la **Gráfica 3: Comparación entre voltaje y binario (4).**

Para esta gráfica, se colocan círculos con los **valores en binario y voltajes a la salida** del circuito PCF8591 escritos dentro de ellos. Estos círculos se distribuyen a lo largo de una circunferencia grande.

Lo que realiza este método para cada uno de los 16 valores seleccionados de voltaje y número en binario:

1. Se obtienen las coordenadas x y y de una circunferencia de radio grande del tamaño de la imagen.
2. Se obtienen las coordenadas x y y del centro de uno de los círculos, basado en la circunferencia grande en la que se colocará.
3. Se coloca el círculo junto con el **número en binario y voltaje a la salida** del circuito PCF8591 correspondiente.
4. Por último, se traza una flecha que apunta hacia el siguiente círculo en el que se pondrá el siguiente número binario y voltaje en la siguiente iteración.

El resultado de esta parte es la siguiente gráfica:

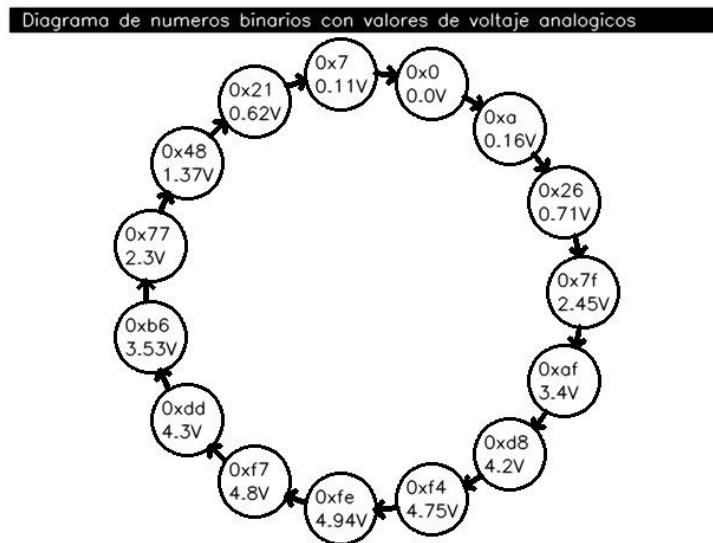


Figura 199. Gráfica 3: Comparación entre voltaje y binario (3).

Se muestra un diagrama con 15 círculos. En cada uno de estos círculos se muestra tanto el **valor en binario** como el **voltaje a la salida** del circuito PCF8591.

6.4.2.3 Amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el programa de Python del circuito **Amplificador de una etapa (c)**, se agregó la clase `calculadora_datos_graficas.py`, quedando el diagrama de clases de la siguiente manera:

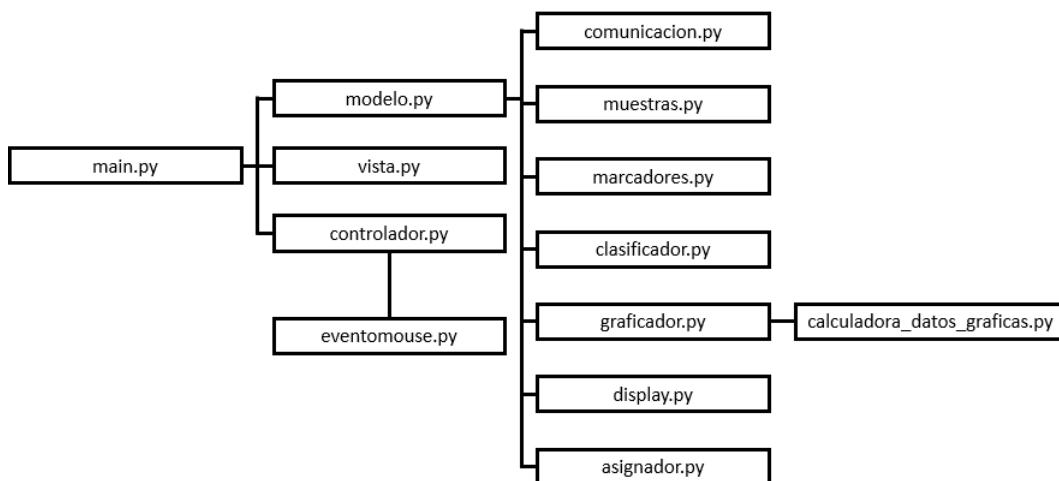


Figura 200. Diagrama de clases para el circuito **Amplificador de una etapa (c)**.

6.4.2.3.1 Modificaciones en la clase modelo.py

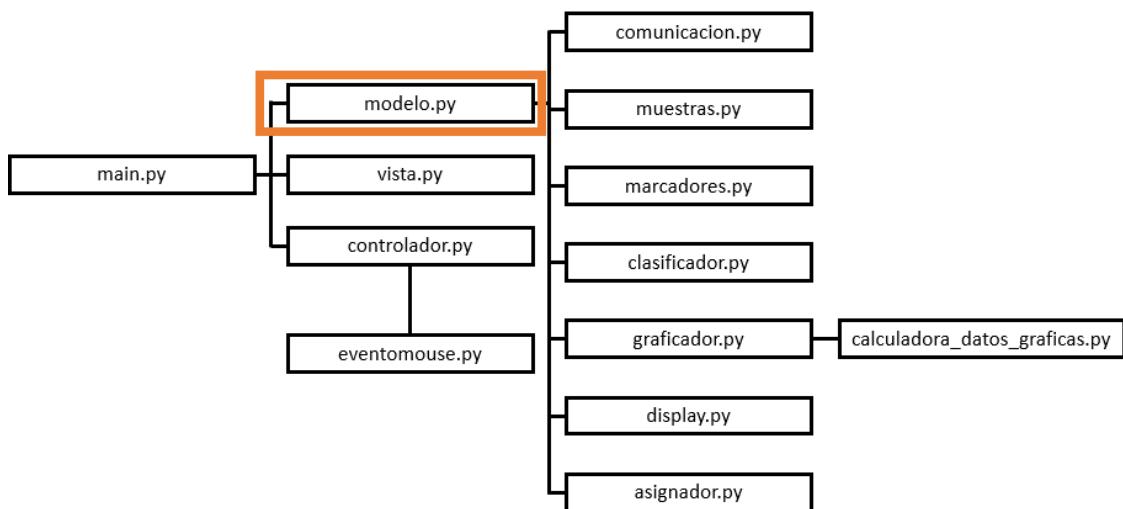


Figura 201. Ubicación de la clase modelo.py.

Esta clase contiene los siguientes métodos y atributos:



Figura 202. Métodos, atributos y propiedades del método modelo.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p⁺ indica que es una **propiedad**.
- f indica que es un **atributo**.

Las modificaciones realizadas en la clase **modelo.py** se realizaron sobre el método “**inicializar_objetos_modelo**”.

En la clase modelo en el método “**inicializar_obterjos_modelo**” se modifica lo siguiente:

```
titulos_vectores_muestras = ["Vgg","Vgs","Ig","Vdd","Vds","Id"]
param_muestras = {'numMuestras': 350, 'numCanales':
len(titulos_vectores_muestras),
'llave': titulos_vectores_muestras}
```

Figura 203. Definición de los títulos de las llaves para los vectores de muestras y número de muestras.

Se toman esta vez **350 muestras** para cada uno de los **6 canales de muestras** V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Hay que recordar que tanto I_g como I_d son la diferencia de voltajes V_{gg} con V_{gs} y V_{dd} con V_{ds} respectivamente.

De las 350 muestras:

- 250 muestras se tomaron para graficar las **curvas I_d vs V_{ds}** (5 series de 50 muestras para 5 **curvas I_d vs V_{ds}**).
- 50 muestras para las demás curvas (1 serie de 50 muestras para las **curvas I_g vs V_{gs} , I_d vs V_{gs} , g_m vs I_d , r_d vs I_d** y cálculo de **parámetros híbridos**).
- 50 muestras para la **curva de amplificación** que muestra el comportamiento del MOSFET como amplificador.

```
titulos_marcadores_display = {'m0': "Voltajes en el tiempo",
'm1': "Ig vs Vgs",
'm2': "Id vs Vgs",
'm3': "Id vs Vds",
'm4': "gm vs Id",
'm5': "rd vs Id",
'm6': "Amplificador. Voltajes",
'm7': "MOSFET 2N7000"}
```

Figura 204. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.

Se presentarán 8 gráficas, descritas en la sección 6.4.1.3.

6.4.2.3.2 Descripción de las variables de los datos recibidos desde el microcontrolador

La variable **vectores_muestras** contiene los 6 canales de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Es un diccionario de tres dimensiones. La variable **lim_max** contiene el número de muestras por canal. En este caso hay 350 muestras por canal.

Para acceder a cada uno de los datos se tiene que especificar:

- **Llave del canal de muestras:** Se indica entre comillas el nombre del canal de muestras a utilizar. Por ejemplo, para acceder las muestras del canal V_{gg} con filtro circular, se coloca 'Vgg_pc'.
- **Serie de datos:** Cada canal de muestras tiene varias series de 50 muestras. Para acceder a cada serie de datos, se indica con un número entero entre 0 y el valor especificado por la expresión $\frac{\text{lim_max}}{50 \text{ muestras}}$. En este caso hay 7 series de datos de 50 muestras. Por ejemplo, para seleccionar la serie 3 que contiene las muestras 101 a 150, se colocaría un 3.
- **Número de la muestra o conjunto de muestras:** Se accede a una muestra específica, colocando un número de entre 0 y 50. También es posible acceder a un conjunto de muestras, colocando la muestra inicial, seguido de dos puntos e indicando la muestra final. Por ejemplo, para acceder de la muestra 4 a 25 de la serie de datos 3, se coloca el rango 4:25.

Por ejemplo, para acceder a las muestras del canal V_{gg} con filtro circular, serie de datos 3 y muestras 4:25, se indica de la siguiente manera:

```
vectores_muestras['Vgg_pc'][3][4:25]
```

Para el programa del **circuito Amplificador de una etapa (c)**, se toman los 6 canales de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d . Por cada canal se toman 6 series de datos de 50 muestras cada uno.

Tomando como ejemplo el canal V_{gg} , se tienen las siguientes series de muestras:

Serie de muestras	V_{gg}
Serie 0 (muestras 0 a 49)	vectores_muestras['Vgg'][0]
Serie 1 (muestras 50 a 99)	vectores_muestras['Vgg'][1]
Serie 2 (muestras 100 a 149)	vectores_muestras['Vgg'][2]
Serie 3 (muestras 150 a 199)	vectores_muestras['Vgg'][3]
Serie 4 (muestras 200 a 249)	vectores_muestras['Vgg'][4]
Serie 5 (muestras 250 a 299)	vectores_muestras['Vgg'][5]
Serie 6 (muestras 300 a 349)	vectores_muestras['Vgg'][6]

Tabla 29. Descripción de los canales de muestras.

A cada canal de muestras se le aplican dos filtros diferentes, un filtro circular y un filtro pasa bajas, guardando cada canal con filtro con un nombre diferente, mostrado en la tabla 30:

Canal de muestras	Nombre del canal de muestras		
	Sin filtro	Con filtro circular	Con filtro pasa bajas
V_{gg}	Vgg	Vgg_pc	Vgg_lp
V_{gs}	Vgs	Vgs_pc	Vgs_lp
I_g	Ig	Ig_pc	Ig_lp
V_{dd}	Vdd	Vdd_pc	Vdd_lp
V_{ds}	Vds	Vds_pc	Vds_lp
I_d	Id	Id_pc	Id_lp

Tabla 30. Descripción de los canales con filtro circular y pasa bajas.

La variable **muestras_llaves** contiene los títulos de las llaves de la variable **vector_muestras** mostrados en la tabla 30. El orden en que se guardan es el siguiente:

Orden	Canal de datos
0	Vgg
1	Vgs
2	Ig
3	Vdd
4	Vds
5	Id
6	Vgg_pc
7	Vgs_pc
8	Ig_pc
9	Vdd_pc
10	Vds_pc
11	Id_pc
12	Vgg_lp
13	Vgs_lp
14	Ig_lp
15	Vdd_lp
16	Vds_lp
17	Id_lp

Tabla 31. Orden de los títulos en la variable `muestras_llaves`.

Cada uno de los canales expuestos de la tabla 31 contienen 350 muestras cada uno. Se observa la variable `muestras_llaves` guarda los títulos de las llaves en el siguiente orden:

0. Títulos de las llaves de los canales **sin filtro**.
1. Títulos de las llaves de los canales **con filtro circular**.
2. Títulos de las llaves de los canales **con filtro pasa bajas**.

Usando la variable **sel_filtro** permite acceder de manera sencilla a estos canales, tomando valores de 0 a 2 en donde

0. Accede a los canales **sin filtro**.
1. Accede a los canales **con filtro circular**.
2. Accede a los canales **con filtro pasa bajas**.

La variable **ind_corrientes** e **ind_voltajes** son arreglos de enteros que permiten acceder a los canales de voltajes y corrientes de manera sencilla, ya que contienen sus índices (con respecto al orden en que se reciben: V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d):

Orden en la variable ind_voltajes	0	1	2	3
Canal de datos	V_{gg}	V_{gs}	V_{dd}	V_{ds}
índice	0	1	3	4

Tabla 32. Orden de los índices de voltajes para la variable **ind_voltajes**.

Orden en la variable ind_corrientes	0	1
Canal de datos	I_g	I_d
índice	2	5

Tabla 33. Orden de los índices de corrientes para la variable **ind_corrientes**.

Se agregaron las siguientes variables arreglos de enteros para acceder de manera más sencilla las series correspondientes a cada curva:

- **num_curvas_id_vds**: Indica los índices de las series de datos utilizadas para calcular las **5 curvas I_d vs V_{ds}** . En este caso, incluye los índices de las primeras 5 series de datos: 0, 1, 2, 3 y 4. (**muestras 0 a 249**)
- **num_curvas_parámetros**: Indica los índices de las series de datos usadas para calcular las **curvas I_g vs V_{ds} , I_g vs V_{gs}** y para calcular los **parámetros híbridos**. En este caso, esta variable incluye el índice 5 de las series de datos. (**muestras 250 a 299**)
- **num_curvas_pequena_señal**: Indica los índices de las series de datos usadas para calcular la **curva de amplificación en fuente común**. En este caso, esta variable incluye el índice 6 de las series de datos. (**muestras 300 a 349**)

Por ejemplo, para acceder a las muestras del canal V_{gg} con filtro circular, serie de datos 3 (se utiliza la variable **num_curvas_id_vds**) y muestras 4:25, se indica de la siguiente manera (utilizando las variables **ind_voltajes**, **lim_max**, **sel_filtro**, **muestras_llaves**):

```
vectores_muestras[muestras_llaves_muestras[ind_voltajes[0]+sel_filtro*(lim_max/50)]][num_curvas_id_vds[3]][4:25]
```

En donde

```
sel_filtro = 1
lim_max = 350
```

6.4.2.3.3. Clase calculadora_datos_graficas.py

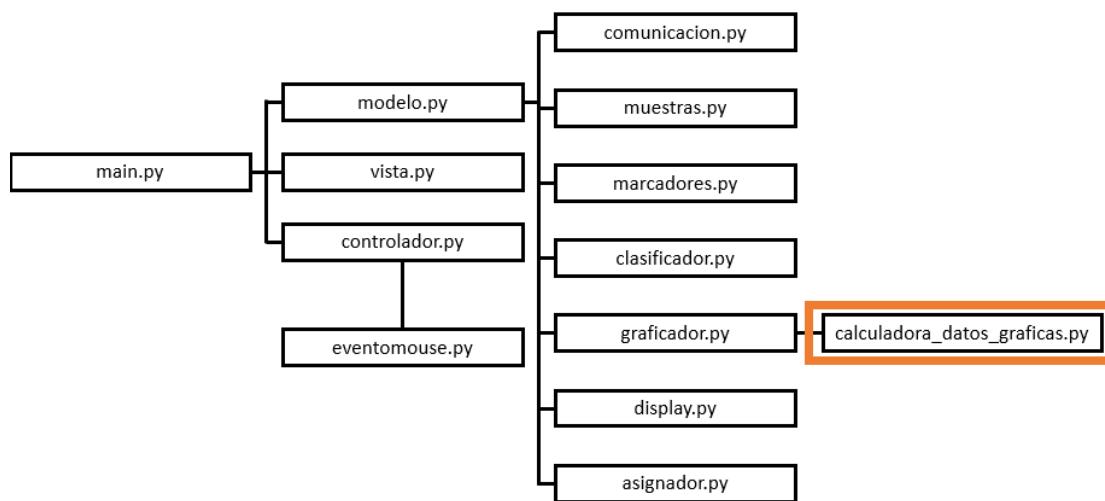


Figura 205. Ubicación de la clase `calculadora_datos_gráficas.py`.

Esta clase contiene los siguientes métodos y atributos:

```
  m.calculadora_datos_graficas.Calculadora_datos_graficas
m __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes,num_curvas,id_vds,num_curvas_parametros,num_curvas_pequena_senal,Rd,Rg)
m estimate_coef(self, x, y)
m plot_regression_line(self, x, y, b)
m construir_series_datos(self)
m obtener_ind_min(self,ind_serie_datos)
m obtener_ind_max(self,ind_serie_datos,ind_min)
m regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max)
m inicializar_excel_datos(self)
m guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId, voltajeVds)
m finalizar_excel_datos(self,writer)
m leer_datos_excel(self,ind_variables, i)
m calcular_lambda(self, linea, lim_y)
m calcular_Vth_Kn_ids(self)
m ajustar_id_con_resistencias(self)
m calcular_gm(self, Kn)
m calcular_resistencias_hibridas(self,MOS_avg_lambda,corrientes,gm)
m puntos_sobre_grafica_ids,ygs(self, Vth, ind_ids)
m calcular_puntos_límite_grafica_ids_vds(self, Vth)
m calcular_punto_operacion_y_parametros_hibridos(self,Kn,MOS_avg_lambda)
m calcular_punto_operacion_puntos_min_max_ganancia_experimental(self)
m calcular_puntos_límite_y_operacion_sobre_grafica_ids_vds(self,x,y,v,op,i,op)
m calcular_posiciones_label(self,ind_serie,ind_muestra)
f num_curvas_id_vds
f muestras_llaves
f Rd
f Rg
f num_curvas_pequena_senal
f sel_filtro
f ind_voltajes
f ind_corrientes
f muestras
f num_curvas_parametros
f modelo
f lim_max
```

Figura 206. Métodos, atributos y propiedades de la clase calculadora_datos_graficas.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p⁺ indica que es una **propiedad**.
- f indica que es un **atributo**.

Se presentan los siguientes métodos:

1. **Método constructor (1).** Se presenta el método “`__init__`”
2. **Método para realizar ajuste de corrientes (2).** Se presenta el método “`ajustar_id_con_con_resistencias`”.
3. **Método para calcular V_{th} , K_n e I_{dss} (3).** Se presenta el método “`calcular_Vth_Kn_Idss`”.
4. **Método para calcular puntos límite en la gráfica I_d vs V_{gs} (4).** Se presenta el método “`puntos_sobre_grafica_ids_vgs`”.
5. **Método para calcular puntos límite en la gráfica I_d vs V_{ds} (5).** Se presenta el método “`calcular_puntos_limite_grafica_ids_vds`”.
6. **Método para calcular g_m (6).** Se presenta el método “`calcular_gm`”.
7. **Métodos para calcular lambda (7).** Se presentan los siguientes métodos:
 - 4a. **Método “calcular_lambda” (7a).**
 - 4b. **Métodos “obtener_ind_min” y método “obtener_ind_max” (7b).**
 - 4c. **Método “regresión_lineal_lambda” (7c).**
 - 4d. **Método “estimate_coef” (7d).**
8. **Método para calcular resistencias del modelo híbrido pi (8).** Se presenta el método “`calcular_resistencias_hibridas`”.
9. **Método para calcular parámetros del punto de operación y parámetros híbridos asociados (9).** Se presenta el método “`calcular_punto_operacion_y_parametros_hibridos`”.
10. **Método para calcular punto de operación, máximos y mínimos sobre la gráfica que muestra el comportamiento del transistor como amplificador y ganancia experimental (10).** Se presenta el método “`calcular_punto_operacion_puntos_min_max_ganancia_experimental`”.
11. **Método para calcular puntos que delimitan regiones de operación y puntos sobre la recta de carga estática sobre las curvas I_d vs V_{ds} (11).** Se presenta el método “`calcular_puntos_limite_y_operacion_sobre_grafica_ids_vds`”.

6.4.2.3.3.1 Constructor (1)

```
def
__init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes,num_c
urvas_id_vds,num_curvas_parametros,num_curvas_pequena_señal,Rd,Rg):

    print("")
    print(" CONSTRUCTOR: Clase: Calculadora_datos_graficas")
    self.modelo = modelo
    self.muestras = modelo.vectores_muestras
    self.muestras_llaves = muestras_llaves
    self.lim_max = lim_max
    self.sel_filtro = sel_filtro
    self.ind_corrientes = ind_corrientes
    self.ind_voltajes = ind_voltajes
    self.num_curvas_id_vds = num_curvas_id_vds
    self.num_curvas_parametros = num_curvas_parametros
    self.num_curvas_pequena_señal = num_curvas_pequena_señal
    self.Rd = Rd
    self.Rg = Rg
```

Figura 207. Método constructor.

En el constructor se traen todas las variables importantes para los cálculos. Estas son:

- **muestras:** Es una copia del diccionario **vectores_muestras** que contiene el diccionario de las muestras tomadas de los 6 canales correspondientes a V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **muestras_llaves.** Contiene un arreglo de los nombres de las llaves del diccionario “**vectores_muestras**”. Es decir, contiene los nombres V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d para identificar cada canal de muestras tomadas.
- **lim_max.** Contiene el número total de muestras tomadas por canal que es de **350 muestras**. En este caso, se tomaron muestras para 7 series de datos para cada canal de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **sel_filtro.** Indica cuál conjunto de datos se usará para todos los canales de muestras (**sin filtro (0)**, con **filtro circular (1)**, con **filtro pasa bajas (2)**).
- **ind_corrientes.** Indica para los 6 canales de muestras, cuáles son los **índices de canales de muestras de corrientes**.
- **ind_voltajes.** Indica de los 6 canales de muestras, cuáles son los **índices de los canales de muestras de voltajes**.
- **num_curvas_id_vds.** Indica los **índices** de las series de cada conjunto de datos corresponden a **las curvas I_d vs V_{ds}** . Como se toman 5 series de 50 muestras para cada curva I_d vs V_{ds} , se toman en total las primeras 250 muestras para estas curvas (**muestras 0 a 249**). Por lo tanto, contiene los índices 0, 1, 2, 3 y 4.
- **num_curvas_parametros.** Indica los **índices** de las series de cada conjunto de datos corresponden para calcular las **curvas I_g vs V_{ds} , I_g vs V_{gs}** y los **parámetros**

híbridos del transistor MOSFET. Se tomó 1 serie de 50 muestras para el cálculo de parámetros, correspondientes a las **muestras 250 a 299**. (índice 5)

- **num_curvas_pequena_señal.** Indica cuáles series de cada conjunto de datos corresponden para calcular la **curva de amplificación en fuente común** del transistor MOSFET. Se tomó 1 serie de 50 muestras para el cálculo de parámetros, correspondientes a las **muestras 300 a 349**. (índice 6)
- **Rd y Rg.** Contiene el valor de las resistencias conectadas al transistor MOSFET.

6.4.2.3.3.2 Método para realizar ajuste de corrientes (2)

```
# -----
# Ajusta el valor de todas las corrientes, puesto que solo son una diferencia de voltajes
(ej. Id=Vdd-Vds ; Ig=Vgg-Vgs)
#
# -----
def ajustar_id_con_resistencias(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    ind_corrientes = self.ind_corrientes
    lim_max = self.lim_max
    Rd = self.Rd
    Rg = self.Rg

    for i in range(ind_corrientes[1],lim_max*3,6):
        #print('Serie '+str(muestras_llaves[i])+' por ajustar')
        for j in range(0,lim_max):
            muestras[muestras_llaves[i]][j] = [x / Rd for x in muestras[muestras_llaves[i]][j]]
            muestras[muestras_llaves[i-3]][j] = [x / Rg for x in muestras[muestras_llaves[i-3]][j]]
    #print('Serie '+str(muestras_llaves[i])+' ajustada')
```

Figura 208. Método “ajustar_id_con_resistencias”.

Este método realiza el **ajuste de las corrientes I_d e I_g** con las respectivas **resistencias R_d y R_g** , puesto que estas corrientes son diferencias de voltajes.

6.4.2.3.3.3 Método para calcular V_{th} , K_n e $Idss$ (3)

Este método realiza lo mismo que el método descrito en la sección 6.4.2.1.3.2, con la única diferencia que ahora hace uso de la variable `num_curvas_parametros` para indicar que realiza los cálculos con esa serie de 50 muestras.

6.4.2.3.3.4 Método para calcular gm (4)

```
# -----
# Calcula gm a partir de Kn para cada curva ids-vds
# Ademas, regresa los valores de corrientes utilizados para cada calculo de gm
# -----
def calcular_gm(self, Kn):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    num_curvas_id_vds = self.num_curvas_id_vds

    gm = [0] * len(num_curvas_id_vds)
    corrientes_gm = [0] * len(num_curvas_id_vds)

    # --- Se calcula gm para cada corriente elegida del conjunto de curvas ids-vds ---
    for i in num_curvas_id_vds:
        corrientes_gm[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][25]
        gm[i] = 2*math.sqrt(Kn*corrientes_gm[i])

    return (gm,corrientes_gm)
```

Figura 209. Método “calcular_gm”.

Este método calcula el parámetro de **transconductancia** g_m para cada conjunto de curvas I_d vs V_{ds} . Implementa la siguiente ecuación:

$$g_m = 2\sqrt{K_n * I_d}$$

Donde el valor de la corriente I_d se toma como la muestra 25 de cada serie de 50 muestras para cada curva I_d vs V_{ds} . Esta muestra está sobre la parte recta de la curva I_d vs V_{ds} .

6.4.2.3.3.5 Métodos para calcular puntos sobre gráfica Id vs Vgs (5)

```

# -----
# Calcula los puntos a graficar sobre las curvas vgs-ids
# Calcula la posicion de tres puntos: vth y dos puntos utilizados para calcular Kn,Vth y
Idss
# -----
def puntos_sobre_grafica_ids_vgs(self, Vth, ind_ids):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_VOLTajes = self.ind_VOLTajes
    num_curvas_parametros = self.num_curvas_parametros

    markers_on = []

    # --- Se obtiene el indice mas cercano al valor Vth previamente calculado ---
    for i in range(0,50):
        if muestras[muestras_llaves[ind_VOLTajes[1]+6*sel_filtro]][num_curvas_parametros[0]][i] <= 0:
            ind_vth = i
            break
    # --- Se arma el arreglo con los tres puntos a graficar sobre la curva ids-vgs ---
    markers_on.append(ind_vth)
    markers_on.append(ind_ids[0])
    markers_on.append(ind_ids[len(ind_ids)-1])

    vgs = []*len(markers_on)
    ids = []*len(markers_on)

    print('markers_on: '+str(markers_on))#, ' 1: '+str(markers_on[0]))
    #print('len(markers_on): '+str(len(markers_on)))

    # --- Se obtienen las coordenadas de los puntos anteriores (vgs,ids)---
    for (i,j) in zip(markers_on,range(0,len(markers_on))):

        vgs.append(muestras[muestras_llaves[ind_VOLTajes[1]+6*sel_filtro]][num_curvas_parametros[0]][i])

        ids.append(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_parametros[0]][i])
        #print('vgs: '+str(vgs[int(j)]))
        #print('ids: '+str(ids[int(j)]))

    return (vgs, ids, markers_on)

```

Figura 210. Método “puntos_sobre_grafica_ids_vgs”.

Este método calcula sobre el conjunto de muestras de la curva I_d vs V_{gs} el **índice** de la muestra que corresponde al **voltaje V_{th}** (calculado previamente, descrito en la sección 6.4.2.3.3.3) y el **índice** de las dos muestras con las que se calcularon los **parámetros V_{th} , K_n e I_{dss}** . (descrito en la sección 6.4.2.3.3.3)

6.4.2.3.3.6 Método para calcular puntos límite en la gráfica Id vs Vds (6)

Este método realiza lo mismo que el método descrito en la sección 6.4.2.1.3.3.

6.4.2.3.3.7 Métodos para calcular lambda (7)

El conjunto de métodos para esta parte realiza lo mismo que los métodos descritos en la sección 6.4.2.1.3.4 aunque con diferencias mínimas en el código ya que utiliza la variable **num_curvas_id_vds**.

6.4.2.3.3.8 Método para calcular resistencias del modelo híbrido pi (8)

```
# -----
# Calcula las resistencias rg y rd del modelo híbrido pi del transistor MOSFET
#
# -----
def calcular_resistencias_hibridas(self,MOS_avg_lambda,corrientes_gm):
    num_curvas_id_vds = self.num_curvas_id_vds

    rg = float('inf')
    print('rg: '+str(rg))

    rd = [0] * len(num_curvas_id_vds)

    # --- Se calcula rd para cada conjunto de curvas ids-vds ---
    for i in num_curvas_id_vds:
        rd[i] = (1/(MOS_avg_lambda*corrientes_gm[i]))
        #print(rd[i])
    print('rd: '+str(rd))
    return (rd,rg)
```

Figura 211. Método “calcular_resistencias_hibridas”.

Este método calcula el valor de las resistencias r_g y r_d del modelo híbrido pi. En el caso de la resistencia r_d , se calcula su valor para cada conjunto de curvas I_d vs V_{ds} .

6.4.2.3.3.9 Método para calcular parámetros del punto de operación y parámetros híbridos asociados (9)

```

# -----
# Calcula todos los parametros del punto de operacion del transistor MOSFET cuando
funciona como amplificador de pequena senal
# Calcula Vrd Id gm rd rlac=(Rd||rd) en el punto de operacion (v_op,i_op)
# Calcula tambien la ganancia teorica esperada
# -----
def calcular_punto_operacion_y_parametros_hibridos(self,Kn,MOS_avg_lambda):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_pequena_senal = self.num_curvas_pequena_senal
    Rd = self.Rd

    # --- Se obtiene el punto de operacion del amplificador. Es la primera muestra del
    # conjunto de curvas para pequena senal ---
    v_op =
    muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][num_curvas_pequena_senal[0]][0]
    i_op =
    muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_pequena_senal[0]][0]
    print('v_op: '+str(v_op))
    print('i_op: '+str(i_op))

    # --- Se calculan los parametros hibridos en el punto de operacion ---
    Vrdq =
    muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_senal[0]][0]
    - v_op
    Idq = Vrdq/Rd
    gmq = 2*math.sqrt(Kn*Idq)
    rdq = 1/(MOS_avg_lambda*Idq)
    Rlac = 1/((1/Rd)+(1/rdq))
    ganancia_teorica = -gmq*Rlac
    return (v_op,i_op,Vrdq,Idq,gmq,rdq,Rlac,ganancia_teorica)

```

Figura 212. Método “calcular_punto_operación_y_parametros_hibridos”.

Este método realiza lo siguiente, usando la variable `num_curvas_pequeña_señal` que accede a la serie de datos que contiene el comportamiento del transistor como amplificador:

1. Calcula el **voltaje en el punto de operación** v_{op} y la **corriente del punto de operación** i_{op} en el que opera el transistor MOSFET.
2. Calcula el **voltaje** V_{Rdq} en la resistencia R_d de 2200Ω que está conectada en la terminal de drenaje del transistor MOSFET.
3. La **resistencia** r_{dq} del modelo híbrido pi en el punto de operación del transistor MOSFET.
4. Calcula la **corriente** I_{dq} con base en el voltaje V_{Rdq} y la resistencia R_d de 2200Ω .
5. La **resistencia equivalente** R_{lac} de salida del amplificador, utilizando la resistencia R_d de 2200Ω en paralelo con la resistencia híbrida r_{dq} .
6. La **ganancia teórica** calculada con la siguiente ecuación: $G = -g_{mq} * R_{lac}$

6.4.2.3.3.10 Método para calcular punto de operación, máximos y mínimos sobre la gráfica que muestra el comportamiento del transistor como amplificador y calcular ganancia experimental (10)

```
# -----
# Calcula la ganancia experimental (vout/vin) de acuerdo a la curva de voltajes en el
tiempo cuando el transistor MOSFET funciona como amplificador en pequeña señal
# Regresa tambien los indices correspondientes a los minimos y maximos de las senales
de entrada y de salida
# -----
def calcular_punto_operacion_puntos_min_max_ganancia_experimental(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes
    num_curvas_pequeña_señal = self.num_curvas_pequeña_señal

    v_max_ind = [0]*2
    v_min_ind = [0]*2
    v_max = [0]*2
    v_min = [0]*2
    v_pp = [0]*2
    marker_v = []

    # --- Se calculan los voltajes maximos y minimos de las senales de entrada (vgs) y
    # salida (vds) ---
    for i in [0,1]:
        v_max[i] =
        v_min[i] =
        np.max(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequeña
        _señal[0]])
```

```

    v_min[i] = np.min(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_senal[0]])

    v_max_ind[i] = muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_senal[0]].index(v_max[i])
    v_min_ind[i] = muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_senal[0]].index(v_min[i])

    v_pp[i] = v_max[i] - v_min[i]

    marker_v.append(v_min_ind[i])
    marker_v.append(v_max_ind[i])

# --- Se calcula la ganancia experimental como vout/vin ---
ganancia_experimental = -v_pp[1]/v_pp[0]
print('ganancia_experimental: '+str(ganancia_experimental))

return (ganancia_experimental,marker_v)

```

Figura 213. Método “calcular_punto_operacion_puntos_min_max_ganancia_experimental”.

Este método calcula los **voltajes máximos y mínimos** tanto de la **señal de entrada V_{gs}** como la **señal de salida V_{ds}** del conjunto de muestras correspondiente al comportamiento del transistor como amplificador. Con estos datos se calcula la ganancia experimental, como

$$G = -\frac{V_{salida_max} - V_{salida_min}}{V_{entrada_max} - V_{entrada_min}}.$$

Además, se calcula el **índice** de estos **4 voltajes mínimos y máximos (marker_v)**, que posteriormente se utilizarán para mostrarlas en la gráfica que muestra el comportamiento del transistor como amplificador.

6.4.2.3.3.11 Método para calcular puntos que delimitan regiones de operación y puntos sobre la recta estática sobre las curvas Id vs Vds (11)

```
# -----
# Calcula la posicion de los puntos que delimitan las regiones de operacion sobre las curvas ids-vds
# Calcula las ecuaciones de la recta de carga estatica y de la parabola que delimita las regiones de operacion del transistor MOSFET
# Calcula tambien la posicion del punto de operacion sobre la recta de carga estatica
# Calcula tambien la posicion del punto de maxima variacion simetrica sobre la recta de carga y otras intersecciones
# -----
def calcular_puntos_lmite_y_operacion_sobre_grafica_ids_vds(self,x,y,v_op,i_op):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes
    num_curvas_pequena_señal = self.num_curvas_pequena_señal
    Rd = self.Rd

    # --- Se hace regresion polinomial al conjunto de puntos que indican el limite entre regiones de operacion ---
    # --- Se observo que una curva de 3er grado predice acertadamente el comportamiento de este conjunto de puntos ---
    parabola_vsat = np.poly1d(np.polyfit(x, y, 3))
    print('parabola inicial: \n'+str(np.poly1d(parabola_vsat)))

    # --- Se resta la ecuacion de la recta de carga estatica a la parabola, esto es restar los coeficientes de los terminos independiente y x ( $x^2$  y  $x^3$  permanecen intactos) ---
    vdd_prom = sum(muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_señal[0]]) / len(muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_señal[0]])
    print('vdd_prom: '+str(vdd_prom))
    parabola_vsat[0] = parabola_vsat[0]-(vdd_prom/Rd)
    parabola_vsat[1] = parabola_vsat[1]-(-1/Rd)

    # --- Se obtienen las raices o soluciones de esta ecuacion de 3er grado. Estas soluciones son las intersecciones entre la curva que delimita regiones y la recta de carga estatica ---
    print('intersecciones: \n'+str(parabola_vsat.r))

    interseccion = []

    # --- Las raices validadas para este problema (calcular el punto de la maxima variacion simetrica) son las no complejas y positivas ---
    for i in parabola_vsat.r:
        if not np.iscomplex(i) and i>=0:
```

```

        interseccion.append(i)
print('intersecciones reales: '+str(interseccion))

# --- Se restaura la ecuacion de la curva, puesto que fue modificada anteriormente ---
parabola_vsat = np.poly1d(np.polyfit(x, y, 3))
#print('parabola sin modificar: '+str(np.poly1d(parabola_vsat)))

# --- Se calcula la ecuacion de la recta estatica ---
puntos_grafica = np.linspace(0,vdd_prom, 50)
recta_carga_estatica = np.polynomial.polynomial.Polynomial([vdd_prom/Rd,-1/Rd])

# --- Se convierten las intersecciones de numeros numpy complejos a numeros python
flotantes ---
inters = np.real(interseccion[0].item())

# --- Se calcula el punto de la maxima variacion simetrica ---
v_sim_max = ((vdd_prom - inters) / 2) + inters

print('Punto de maxima variacion simetrica: '+str(EngNumber(v_sim_max))+','
'+str(EngNumber(recta_carga_estatica(v_sim_max)))))

# --- Se arma el arreglo de estos tres puntos: punto de operacion, interseccion entre la
curva que delimita regiones y la recta estatica y el punto de la maxima variacion simetrica -
--
puntos_x = [v_op,inters,v_sim_max]
puntos_y = [i_op,recta_carga_estatica(inters),recta_carga_estatica(v_sim_max)] 

return (recta_carga_estatica,parabola_vsat,puntos_x,puntos_y,puntos_grafica)

```

Figura 214. Método “calcular_puntos_limite_y_operacion_sobre_grafica_ids_vds”.

Este método realiza lo siguiente:

1. Se calcula la **ecuación** de la **curva parabola_vsat** que delimita las regiones de operación utilizando los puntos calculados en el método de la sección 6.4.2.3.3.6.
2. Se calcula la **intersección** de la **curva parabola_vsat** con la **recta de carga estática**, al restar los coeficientes independiente y lineal de las ecuaciones de la curva parabola_vsat y los coeficientes de la recta de carga estática.
3. Se **restaura** la **ecuación** de la **curva parabola_vsat**, puesto que fue modificada en el paso anterior.
4. Se calcula la **ecuación** de **recta_carga_estatica** con los coeficientes para esta recta.
5. Se calcula el **punto de la máxima variación simétrica**, entre sobre la recta de carga estática entre los siguientes puntos:
 - Punto de la intersección de la curva parabola_vsat con la recta de carga estática
 - Punto de la intersección entre la recta de carga estática con el eje x (voltaje V_{ds}).

El método regresa los siguientes valores:

- **recta_carga_estatica** es la ecuación de la recta de carga estática.
- **parabola_vsat** es la ecuación de la curva que delimita las regiones de operación óhmica y de saturación del transistor MOSFET.
- **puntos_x** son las coordenadas x (**los voltajes**) asociadas a los siguientes puntos:
 - **v_op**: Voltaje del punto de operación.
 - **inters_v**: Voltaje del punto de intersección de curva parabola_vsat y de la recta recta_carga_estatica.
 - **v_sim_max**: Voltaje del punto de la máxima variación simétrica.
- **puntos_y** son las coordenadas y (**las corrientes**) asociadas a los siguientes puntos:
 - **i_op**: Corriente del punto de operación.
 - **recta_carga_estatica(inters)**: Corriente del punto de la intersección de curva parabola_vsat y de la recta recta_carga_estatica.
 - **recta_carga_estatica(v_sim_max)**: Corriente del punto de la máxima variación simétrica.

6.4.2.3.4 Modificaciones en la clase graficador.py

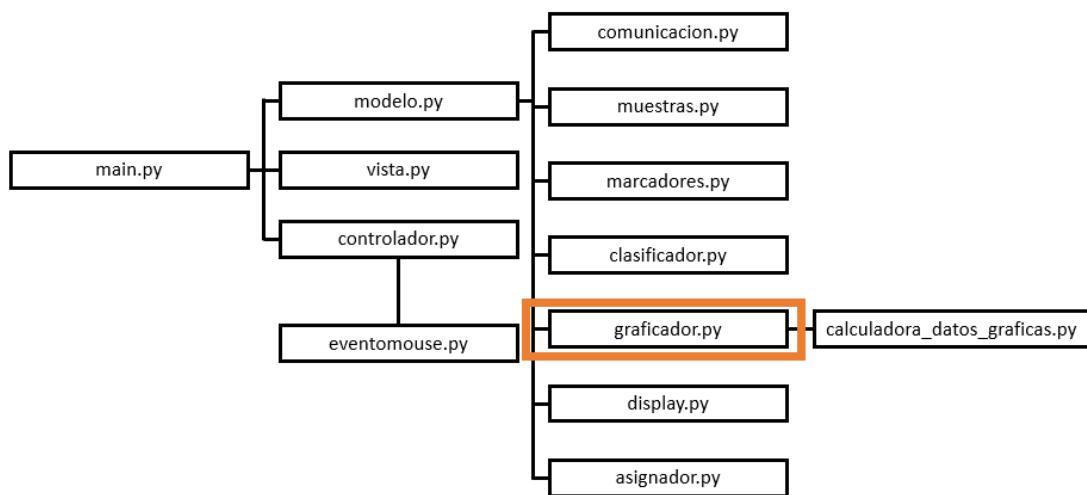


Figura 215. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

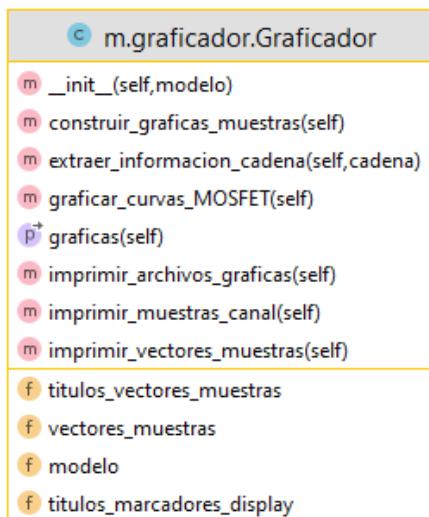


Figura 216. Métodos, atributos y propiedades de la clase graficador.py.

Donde:

- c** indica que es una **clase**.
- m** indica que es un **método**.
- p+** indica que es una **propiedad**.
- f** indica que es un **atributo**.

De esta clase se presenta el método “graficar_curvas_MOSFET”. Este método se puede dividir en 9 partes principales:

- i. **Cálculo de datos.** Se calculan parámetros del transistor y datos necesarios para la obtención de las gráficas.
- ii. Código para la obtención de la **Gráfica 0: Voltajes en el tiempo.**
- iii. Código para la obtención de la **Gráfica 1: I_g vs V_{gs} .**
- iv. Código para la obtención de la **Gráfica 2: I_d vs V_{gs} .**
- v. Código para la obtención de la **Gráfica 3: I_d vs V_{ds} .**
- vi. Código para la obtención de la **Gráfica 4: g_m vs I_d .**
- vii. Código para la obtención de la **Gráfica 5: r_d vs I_d .**
- viii. Código para la obtención de la **Gráfica 6: Voltajes del amplificador en fuente común.**
- ix. Código para la obtención de la **Gráfica 7: Parámetros híbridos.**

6.4.2.3.4.1 Cálculo de datos (i)

```
def graficar_curvas_MOSFET(self):  
    global graficas  
    graficas = []  
  
    muestras = self.vectores_muestras  
    muestras_llaves = list(muestras)  
  
    # Son los indices de los canales de muestras recibidos como se defnio en la clase  
    modelo  
    ind_voltajes = [0,1,3,4] #Vgg, Vgs, Vdd, Vds  
    ind_corrientes = [2,5] # Ig, Id  
  
    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras  
    hay 6 curvas)  
    lim_max = len(muestras['Id'])  
    #print('lim_max: '+str(lim_max))  
    #print('muestras_llaves: '+str(len(muestras_llaves)))  
  
    # Establecer el numero de curvas para cada conjunto de parametros  
    num_curvas_id_vds = [0,1,2,3,4]  
    num_curvas_parametros = [5]  
    num_curvas_pequena_señal = [6]  
  
    Rd = 2200  
    Rg = 100000  
  
    # Selección entre las series de muestras sin filtro, con filtro circular o con filtro pasa-  
    bajas  
    # 0 = sin filtro  
    # 1 = con filtro circular cp  
    # 2 = con filtro pasa bajas lp  
  
    sel_filtro = 1
```

Figura 217. Fragmento de código para definir las variables a utilizar.

Al principio del método se definen algunas variables útiles como:

- **muestras:** Es una copia del diccionario **vectores_muestras** que contiene el diccionario de las muestras tomadas de los 6 canales correspondientes a V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **muestras_llaves.** Contiene un arreglo de los nombres de las llaves del diccionario “**vectores_muestras**”. Es decir, contiene los nombres V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d para identificar cada canal de muestras tomadas.

- **lim_max.** Contiene el número total de muestras tomadas por canal que es de **350 muestras**. En este caso, se tomaron muestras para 7 series de datos para cada canal de muestras V_{gg} , V_{gs} , I_g , V_{dd} , V_{ds} e I_d .
- **sel_filtro.** Indica cuál conjunto de datos se usará para todos los canales de muestras (**sin filtro (0)**, con **filtro circular (1)**, con **filtro pasa bajas (2)**).
- **ind_corrientes.** Indica para los 6 canales de muestras, cuáles son los **índices de canales de muestras de corrientes**.
- **ind_voltajes.** Indica de los 6 canales de muestras, cuáles son los **índices de los canales de muestras de voltajes**.
- **num_curvas_id_vds.** Indica los **índices** de las series de cada conjunto de datos corresponden a **las curvas I_d vs V_{ds}** . Como se toman 5 series de 50 muestras para cada curva I_d vs V_{ds} , se toman en total las primeras 250 muestras para estas curvas (**muestras 0 a 249**). Por lo tanto, contiene los índices 0, 1, 2, 3 y 4.
- **num_curvas_parametros.** Indica los **índices** de las series de cada conjunto de datos corresponden para calcular las **curvas I_g vs V_{ds} , I_g vs V_{gs}** y los **parámetros híbridos** del transistor MOSFET. Se tomó 1 serie de 50 muestras para el cálculo de parámetros, correspondientes a las **muestras 250 a 299**. (índice 5)
- **num_curvas_pequena_señal.** Indica cuáles series de cada conjunto de datos corresponden para calcular la **curva de amplificación en fuente común** del transistor MOSFET. Se tomó 1 serie de 50 muestras para el cálculo de parámetros, correspondientes a las **muestras 300 a 349**. (índice 6)
- **Rd y Rg.** Se define el valor de las resistencias conectadas al transistor MOSFET.

```

calc =  

Calculadora_datos_graficas(self.modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes  

,ind_voltajes,num_curvas_id_vds,num_curvas_parametros,num_curvas_pequena_señal,  

Rd,Rg)

# Ajuste de Id con Rd = 220 ohm y de Ig con Rg = 100 Kohm  

calc.ajustar_id_con_resistencias()

# Se obtienen los parametros Vth, Kn y gm.  

Vth, Kn, Idss, ind_ids = calc.calcular_Vth_Kn_Idss()

gm, corrientes_gm = calc.calcular_gm(Kn)  

#print('corriente: '+str(corrientes_gm)+', gm: '+str(gm))

#print('Parametros Híbridos')

# --- Calculos para la grafica 2 -----
-----  

vgs, ids, markers_on = calc.puntos_sobre_grafica_ids_vgs(Vth,ind_ids)

# --- Calculos para la grafica 3 -----
-----  

# --- Limites de las regiones de operacion -----
-----  

linea_lim_x,linea_lim_y,Vgs_prom = calc.calcular_puntos_límite_grafica_ids_vds(Vth)

```

```

ids_min,ids_max,MOS_lambdas = calc.calcular_lambda(linea_lim_y)
MOS_avg_lambda = -(1/statistics.mean(MOS_lambdas))
#print('MOS_avg_lambda: '+str(MOS_avg_lambda))
# --- Parametros adicionales: rg, rd -----
-----
rd, rg = calc.calcular_resistencias_hibridas(MOS_avg_lambda,corrientes_gm)

# --- Punto de operacion -----
-----
v_op,i_op,Vrdq,ldq,gmq,rdq,Rlac,ganancia_teorica = =
calc.calcular_punto_operacion_y_parametros_hibridos(Kn,MOS_avg_lambda)

# --- Ganancia experimental -----
-----
ganancia_experimental,marker_v = =
calc.calcular_punto_operacion_puntos_min_max_ganancia_experimental()

# --- Coordenadas de los puntos de los limites de las regiones de operacion, recta de
carga estatica, punto de operacion -----
recta_carga_estatica,parabola_vsat,puntos_x,puntos_y,puntos_grafica = =
calc.calcular_puntos_límite_y_operacion_sobre_grafica_ids_vds(linea_lim_x,linea_lim_y,v
_op,i_op)

```

Figura 218. Fragmento de código para el **cálculo de datos (i)** necesarios para las gráficas.

Esta parte del método se realiza lo siguiente:

1. Se crea un **objeto** de la **clase calculadora_datos_graficas.py**, con la cual se calculan todos los datos y puntos a graficar.
2. Se realiza el **ajuste de corrientes I_d e I_g** , descrito en la sección 6.4.2.3.3.2.
3. Se calcula V_{th} , K_n e I_{dss} , descrito en la sección 6.4.2.3.2.3.
4. Se calcula g_m , descrito en la sección 6.4.2.3.3.4.
5. Se calculan **puntos sobre la gráfica I_d vs V_{gs}** , descrito en la sección 6.4.2.3.3.5.
6. Se realiza el cálculo de **puntos límite sobre la gráfica I_d vs V_{ds}** , descrito en la sección 6.4.2.3.3.6.
7. Se calcula **lambda**, descrito en la sección 6.4.2.3.3.7.
8. Se calcula **r_g y r_d** , descrito en la sección 6.4.2.3.3.8.
9. Se calculan parámetros del **punto de operación, parámetros híbridos asociados y ganancia teórica**, descrito en la sección 6.4.2.3.3.9.
10. Se calcula la **ganancia experimental**, así como **puntos sobre la gráfica que muestra el comportamiento del transistor como amplificador**, descrito en la sección 6.4.2.3.3.10.
11. Se calcula la **ecuación de la recta de carga estática, parábola que delimita regiones de operación y puntos sobre la recta de carga estática** en la gráfica de curvas I_d vs V_{ds} , descrito en la sección 6.4.2.3.3.11.

6.4.2.3.4.2 Gráfica 0: Voltajes en el tiempo (ii)

El código para generar esta **gráfica 0** es idéntico al mostrado en la sección 6.4.2.1.3.2. Muestra una comparación de los voltajes V_{gg} , V_{gs} , V_{dd} y V_{ds} .

El resultado de esta parte es la siguiente gráfica:

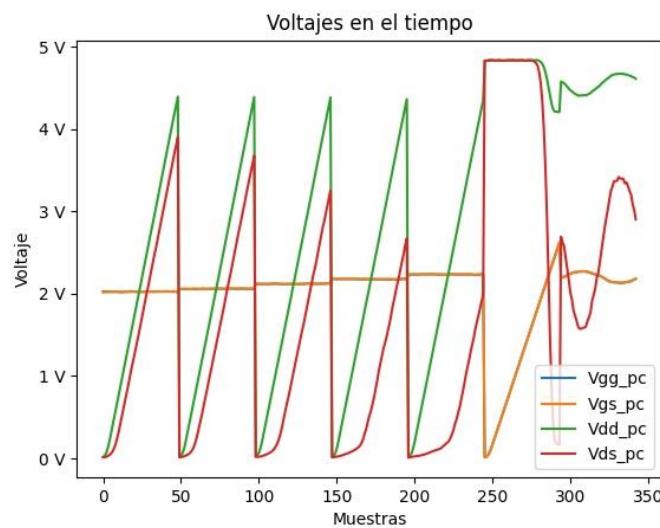


Figura 219. Gráfica 0: Voltajes en el tiempo.

Se observan los voltajes V_{gg} (azul), V_{gs} (naranja), V_{dd} (verde) y V_{ds} (rojo). Se tomaron 300 muestras divididas en 7 series de 50 muestras. Para este caso, se eligió el conjunto de muestras con filtro circular.

6.4.2.3.4.3 Gráfica 1: I_g vs V_{gs} (iii)

El código para generar esta **gráfica 1** es idéntico al mostrado en la sección 6.4.2.1.3.3. Muestra una comparación entre la **corriente de compuerta I_g** y el **voltaje entre compuerta y fuente V_{gs}** .

El resultado de esta parte es la siguiente gráfica:

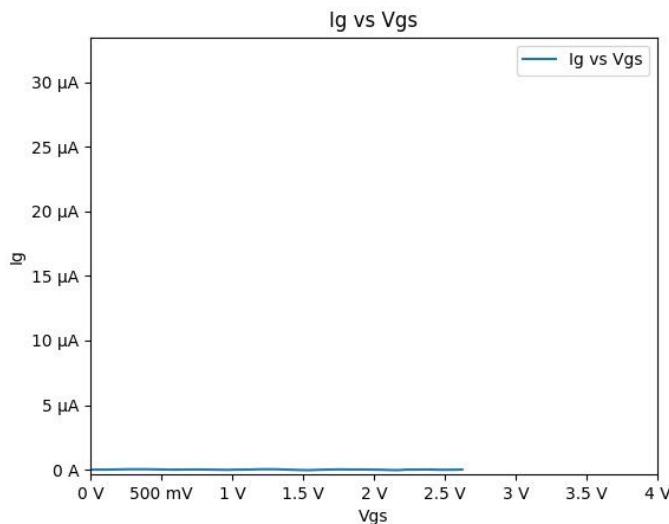


Figura 220. Gráfica 1: I_g vs V_{gs} .

Se muestra en azul la **curva I_g vs V_{gs}** .

6.4.2.3.4.4 Gráfica 2: I_d vs V_{gs} (iv)

El código es similar al presentado en la sección 6.4.2.1.3.4. En esta **gráfica 2** se muestra una comparación entre la **corriente de drenaje I_d** y el **voltaje entre compuerta y fuente V_{gs}** .

```
print('Grafica 2')
# --- Grafica 2: Id vs Vgs -----
-----
plt.figure(2)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][num_curvas_parametros[0]]
[0:(markers_on[len(markers_on)-1]+2)],muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_parametros[0]
][0:(markers_on[len(markers_on)-1]+2)],'o-', markevery=markers_on, label='Id vs Vgs')

for i in range(0,len(markers_on)):
    vth_str = ""
    if i == 0:
        vth_str = 'Vth: '
    plt.annotate(vth_str+str(EngNumber(vgs[i]))+'V',
    '+str(EngNumber(ids[i]))+'A',(vgs[i],ids[i]),textcoords="offset points",xytext=(-5,3),ha='right')

x1,x2,y1,y2=plt.axis()
plt.axis([0,x2,y1,y2])

plt.xlabel('Vgs')
plt.ylabel('Id')
plt.title("Id vs Vgs")
plt.legend()
grafica = 'g/02_Id_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 221. Fragmento de código para la **Gráfica 2: I_d vs V_{gs}** .

Esta gráfica, además muestra los puntos asociados al voltaje de umbral V_{th} y los puntos utilizados para calcular K_n , V_{th} e I_{dss} .

El resultado de esta parte es la siguiente gráfica:

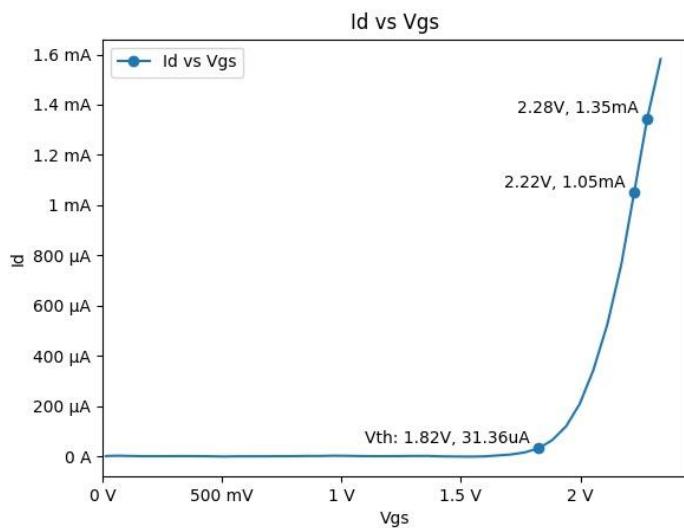


Figura 222. Gráfica 2: I_d vs V_{gs} .

En azul se observa la **curva I_d vs V_{gs}** . Sobre esta gráfica se marcan los **puntos del voltaje de umbral V_{th}** y **dos puntos utilizados para calcular K_n , V_{th} e I_{dss}** .

6.4.2.3.4.5 Gráfica 3: I_d vs V_{ds} (v)

El código es similar al presentado en la sección 6.4.2.1.4.5. En esta **gráfica 3** se muestra una comparación entre la **corriente de drenaje I_d** y el **voltaje entre drenaje y fuente V_{ds}** . Para esta gráfica se tomaron **5 series** de datos para **5 curvas**, variando el **voltaje V_{gs}** para cada barido del **voltaje V_{ds}** .

```
print('Grafica 3')
# --- Grafica 3: Id vs Vds (Cinco curvas) -----
plt.figure(3)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

# --- Cuevas características ids-vds ---
for [i,j] in zip(num_curvas_id_vds,range(0,len(num_curvas_id_vds))): 

    plt.plot(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i],muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i],label='Vgs = '+str(Vgs_prom[i])#,o',
              markevery=[ids_min[j],ids_max[j]], label='Vgs = '+str(Vgs_prom[i])) 

    # --- Puntos que delimitan las regiones de operacion ---
    plt.plot(linea_lim_x,linea_lim_y,linestyle = 'none',marker='o')
    for i in range(1,len(linea_lim_x)):
        plt.annotate(str(EngNumber(linea_lim_x[i]))+'V',
                     '+str(EngNumber(linea_lim_y[i]))+'A',(linea_lim_x[i],linea_lim_y[i]),textcoords="offset points",xytext=(5,-7),ha='left')

    # --- Recta de carga estatica ---
    plt.plot(puntos_grafica,recta_carga_estatica(puntos_grafica), label='Recta de carga estatica')

    # --- Punto de operacion,intersección entre curva que delimita regiones y recta de carga estatica, punto de maxima variación simétrica ---
    plt.plot(puntos_x[0],puntos_y[0],'o',label='Punto de operacion')
    #plt.plot([puntos_x[1],puntos_x[2]],[puntos_y[1],puntos_y[2]],'o')
    plt.plot(puntos_x[1:(len(puntos_x))],puntos_y[1:(len(puntos_x))],'o')

    for i in range(0,len(puntos_x)):
        v_sim_str = "
        alineacion = 'left'
        desplazamiento = (5,3)
        if i == 2:
            v_sim_str = 'Maxima variación simétrica: \n'
            desplazamiento = (5,0)
```

```

    elif i == 0 and puntos_x[0] < puntos_x[2]:
        alineacion = 'right'
        desplazamiento = (-5,-3)
    elif i == 0 and puntos_x[0] > puntos_x[2]:
        alineacion = 'left'
        desplazamiento = (5,-10)
    plt.annotate(v_sim_str+str(EngNumber(puntos_x[i]))+'V,
'+str(EngNumber(puntos_y[i]))+'A',(puntos_x[i],puntos_y[i]),textcoords="offset
points",xytext=desplazamiento,ha=alineacion)

# --- Curva que delimita regiones de operacion ---
puntos_grafica = np.linspace(0, puntos_x[1]*9/8, 100)
plt.plot(puntos_grafica,parabola_vsat(puntos_grafica),'--',label='Limite entre regiones')

x1,x2,y1,y2=plt.axis()
plt.axis([x1,5,y1,y2])

plt.xlabel('Vds')
plt.ylabel('Id')
plt.title("Id vs Vds")
plt.legend()
grafica = 'g/03_Id_vs_Vds' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

```

Figura 223. Fragmento de código para la **Gráfica 3: Id vs Vds**

En esta parte del método, además de graficar las cinco curvas I_d vs V_{ds} , agrega lo siguiente:

- Agrega los **puntos** en donde el transistor MOSFET **cambia de la región de operación** óhmica a la de saturación.
- Agrega la **curva que delimita las regiones de operación** óhmica y de saturación.
- Agrega la **recta de carga estática**.
- Agrega el **punto de máxima variación simétrica**.
- Agrega el **punto de operación del transistor**.

Para cada punto agregado, se calculan las coordenadas de su etiqueta, procurando que no se encimen unas de otras.

El resultado de esta parte es la siguiente gráfica:

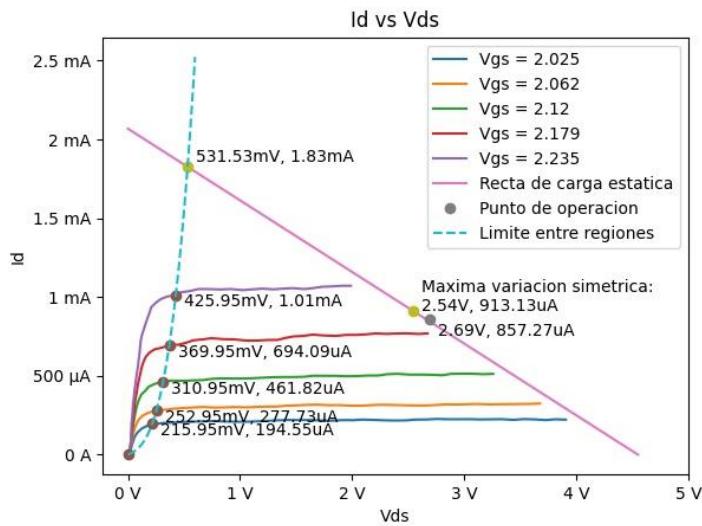


Figura 224. Gráfica 3: I_d vs V_{ds} .

Muestra las **5 curvas I_d vs V_{ds}** en azul, naranja, verde, rojo y gris. Para cada curva se especifica el voltaje V_{gs} utilizado. Adicionalmente, se observan los **puntos que delimitan las regiones de operación** y sobre estos puntos se traza la **curva que delimita las regiones de operación** discontinua en azul claro.

También se observa la **recta de carga estática** en rosa, sobre ella están el **punto de intersección** en verde claro **con la curva que delimita las regiones de operación**, el **punto de máxima variación simétrica** en verde claro y el **punto de operación** en gris.

6.4.2.3.4.6 Gráfica 4: gm vs Id (vi)

El código para generar esta **gráfica 4** es idéntico al mostrado en la sección 6.4.2.1.3.6. Se compara el parámetro de transconductancia g_m calculado con el valor correspondiente de **corriente de drenaje I_d** .

El resultado de esta parte es la siguiente gráfica:

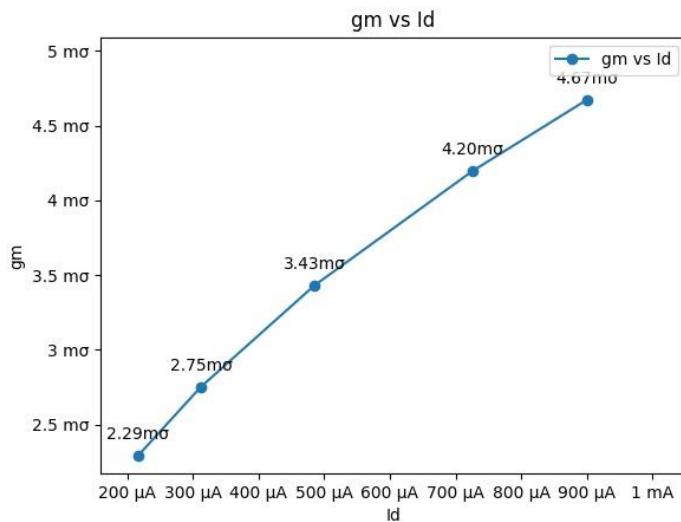


Figura 225. Gráfica 4: gm vs Id .

Se observa en azul la **curva g_m vs I_d** y se marcan las cinco muestras correspondientes a las cinco curvas I_d vs V_{ds} .

6.4.2.3.4.7 Gráfica 5: r_d vs I_d (vii)

El código para generar esta gráfica es idéntico al mostrado en la sección 6.4.2.1.3.7. Se compara la **resistencia r_d** del modelo híbrido pi previamente calculada con el valor correspondiente de **corriente de drenaje I_d** .

El resultado de esta parte es la siguiente gráfica:

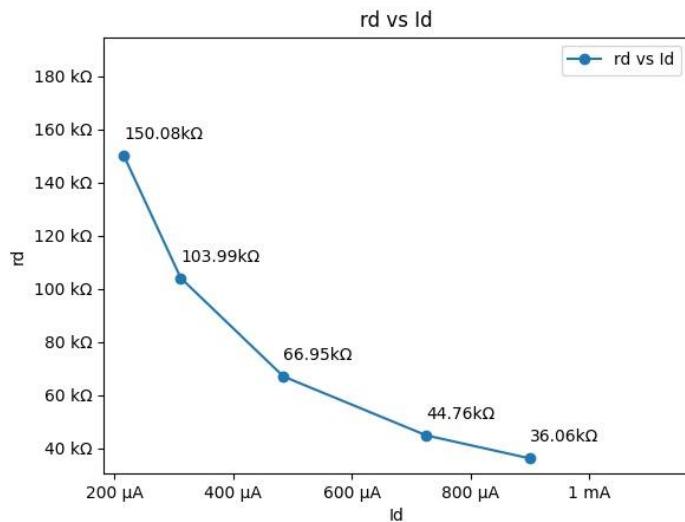


Figura 226. Gráfica 5: r_d vs I_d .

En azul se muestra la **curva r_d vs I_d** y se marcan las cinco muestras correspondientes a las cinco curvas I_d vs V_{ds} .

6.4.2.3.4.8 Gráfica 6: Voltajes del amplificador en fuente común (viii)

En esta parte se grafican el **voltaje de entrada** V_{gs} y el **voltaje de salida** V_{ds} cuando el transistor funciona como **amplificador en fuente común**.

```
print('Grafica 6')
# --- Grafica 6 Voltajes en el tiempo. Amplificador en fuente comun -----
-----
plt.figure(6)

for i in [0,1]:
    mark_v = [marker_v[2*i],marker_v[2*i+1]]

plt.plot(muestras[muestras_llaves[ind_voltajes[i*2]+6*sel_filtro]][num_curvas_pequena_se
nal[0]], label=muestras_llaves[ind_voltajes[2*i]+6*sel_filtro])

plt.plot(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_
senal[0]],'o-', markevery=mark_v, label=muestras_llaves[ind_voltajes[2*i+1]+6*sel_filtro])
    for j in [0,1]:
        voltaje =
        muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_senal[0]
][marker_v[2*i+j]] =
        xoffset,yoffset =
        calc.calcular_posiciones_label(ind_voltajes[1+i*2]+6*sel_filtro,marker_v[2*i+j])

plt.annotate(str(EngNumber(voltaje))+'V',(marker_v[2*i+j],voltaje),textcoords="offset
points",xytext=(xoffset,yoffset),ha='right')

x1,x2,y1,y2=plt.axis()
plt.axis([x1,x2,0,y2])

plt.xlabel('Muestras')
plt.ylabel('Voltaje')
plt.title("Voltajes en el tiempo. Amplificador en pequena senal")
plt.legend()
grafica = 'g/06_Voltajes_Amplificador_Fuente_Comun' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 227. Fragmento de código para la **Gráfica 6: Voltajes del amplificador en fuente común**.

Se muestran los valores de voltaje máximos y mínimos tanto del voltaje de entrada V_{gs} y de salida V_{ds} . También se muestra el voltaje de alimentación V_{gg} y V_{dd} .

El resultado de esta parte es la siguiente gráfica:

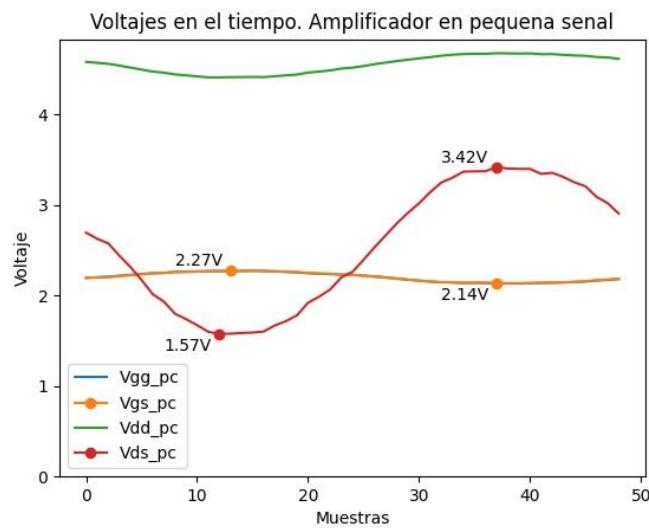


Figura 228. Gráfica 6: Voltajes del amplificador en fuente común.

En esta gráfica se observan los **voltajes** V_{gg} (azul), V_{gs} (naranja), V_{dd} (verde) y V_{ds} (rojo). En las curvas de V_{gs} y V_{ds} se marcan los puntos de máximo y mínimo voltaje.

6.4.2.3.4.9 Gráfica 7: Parámetros híbridos y punto de operación(ix)

En esta **gráfica 7** se muestra de forma condensada todos los **parámetros híbridos** calculados del transistor MOSFET. Estos parámetros incluyen:

- **Lambda.**
- El **voltaje de umbral de encendido** V_{th} del transistor MOSFET.
- Los parámetros I_{dss} , K_n y la **resistencia** r_d del modelo híbrido pi.

También se muestran los **parámetros** del **punto de operación** cuando el transistor MOSFET funciona como **amplificador**. Los parámetros asociados al punto de operación incluyen:

- El **voltaje entre drenaje y fuente del punto de operación** V_{dsq} .
- La **corriente de drenaje en el punto de operación** I_{dq} .
- Las **resistencias** r_g y r_d del modelo híbrido pi.
- La **resistencia equivalente** R_{lac} de salida utilizando la resistencia del circuito R_d en paralelo con la resistencia híbrida r_d .
- La **ganancia del amplificador** calculada de dos maneras, tanto **teórica** como **experimental**.

```
# --- Grafica 7 Resumen parametros hibridos-----
-----
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- RESUMEN PARAMETROS HIBRIDOS ---

cv2.putText(canvas2,"Transistor           MOSFET           2N7000", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"PARAMETROS           HIBRIDOS", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Lambda : " + str(EngNumber(MOS_avg_lambda)) + 'V^-1', (30,
130),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Voltaje de umbral de encendido : " + str(EngNumber(Vth)) + 'V',
(30, 150),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Idss   : " + str(EngNumber(Idss)) + 'A', (30,
170),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Kn    : " + str(EngNumber(Kn)) + 'A/V^2', (30,
190),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# --- PUNTO DE OPERACION (AMPLIFICADOR EN PEQUENA SENAL) ---
```

```

cv2.putText(canvas2,"PUNTO DE OPERACION", (30,
240),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Vdsq : " + str(EngNumber(v_op))+ 'V', (30,
270),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Idq : " + str(EngNumber(Idq))+ 'A', (30,
290),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"gm : " + str(EngNumber(gmq))+ 'S', (30,
310),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"rd : " + str(EngNumber(rdq))+ 'ohm', (30,
330),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"rg : " + str(rg), (30, 350),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Rlac (Rd||rd) : " + str(EngNumber(Rlac))+ 'ohm', (30,
375),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Ganancia experimental (Vout/Vin): " +
str(EngNumber(ganancia_experimental)), (30, 395),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Ganancia teorica (-gm*Rlac): " +
str(EngNumber(ganancia_teorica)), (30, 425),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0,
0), 1, cv2.LINE_AA)

filename = "g/07_parametros_hibridos" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

```

Figura 229. Fragmento de código para la **Gráfica 7: Parámetros híbridos y punto de operación.**

El resultado de esta parte es la siguiente gráfica:

```
Transistor MOSFET 2N7000
PARAMETROS HIBRIDOS
Lambda : 30.80mV^-1
Voltaje de umbral de encendido : 1.81V
Idss : 19.85mA
Kn : 6.07mA/V^2

PUNTO DE OPERACION
Vdsq : 2.69V
Idq : 857.27uA
gm : 4.56mS
rd : 37.88kohm
rg : inf
Riac (Rd||rd) : 2.08kohm
Ganancia experimental (Vout/Vin): -13.45
Ganancia teorica (-gm*Riac): -9.48
```

Figura 230. Gráfica 7: Parámetros híbridos y punto de operación.

Se muestra un resumen de los **parámetros híbridos** calculados del transistor y del **punto de operación** cuando funciona como **amplificador**.

6.4.2.4 Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el programa de Python del **circuito secuencial (d)**, se agregó la clase calculadora_datos_graficas.py y la clase fsm.py, quedando el diagrama de clases de la siguiente manera:

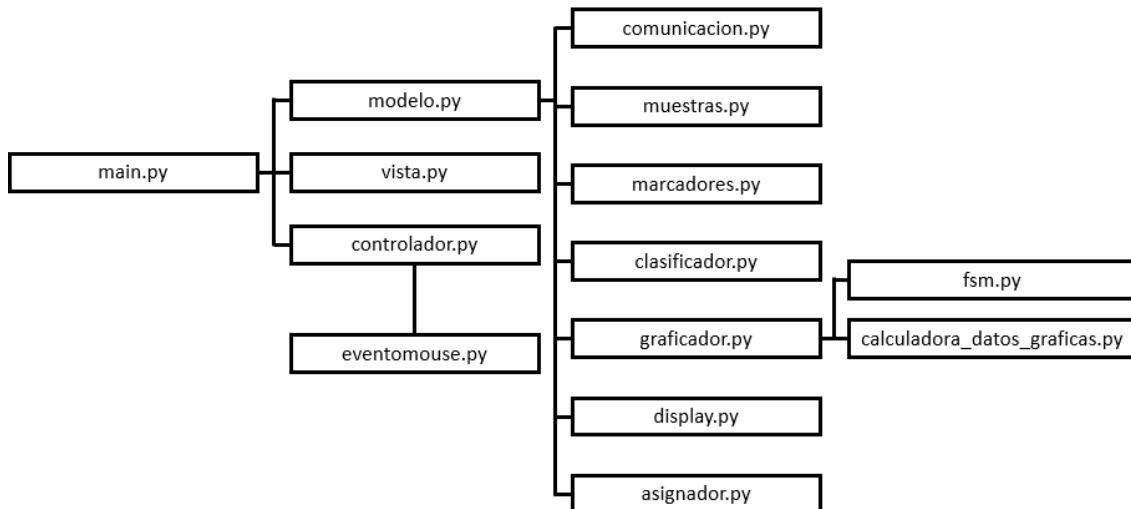


Figura 231. Diagrama de clases para el **circuito secuencial (d)**.

6.4.2.4.1 Modificaciones en la clase modelo.py

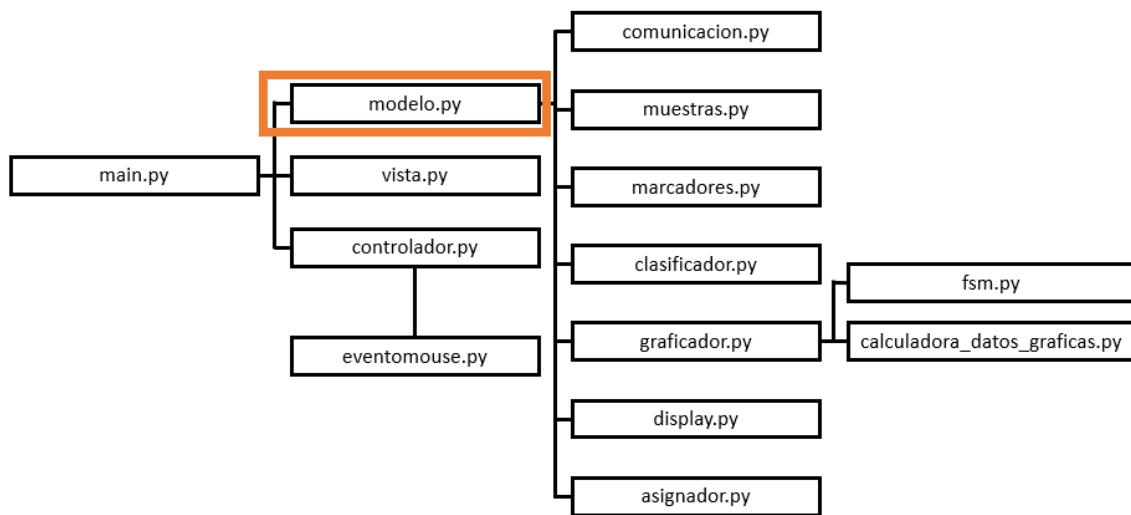


Figura 232. Ubicación de la clase modelo.py.

Esta clase contiene los siguientes métodos y atributos:



Figura 233. Métodos, atributos y propiedades del método modelo.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p[→] indica que es una **propiedad**.
- f indica que es un **atributo**.

Las modificaciones realizadas en la clase **modelo.py** se realizaron sobre el método “**inicializar_objetos_modelo**”.

En la clase modelo en el método “**inicializar_objetos_modelo**” se modifica lo siguiente:

```
titulos_vectores_muestras = ["entrada", "estado", "salida"]  
param_muestras = {'numMuestras': 20, 'numCanales': len(titulos_vectores_muestras),  
'llave': titulos_vectores_muestras}
```

Figura 234. Definición de los títulos de llaves para los vectores de muestras y número de muestras.

Se toman 20 muestras para los canales de muestras entrada, estado y salida. Es decir, se toman muestras para la respuesta y estado del circuito secuencial dada una entrada.

```
titulos_marcadores_display = {'m0': "Senales en el tiempo",  
'm1': "Tabla de estados",  
'm2': "Diagrama de estados",  
'm3': "Diagrama de estados",  
'm4': "Vector de prueba"}
```

Figura 235. Definición de los títulos de las gráficas a mostrar sobre la imagen del circuito.

Se presentarán 5 gráficas, descritas en la sección 6.4.1.4.

6.4.2.4.2 Descripción de las variables de los datos recibidos desde el microcontrolador

La variable **vectores_muestras** contiene los 3 canales de muestras entrada, estado y salida. Es un diccionario de tres dimensiones. La variable **lim_max** contiene el número de muestras por canal. En este caso hay 20 muestras por canal.

Para acceder a cada uno de los datos se tiene que especificar:

- **Llave del canal de muestras:** Se indica entre comillas el nombre del canal de muestras a utilizar.
- **Número de la muestra o conjunto de muestras:** Se accede a una muestra específica, colocando un número de entre 0 y 50. También es posible acceder a un conjunto de muestras, colocando la muestra inicial, seguido de dos puntos e indicando la muestra final.

Por ejemplo, para acceder a las muestras del canal entrada y muestras 4:25, se indica de la siguiente manera:

```
vectores_muestras['Voltaje'][4:25]
```

Para el programa del **circuito secuencial (d)**, se toman los 2 canales de muestras Voltaje y Binario. Por cada canal se toman 2 series de datos de 50 muestras cada uno.

A cada canal de muestras se le aplican dos filtros diferentes, un filtro circular y un filtro pasa bajas, guardando cada canal con filtro con un nombre diferente, mostrado en la tabla 34:

Canal de muestras	Nombre del canal de muestras		
	Sin filtro	Con filtro circular	Con filtro pasa bajas
entrada	entrada	entrada_pc	entrada_lp
estado	estado	estado_pc	estado_lp
salida	salida	salida_pc	salida_lp

Tabla 34. Descripción de los canales con filtro circular y pasa bajas.

La variable **muestras_llaves** contiene los títulos de las llaves de la variable **vector_muestras** mostrados en la tabla 35. El orden en que se guardan es el siguiente:

Orden	Canal de datos
0	Voltaje
1	Binario
2	Voltaje_pc
3	Binario_pc
4	Voltaje_lp
5	Binario_lp

Tabla 35. Orden de los títulos en la variable **muestras_llaves**.

Cada uno de los canales expuestos de la tabla 28 contienen 20 muestras cada uno. Se observa la variable **muestras_llaves** guarda los títulos de las llaves en el siguiente orden:

0. Títulos de las llaves de los canales **sin filtro**.
1. Títulos de las llaves de los canales **con filtro circular**.
2. Títulos de las llaves de los canales **con filtro pasa bajas**.

6.4.2.4.3. Clase fsm.py

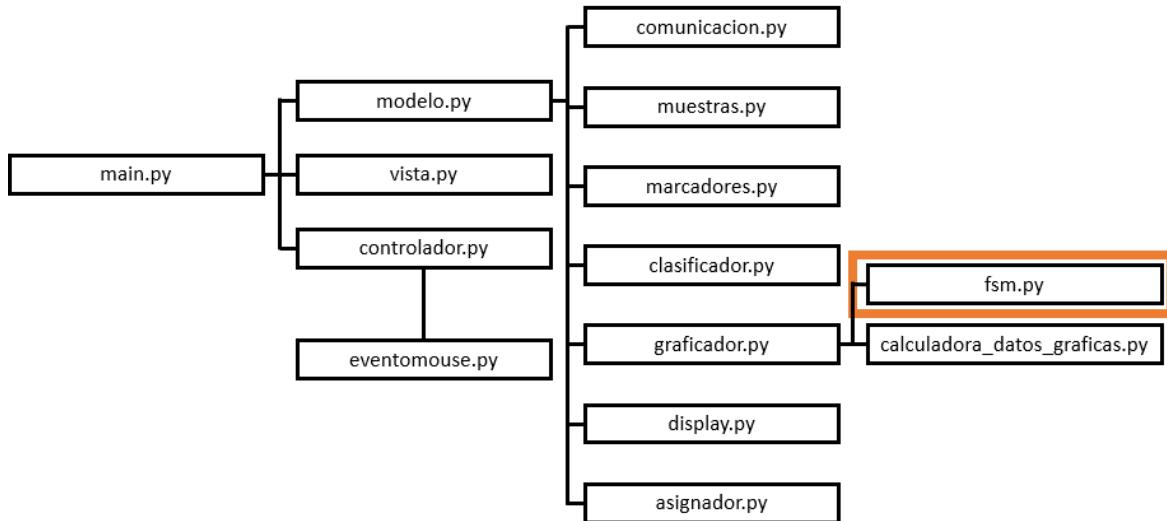


Figura 236. Ubicación de la clase fsm.py.

Esta clase contiene todos los datos que especifican el comportamiento del circuito secuencial o máquina de estados finitos.

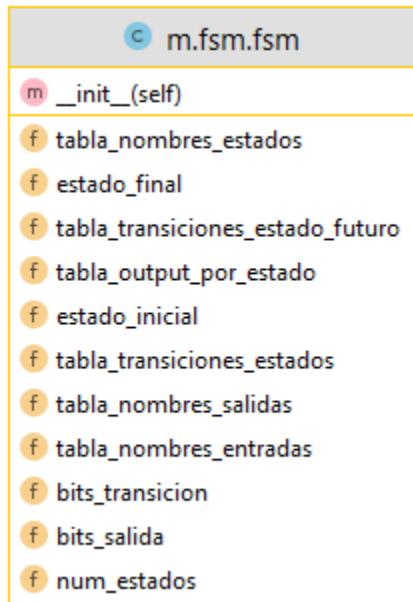


Figura 237. Métodos, atributos y propiedades de la clase fsm.py.

Donde:

-  indica que es una **clase**.
-  indica que es un **método**.
-  indica que es una **propiedad**.
-  indica que es un **atributo**.

Se presentará el método “`__init__`”.

6.4.2.4.3.1 Constructor

```
class fsm():
    def __init__(self):
        print("")
        print(" CONSTRUCTOR: Clase: fsm")

        # Copiar del programa de ARDUINO IMPLEMENTACION FSM todos los datos de la
        # maquina de estados y adaptarlo a la sintaxis de python

        #
        # num_estados -> indica cuantos estados tiene la maquina
        # bits_transicion -> Indica cuantos bits tiene las transiciones que permiten cambiar
        # entre estados
        # bits_salida -> Indica la cantidad de bits de salida de la maquina de estados
        #

        self.num_estados = 4
        self.bits_transicion = 3
        self.bits_salida = 3
        self.estado_inicial = 0
        self.estado_final = 1

        #
        # Las siguientes tablas especifican el funcionamiento de la maquina de estados finitos
        # El tamano de estas tablas esta determinador por las constantes num_estados,
        # bits_salida y bits_transicion
        # Cada estado de la maquina corresponde a una entrada de la tabla, como se indica
        # en cada una de las tablas.
        #
        # tabla_transiciones_estados -> Indica cuales son las transiciones con las que la
        # maquina cambia de estado
        #           Las transiciones que no aparecen, se asume que la maquina no
        # cambia de estado
        #           IMPORTANTE: Se completan la tabla con -1 para las transiciones
        # en las que no hay cambio de estado
        #
        # tabla_transiciones_estado_futuro -> Indica el estado al cual va a cambiar la maquina
        # en el siguiente pulso
        #           de reloj. Cada estado especificado en esta tabla corresponde
        # a la
        #           transicion de la tabla tabla_transiciones_estados
        #
        # tabla_output_por_estado -> Indica la salida del estado actual de la maquina
        #

        self.tabla_transiciones_estados = [
            [0B100,0B101],                                     # estado 0: Apagado
```

```

Encendido funcionando [0B000,0B001,0B010,0B011,0B101,0B111], # estado 1:
Corto circuito      [0B000,0B001,0B010,0B011,0B110,0B111], # estado 2:
Reset              [0B000,0B001,0B010,0B011,0B100,0B101] # estado 3:
                    ]

self.tabla_transiciones_estado_futuro = [
    [1,1],           # estado 0
    [0,0,0,0,2,2],   # estado 1
    [0,0,0,0,3,3],   # estado 2
    [0,0,0,0,1,2]    # estado 3
]

self.tabla_output_por_estado = [
    0B000,    # estado 0
    0B100,    # estado 1
    0B001,    # estado 2
    0B011    # estado 3
]

self.tabla_nombres_estados = [
    'q0 Apagado',
    'q1 Encendido funcionando',
    'q2 Corto circuito',
    'q3 Reset'
]

self.tabla_nombres_salidas = [
    'bit 2: Relevador ON/OFF',
    'bit 1: LED de Reset',
    'bit 0: LED de Corto circuito'
]

self.tabla_nombres_entradas = [
    'bit 2: Boton de Alimentacion',
    'bit 1: Boton de Reset',
    'bit 0: Sensor de Corto Circuito'
]

```

Figura 238. Método “`__init__`”

El constructor inicializa todas las tablas y variables de la máquina de estados finitos. Esto incluye:

- **num_estados:** Es el **número de estados** de la máquina de estados finitos.
- **bits_transicion:** Es el **número de bits utilizados para cambiar entre estados**.
- **bits_salida:** Es el **número de bits de salida** de la máquina de estados finitos.
- **estado_inicial:** Es el **número del estado inicial** de la máquina de estados finitos.
- **estado_final:** Es el **número del estado final** en el que se considera que la máquina de estados ya terminó.
- **tabla_transiciones_estados:** Indica para cada estado de la máquina de estados finitos la **entrada con la que cambia** la máquina de estados finitos **hacia otro estado diferente** del actual.
- **tabla_transiciones_estado_futuro:** Indica, según la tabla "tabla_transiciones_estados", el **estado al cuál cambia** la máquina de estados finitos.
- **tabla_output_por_estado:** Indica la **salida de cada estado**.
- **tabla_nombres_estados:** Indica el **nombre de cada estado**.
- **tabla_nombres_salidas:** Indica el **nombre de cada bit de la salida**.
- **tabla_nombres_entradas:** Indica el **nombre de cada bit de la entrada**.

Esta clase se modifica cada que cambia la máquina de estados a utilizar.

6.4.2.4.4. Clase calculadora_datos_graficas.py

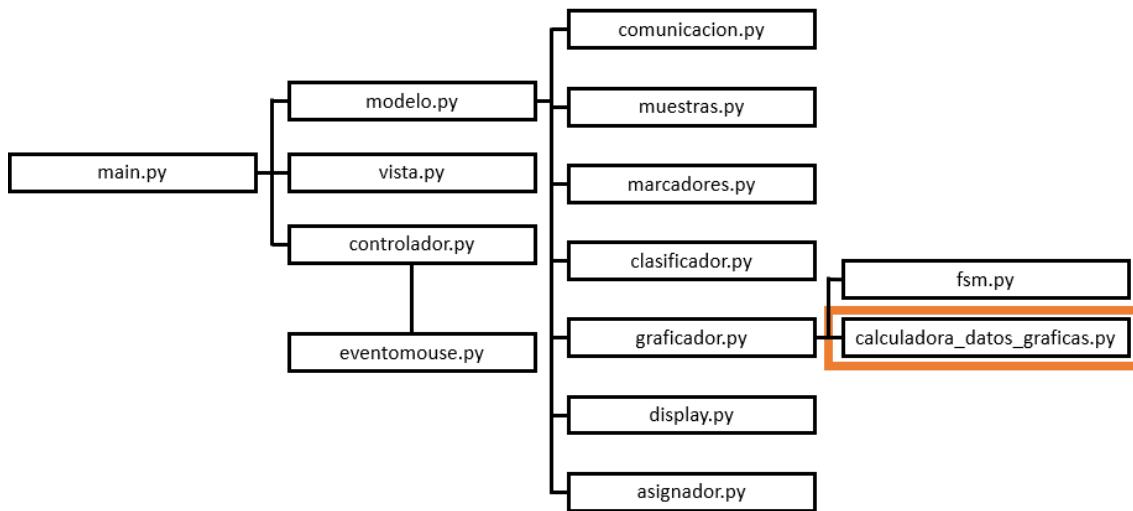


Figura 239. Ubicación de la clase calculadora_datos_gráficas.py.

Esta clase contiene los siguientes métodos y atributos:

```

c m.calculadora_datos_gráficas.calculadora_datos_gráficas
m __init__(self)
m ec_recta_dos_puntos(self,p1,p2)
m ec_recta_punto_pendiente_perpendicular(self,p1,m)
m puntos_equidistantes_sobre_recta_desde_punto(self,p1,m,d)
m list_duplicates_of(self,seq,item)
m decimal_a_binario(self,decimal,num_bits)
m obtener_bits_cambiantes(self,fsm1,transiciones,estado)
m obtener_str_de_binario_transicion(self,bits,binario,diff)
m circle_line_segment_intersection(self,circle_center,circle_radius,pt1,pt2,full_line=True,tangent_tol=1e-9)
m obtener_puntos_lineas_transicion(self,canvas2,xcoords,ycoords,c_radio,length_lineas,c_actual,c_transicion)
m obtener_coordenadas_labels(self,p1,p2)
m get_intersections(self, x0, y0, r0, x1, y1, r1)

```

Figura 240. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p⁺ indica que es una **propiedad**.
- f indica que es un **atributo**.

Se presentarán los siguientes métodos:

1. **Métodos para convertir números decimales a binario sin y con condiciones X de no importa (1).** Se presentan los siguientes métodos:
 - a. Método “decimal a binario” (1a).
 - b. Método “list_duplicates_of” (1b).
 - c. Método “obtener_bits_cambiantes” (1c).
 - d. Método “obtener_str_de_binario_transicion” (1d).
2. **Métodos para obtener las coordenadas de diferentes elementos del diagrama de estados (sin utilizar el paquete Graphviz) (2).** Se presentan los siguientes métodos:
 - a. Método “obtener_puntos_lineas_transicion” (2a).
 - b. Método “circle_line_segment_intersection” (2b).
 - c. Método “ec_recta_dos_puntos” (2c).
 - d. Método “ec_recta_punto_pendiente_perpendicular” (2d).
 - e. Método “puntos_equidistantes_sobre_recta_desde_punto” (2e).
3. **Métodos auxiliares para calcular coordenadas de diferentes elementos en diagramas.** Se presentan los siguientes métodos:
 - a. Método “get_intersections”.
 - b. Método “obtener_coordenadas_labels”.

6.4.2.4.4.1 Métodos para convertir números decimales a binario sin y con condiciones X de no importa (1)

6.4.2.4.4.1.1 Método “decimal a binario” (1a)

```
def decimal_a_binario(self,decimal,num_bits):
    #str_bin = '0b'
    str_bin = ""
    for i in range(num_bits-1,-1,-1):
        bstr = decimal & int(pow(2,i))
        bstr >= i
        str_bin +=str(bstr)
    return str_bin
```

Figura 241. Método “decimal a binario”.

Este método **convierte un número decimal a binario**. No considera si hay condiciones X de no importa.

Para poder realizar esta consideración, se utilizan los siguientes métodos de las secciones 6.4.2.4.3.1.2 a 6.4.2.4.3.1.4.

6.4.2.4.4.1.2 Método “list_duplicates_of” (1b)

```
# -----
# Obtiene todos los indices de un elemento repetido en una lista.
# Tomado de https://stackoverflow.com/questions/5419204/index-of-duplicates-items-in-a-python-list
#
def list_duplicates_of(self,seq,item):
    start_at = -1
    locs = []
    while True:
        try:
            loc = seq.index(item,start_at+1)
        except ValueError:
            break
        else:
            locs.append(loc)
            start_at = loc
    return locs
```

Figura 242. Método “list_duplicates_of”.

Este método **obtiene los elementos repetidos de una lista**. Se utiliza en el programa para obtener todas entradas que, aplicadas a la máquina de estados finitos, cambian a un mismo estado.

Por ejemplo, si la máquina de estados finitos está en el estado 0, y se consideran las entradas 100 y 101, la máquina de estados finitos cambia al estado 1. Este método identifica que la entrada 100 y 101 cambian ambos al estado 1.

6.4.2.4.4.1.3 Método “obtener_bits_cambiantes” (1c)

```
def obtener_bits_cambiantes(self,fsm1,transiciones,estado):
    diff = 0
    if len(transiciones) > 1:
        for k in range(1,len(transiciones)):
            diff_orx      = fsm1.tabla_transiciones_estados[estado][transiciones[0]]      ^
fsm1.tabla_transiciones_estados[estado][transiciones[k]]
            diff |= diff_orx
    return diff
```

Figura 243. Método “obtener_bits_cambiantes”.

Este método observa si hay bits diferentes en todas las entradas que cambian de estado a la máquina de estados finitos (obtenidas en la función anterior). Si dos bits son diferentes, se coloca un 1, si no se coloca un 0.

Por ejemplo, una vez identificadas las entradas 100 y 101, se les aplica una or exclusiva bit a bit. Esto es realizar la siguiente operación:

$$100 \oplus 101 = 001$$

Esto significa que el bit menos significativo b_0 (el bit de la derecha) es el que cambia (1 lógico) en los dos números, mientras que los bits más significativos b_1 y b_2 no cambian (0 lógico).

6.4.2.4.4.1.4 Método “obtener_str_de_binario_transicion” (1d)

```
# -----
# Obtiene una representacion en binario de un numero en decimal, que ademas incluye
condiciones de no importa (representado por una "X")
#
# -----
def obtener_str_de_binario_transicion(self,bits,binario,diff):
    str_transicion =""
    for k in range(bits,-1,-1):
        if diff >= math.pow(2,k):
            diff -= math.pow(2,k)
            str_transicion += 'X'
        else:
            binstr = binario & int(math.pow(2,k))
            binstr >>= k
            str_transicion += str(binstr)
    return str_transicion
```

Figura 244. Método “obtener_str_de_binario_transicion”.

Con base en los resultados de los dos métodos obtenidos anteriormente, se coloca una condición X de no importa en la posición de los bits cuyo valor es un 1 lógico (esto es si hay bits con diferente valor en las entradas). Si no, se coloca el valor del bit que es común para todas las entradas.

Por ejemplo, utilizando el resultado $100 \oplus 101 = 001$, se coloca una condición X en el bit menos significativo b_0 , quedando $10X$ para todas entradas que hacen que la máquina de estados finitos cambie del estado 0 al estado 1.

6.4.2.4.4.2 Métodos para obtener las coordenadas de diferentes elementos del diagrama de estados (sin utilizar el paquete graphviz) (2)

6.4.2.4.4.2.1 Método “obtener_puntos_lineas_transicion” (2a)

```
# -----
# Obtiene las coordenadas utilizadas para dibujar lineas entre estados para el diagrama de estados
# Se siguió la siguiente secuencia de pasos:
# 1. Encontrar las intersecciones entre la circunferencia del estado actual y la linea que une los centros de la circunferencia actual y la del centro que es de transicion.
# 2. Calcular la recta perpendicular a la linea generada en el paso anterior sobre el punto de intersección.
# 3. Calcular la intersección de la recta perpendicular anterior con la circunferencia del estado actual. Se deben obtener dos soluciones.
# Hay dos casos especiales que son que la pendiente de la recta del paso 1 sea horizontal o vertical:
# a. Cuando la pendiente m tiende a infinito, se calcula las coordenadas de los puntos que equidistan de la intersección del paso 1 y se ubican sobre la recta perpendicular
# b. Si la pendiente m no tiende a infinito o m es igual a cero, se utiliza un método más sencillo. Se toma la intersección del paso 1 como centro de una nueva circunferencia
# y se buscan las intersecciones con la circunferencia del estado actual.
# 4. Calcular las intersecciones con la circunferencia de transición
# Hay dos casos especiales que igual son dependiendo de la pendiente de la recta del paso 1 si es horizontal o vertical:
# a. Cuando la pendiente m tiende a infinito, se realiza el mismo procedimiento descrito en los pasos 1, 2 y 3.a, esta vez con la circunferencia de transición.
# b. Si la pendiente m no tiende a infinito, se calcula la ecuación que une las circunferencias del estado actual y la de transición en los puntos calculados en el paso 3
# para encontrar las intersecciones con la circunferencia de transición.
# -----
def
obtener_puntos_lineas_transicion(self,canvas2,xcoords,ycoords,c_radio,length_lineas,c_actual,c_transicion):
    lineas_transiciones = []

    # Se obtienen las intersecciones de la recta que une los centros de las circunferencias del estado actual y de transición y la circunferencia del estado actual
    inters_origen =
    self.circle_line_segment_intersection((xcoords[c_actual],ycoords[c_actual]),c_radio[1],(xcoords[c_actual],ycoords[c_actual]),(xcoords[c_transicion],ycoords[c_transicion]),False,1e-9)
    #print("inters_origen: "+str(inters_origen), "len(inters_origen): "+str(len(inters_origen)))

    # Calcula la ecuación de la recta que une los centros de las circunferencias del estado actual y de transición
    m,b =
    self.ec_recta_dos_puntos(inters_origen[0],[xcoords[c_transicion],ycoords[c_transicion]])
```

```

#recta = np.poly1d([m,b])

# Calcula la ecuacion de la recta perpendicular a la anterior y sobre el punto de interseccion
mp, bp = self.ec_recta_punto_pendiente_perpendicular(inters_origen[0], m)
recta_p = np.poly1d([mp, bp])
#print('recta perpendicular calculada')
#print("m: "+str(m)+" , mp: "+str(mp))

if mp > 9223372036854775807:# la recta recta_p es perpendicular
    #print('calcular puntos con recta perpendicular')
    xp1, yp1, xp2, yp2 =
    self.puntos_equidistantes_sobre_recta_desde_punto((inters_origen[0][0], inters_origen[0][1]), mp, length_lineas)
    lineas_transiciones.append([xp1, yp1])
    lineas_transiciones.append([xp2, yp2])
    #print(lineas_transiciones)

else:# la recta recta_p no es perpendicular
    if m > 9223372036854775807:# la recta recta_p es horizontal (la recta que une los centros de las circunferencias del estado actual y de transicion es vertical)
        #print('calcular puntos con circunferencia especial')
        # Se realiza un ajuste para obtener las intersecciones con la circunferencia del estado actual utilizando el punto de interseccion obtenido anteriormente como centro de una circunferencia
        lineas_transiciones =
        self.circle_line_segment_intersection(inters_origen[0], length_lineas, (xcoords[c_actual]-50, int(recta_p(xcoords[c_actual]))), (xcoords[c_transicion]+50, int(recta_p(xcoords[c_transicion]))), True, 1e-9)

    else:
        #print('calcular puntos con circunferencia')
        # Se obtienen las intersecciones con la circunferencia del estado actual utilizando el punto de interseccion obtenido anteriormente como centro de una circunferencia
        lineas_transiciones =
        self.circle_line_segment_intersection(inters_origen[0], length_lineas, (xcoords[c_actual], int(recta_p(xcoords[c_actual]))), (xcoords[c_transicion], int(recta_p(xcoords[c_transicion]))), True, 1e-9)

#print('primeras dos transiciones calculadas')
#print("lineas_transiciones: "+str(lineas_transiciones))

# Se calculan las intersecciones con la circunferencia de transicion
if m < 9223372036854775807:# la recta que une los centros de las circunferencias del estado actual y de transicion no es vertical
    for i in [0,1]:
        # Se calcula una recta que une alguna de las intersecciones anteriores con la circunferencia de transicion
        recta_transicion = np.poly1d([m, -m*lineas_transiciones[i][0]+lineas_transiciones[i][1]])

```

```

        #print(str(lineas_transiciones[i][0])+
"+str(recta_transicion(lineas_transiciones[i][0])))
        #print(str(xcoords[c_transicion])+
"+str(recta_transicion(xcoords[c_transicion])))

        # Puntos de inicio y final de esta recta
ptr0 = (lineas_transiciones[i][0],recta_transicion(lineas_transiciones[i][0]))
ptr1 = (xcoords[c_transicion],recta_transicion(xcoords[c_transicion]))

#print("ptr0: "+str(ptr0))
#print("ptr1: "+str(ptr1))

        # Se obtienen las coordenadas de las intersecciones con la circunferencia de
transicion con la recta recta_transicion
        p_transicion =
self.circle_line_segment_intersection((xcoords[c_transicion],ycoords[c_transicion]),c_radio
[1],ptr0,ptr1,False,1e-9)

        p_transicion = np.reshape(p_transicion,-1)
p_transicion = p_transicion.tolist()
lineas_transiciones.append(p_transicion)

else:# la recta que une los centros de las circunferencias del estado actual y de
transicion es vertical
        # Obtiene las coordenadas de la interseccion entre la recta que une los centros de
las circunferencias de estado actual y de transicion y la circunferencia de transicion
        inters_fin =
self.circle_line_segment_intersection((xcoords[c_transicion],ycoords[c_transicion]),c_radio
[1],(xcoords[c_transicion],ycoords[c_transicion]),(xcoords[c_actual],ycoords[c_actual]),Fals
e,1e-9)
#print("inters_fin: "+str(inters_fin), "len(inters_fin): "+str(len(inters_fin)))

#cv2.circle(canvas2, tuple(inters_fin[0]), 5, (0,255,0), 3)

        # Calcula la recta perpendicular a la anterior en el punto de la interseccion
mp,bp = self.ec_recta_punto_pendiente_perpendicular(inters_fin[0],m)
#recta_p_aux = np.poly1d([mp,bp])
#print(": "+str(recta_p_aux))

        # Se obtienen los puntos equidistantes al punto de interseccion sobre la recta
perpendicular.
        xp1,yp1,xp2,yp2 =
self.puntos_equidistantes_sobre_recta_desde_punto((inters_fin[0][0],inters_fin[0][1]),mp,le
ngth_lineas)

        lineas_transiciones.append([xp1,yp1])
lineas_transiciones.append([xp2,yp2])
#print(lineas_transiciones)

#print("lineas_transiciones: "+str(lineas_transiciones))
#for i in [0,1,2,3]:

```

```

#cv2.circle(canvas2, tuple(lineas_transiciones[i]), 5, (0,0,0), 3)

return lineas_transiciones

```

Figura 245. Método “obtener_puntos_lineas_transicion”.

Considerando un diagrama de estados, en donde cada circulo representa un estado y cada flecha una transición entre estados:

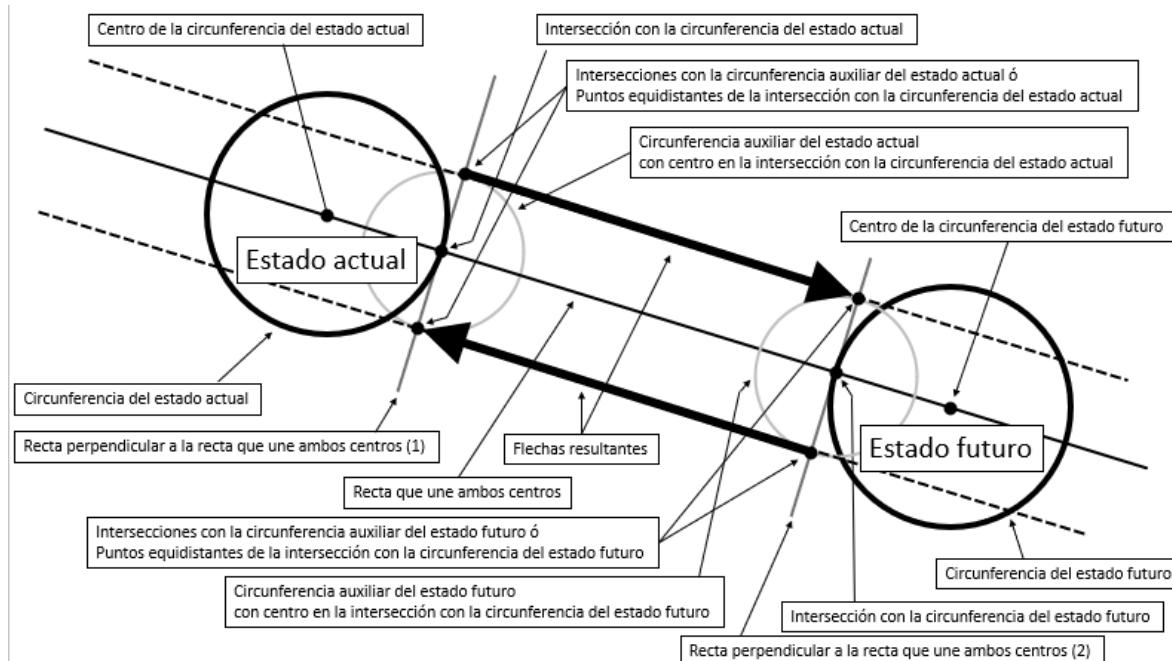


Figura 246. Ejemplo de para el método “obtener_puntos_lineas_transición”.

Este método realiza lo siguiente:

1. Considerando dos circunferencias, la **circunferencia del estado actual** y la **circunferencia del estado futuro**, se encuentra la **recta que une ambos centros**. Calcula los **puntos de intersección** de esta recta que une ambos centros **con la circunferencia del estado actual** y la **circunferencia del estado futuro**.
2. Calcula la **recta perpendicular (1)** a la **recta que une ambos centros** en el **punto de intersección** con la **circunferencia del estado actual**.
3. Calcula las **intersecciones con la circunferencia auxiliar actual** de la **recta perpendicular a la recta que une los centros**. Deben de resultar dos puntos:
 - a. Si la pendiente de la **recta perpendicular a la recta que une los centros** tiende a infinito (es vertical), se calculan las coordenadas de los puntos que equidistan de la intersección del paso 1 y se ubican sobre la recta perpendicular.
 - b. Si la pendiente de la **recta perpendicular a la recta que une los centros** no tiende a infinito (no es vertical), se utiliza un método más sencillo. Se toma la intersección del paso 1 como centro de una nueva **circunferencia auxiliar**

- del estado actual** y se buscan las intersecciones de la recta perpendicular a la recta que une los centros con esta circunferencia auxiliar del estado actual.
4. Se calcula otra **recta perpendicular a la recta que une los centros (2)** sobre la **intersección con la circunferencia del estado futuro**. Calcula **las intersecciones con la circunferencia auxiliar del estado futuro**:
 - a. Si la pendiente de la **recta que une los centros** tiende a infinito (es vertical), se realiza el mismo procedimiento descrito en el paso 3a, esta vez con la **circunferencia del estado futuro**.
 - b. Si la pendiente de la **recta que une los centros** no tiende a infinito (no es vertical), se calcula la ecuación de la recta entre los **puntos de intersección de la circunferencia auxiliar del estado actual y los puntos de intersección con la circunferencia auxiliar del estado futuro**.

Este método devuelve las coordenadas de los puntos de inicio y final de las dos **flechas resultantes**.

Este método se apoya en los siguientes métodos de las secciones 6.4.2.4.3.2.2 a 6.4.2.4.3.2.4.

6.4.2.4.4.2.2 Método “circle_line_segment_intersection” (2b)

```
# -----
# Obtiene las coordenadas de las intersecciones entre un circulo y una recta.
# Tomado de https://stackoverflow.com/questions/30844482/what-is-most-efficient-way-
to-find-the-intersection-of-a-line-and-a-circle-in-py
#
def circle_line_segment_intersection(self, circle_center, circle_radius, pt1, pt2,
full_line=True, tangent_tol=1e-9):
    """ Find the points at which a circle intersects a line-segment. This can happen at 0, 1,
or 2 points.

:param circle_center: The (x, y) location of the circle center
:param circle_radius: The radius of the circle
:param pt1: The (x, y) location of the first point of the segment
:param pt2: The (x, y) location of the second point of the segment
:param full_line: True to find intersections along full line - not just in the segment. False
will just return intersections within the segment.
:param tangent_tol: Numerical tolerance at which we decide the intersections are close
enough to consider it a tangent
:return Sequence[Tuple[float, float]]: A list of length 0, 1, or 2, where each element is a
point at which the circle intercepts a line segment.
```

Note: We follow: <http://mathworld.wolfram.com/Circle-LineIntersection.html>

(p1x, p1y), (p2x, p2y), (cx, cy) = pt1, pt2, circle_center

```

(x1, y1), (x2, y2) = (p1x - cx, p1y - cy), (p2x - cx, p2y - cy)
dx, dy = (x2 - x1), (y2 - y1)
dr = (dx ** 2 + dy ** 2)**.5
big_d = x1 * y2 - x2 * y1
discriminant = circle_radius ** 2 * dr ** 2 - big_d ** 2

if discriminant < 0: # No intersection between circle and line
    return []
else: # There may be 0, 1, or 2 intersections with the segment
    intersections = [
        [int(cx + (big_d * dy + sign * (-1 if dy < 0 else 1) * dx * discriminant**.5) / dr ** 2),
         int(cy + (-big_d * dx + sign * abs(dy) * discriminant**.5) / dr ** 2)],
        for sign in ((1, -1) if dy < 0 else (-1, 1))] # This makes sure the order along the
    segment is correct
    if not full_line: # If only considering the segment, filter out intersections that do not
        fall within the segment
        fraction_along_segment = [(xi - p1x) / dx if abs(dx) > abs(dy) else (yi - p1y) / dy
        for xi, yi in intersections]
        intersections = [pt for pt, frac in zip(intersections, fraction_along_segment) if 0 <=
        frac <= 1]
        if len(intersections) == 2 and abs(discriminant) <= tangent_tol: # If line is tangent to
            circle, return just one point (as both intersections have same location)
            return [intersections[0]]
        else:
            return intersections

```

Figura 247. Método “circle_line_segment_intersection”.

El método “circle_line_segment_intersection” obtiene las intersecciones entre una circunferencia y una línea recta.

6.4.2.4.4.2.3 Método “ec_recta_dos_puntos” (2c)

```

# -----
# Calcula la ecuacion de una recta (pendiente m y ordenada al origen b) dados dos
# puntos
#
# -----
def ec_recta_dos_puntos(self,p1,p2):
    xp1,yp1 = p1
    xp2,yp2 = p2
    m = float('inf')
    b = float('inf')
    if xp2 - xp1 != 0:
        m = (yp2 - yp1) / (xp2 - xp1)
        b = (-xp1*yp2 - yp1*xp2) / (xp2 - xp1)

    #print("m: "+str(m)+" , b: "+str(b))

```

```
    return m,b
```

Figura 248. Método “ec_recta_dos_puntos”.

El método “ec_recta_dos_puntos” calcula la pendiente m y la ordenada al origen b dados dos puntos.

6.4.2.4.4.2.4 Método “ec_recta_punto_pendiente_perpendicular” (2d)

```
# -----
# Calcula la ecuacion de una recta perpendicular (pendiente m y ordenada al origen b)
# dado un punto y una pendiente
#
# -----
def ec_recta_punto_pendiente_perpendicular(self,p1,m):
    xp1,yp1 = p1

    if m == 0:
        mp = float('inf')
    else:
        mp = -1 / m

    bp = (-mp*xp1 + yp1)
    #print("m: "+str(mp)+", b: "+str(bp))
    return mp,bp
```

Figura 249. Método “ec_recta_punto_pendiente_perpendicular”.

El método “ec_recta_punto_pendiente_perpendicular” calcula la pendiente mp y la ordenada al origen bp de una recta perpendicular a otra, dado un punto y una pendiente.

6.4.2.4.2.5 Método “puntos_equidistantes_sobre_recta_desde_punto” (2e)

```
# -----
# Calcula las coordenadas de dos puntos equidistantes dado un punto, la pendiente de
una recta y la distancia entre ellos.
#
# -----
def puntos_equidistantes_sobre_recta_desde_punto(self,p1,m,d):
    xp1,yp1 = p1
    x3a = xp1
    x3b = xp1
    y3a = yp1 + d
    y3b = yp1 - d

    if m < 9223372036854775807 and m != 0: # la recta no es vertical ni horizontal
        ec_puntos = np.poly1d([1+m**2,-2*xp1*(1+m**2),(1+m**2)*xp1-d**2])
        print(ec_puntos)
        x3a,x3b = ec_puntos.r
        print(ec_puntos.r)
        y3a = m * x3a - m * xp1 + yp1
        y3b = m * x3b - m * xp1 + yp1
    if m == 0: # la recta es horizontal
        #print('aqui')
        x3a = xp1 - d
        x3b = xp1 + d
        y3a = yp1
        y3b = yp1
    """print(x3a)
    print(y3a)
    print(x3b)
    print(y3b)"""
    return int(x3a),int(y3a),int(x3b),int(y3b)
```

Figura 250. Método “puntos_equidistantes_sobre_recta_desde_punto”.

El método “puntos_equidistantes_sobre_recta_desde_punto” calcula dos puntos equidistantes, dados una recta, un punto sobre esta recta y la distancia que debe de haber entre ellos.

6.4.2.4.4.3 Métodos auxiliares para calcular coordenadas de diferentes elementos en diagramas (3)

6.4.2.4.4.3.1 Método “get_intersections” (3a)

```
# -----
# Obtiene las intersecciones dadas dos circunferencias, dadas las coordenadas de su
# centro y radio.
# Tomado de https://stackoverflow.com/questions/55816902/finding-the-intersection-of-
# two-circles
# -----
def get_intersections(self, x0, y0, r0, x1, y1, r1):
    # circle 1: (x0, y0), radius r0
    # circle 2: (x1, y1), radius r1

    d = math.sqrt((x1-x0)**2 + (y1-y0)**2)

    # non intersecting
    if d > r0 + r1 :
        return None
    # One circle within other
    if d < abs(r0-r1):
        return None
    # coincident circles
    if d == 0 and r0 == r1:
        return None
    else:
        a=(r0**2-r1**2+d**2)/(2*d)
        h=math.sqrt(r0**2-a**2)
        x2=x0+a*(x1-x0)/d
        y2=y0+a*(y1-y0)/d
        x3=x2+h*(y1-y0)/d
        y3=y2-h*(x1-x0)/d

        x4=x2-h*(y1-y0)/d
        y4=y2+h*(x1-x0)/d

    return (x3, y3, x4, y4)
```

Figura 251. Método “get_intersections”.

Dados los radios y coordenadas de los centros de dos circunferencias, el método “get_intersections” obtiene las intersecciones entre ellas.

6.4.2.4.4.3.3 Método “obtener_coordenadas_labels” (3b)

```
# -----
# Obtiene las coordenadas de las etiquetas para las líneas de transiciones del diagrama
de estados.
#
# -----
def obtener_coordenadas_labels(self,p1,p2):
    xp1,yp1 = p1
    xp2,yp2 = p2

    x_lb = min(xp1,xp2) + int(abs(xp2 - xp1)/2)
    y_lb = min(yp1,yp2) + int(abs(yp2 - yp1)/2)

    if xp1 < xp2:
        if yp1 < yp2:
            x_lb += 0
            y_lb -= 0
        elif yp1 > yp2:
            x_lb -= 30
            y_lb -= 8
        else:
            x_lb -= 17
            y_lb -= 8
    elif xp1 > xp2:
        if yp1 < yp2:
            x_lb += 5
            y_lb += 10
        elif yp1 > yp2:
            x_lb += 5
            y_lb += 0
        else:
            x_lb -= 17
            y_lb += 18
    else:
        if yp1 < yp2:
            x_lb += 2
        elif yp1 > yp2:
            x_lb -= 33
    return x_lb,y_lb
```

Figura 252. Método ”obtener_coordenadas_labels”.

El método ”obtener_coordenadas_labels” obtiene las coordenadas de la etiqueta a colocar dependiendo de la pendiente de la curva. Esto es para los casos en que se debe colocar la etiqueta abajo o arriba y a la derecha o izquierda del punto.

6.4.2.4.5. Modificaciones en la clase graficador.py

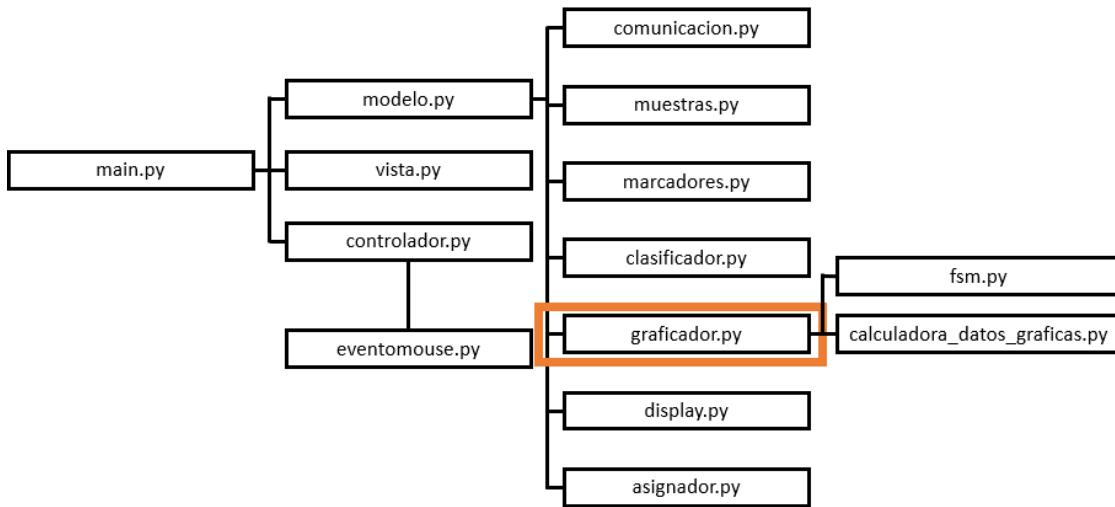


Figura 253. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

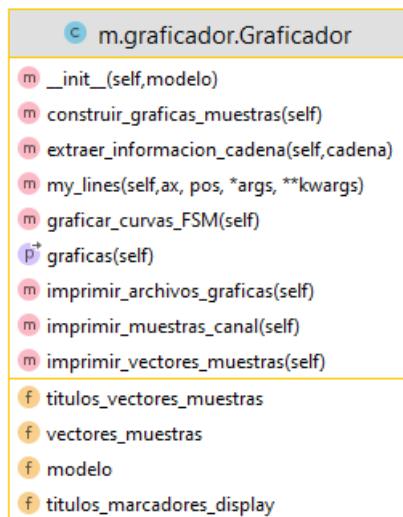


Figura 254. Métodos, atributos y propiedades de la clase graficador.py.

Donde:

- (c) indica que es una **clase**.
- (m) indica que es un **método**.
- (p*) indica que es una **propiedad**.
- (f) indica que es un **atributo**.

De esta clase se presenta el método “graficar_curvas_FSM”. Este método se puede dividir en 6 partes principales:

- i. **Definición de variables** útiles para la presentación de las gráficas.
- ii. Código para la obtención de la **Gráfica 0: Señales binarias en el tiempo**.
- iii. Código para la obtención de la **Gráfica 1: Tabla de estados binarios**.
- iv. Código para la obtención de la **Gráfica 2: Diagrama de estados (1)**.
- v. Código para la obtención de la **Gráfica 3: Diagrama de estados (2)**.
- vi. Código para la obtención de la **Gráfica 3: Diagrama de transiciones de la máquina de estados finitos**.

6.4.2.4.5.1 Definición de variables (i)

```
def graficar_curvas_FSM(self):  
    global graficas  
    graficas = []  
  
    muestras = self.vectores_muestras  
    muestras_llaves = list(muestras)  
    #print('muestras_llaves: '+str(muestras_llaves))  
  
    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras  
    # hay 6 curvas).  
    lim_max = len(muestras[muestras_llaves[0]])  
  
    fsm1 = fsm()  
    calc = calculadora_datos_graficas()
```

Figura 255. Fragmento de código para definir las variables a utilizar.

Al principio del método se definen algunas variables útiles como:

- **muestras:** Es una copia del diccionario **vectores_muestras** de las muestras tomadas de Arduino.
- **muestras_llaves:** Contiene una lista de las llaves del diccionario **vectores_muestras**.
- **lim_max:** Es la longitud de cada canal de muestras. En este caso es de 20 muestras.
- **fsm1:** Contiene los datos de la **máquina de estados finitos** a utilizar.

6.4.2.4.5.2 Gráfica 0: Señales binarias en el tiempo

La **gráfica 0** muestra la salida y el estado de la **máquina de estados finitos** al aplicarle una **entrada**.

```
# --- Grafica 0 : Entrada, Estado y Salida en el tiempo -----
plt.figure(0)
for i in range(0,len(muestras_llaves)):
    plt.subplot(3, 1, i+1)

    #Se colocan las lineas auxiliares de las graficas.
    self.my_lines('x',      list(range(len(muestras[muestras_llaves[i]])+1)),      color='5',
    linewidth=0.5)
    self.my_lines('y',      list(range(int(max(muestras[muestras_llaves[i]])+1))),  color='5',
    linewidth=0.5)

    # Se grafican entrada, estado y salida.
    plt.step(range(0,lim_max),muestras[muestras_llaves[i]],      linewidth      =
2,color=colores_lineas[i], where='post', label=muestras_llaves[i])

    # Se modifican los ejes para que muestren cada numero entero.
    plt.xticks(np.arange(0,len(muestras[muestras_llaves[i]])+1,step = 1))
    plt.yticks(np.arange(0,max(muestras[muestras_llaves[i]])+1,step = 1))

    plt.ylabel(muestras_llaves[i])
    plt.legend()

    plt.xlabel('Muestras')

plt.subplot(3, 1, 1)
plt.title("Señales logicas en el tiempo")

grafica = 'g/00_Senales_binarias_en_el_tiempo' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)
```

Figura 256. Fragmento de código para la **Gráfica 0: Señales binarias en el tiempo**

En este caso se utiliza una gráfica de tipo step para graficar las tres señales, de tal manera que sea posible comparar cada valor correspondientemente.

El resultado de esta parte es la siguiente gráfica:

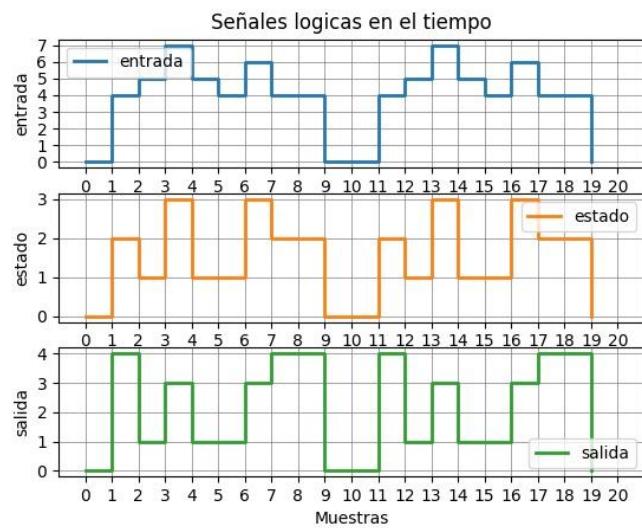


Figura 257. Gráfica 0: Señales binarias en el tiempo.

Se muestra en azul la **señal de entrada** en la parte superior de la gráfica, la **señal de estado** en amarillo en la parte media de la gráfica y la **señal de salida** en verde en la parte inferior de la gráfica.

6.4.2.4.5.3 Gráfica 1: Tabla de estados binarios

En esta **gráfica 1** se muestra el **comportamiento** de la **máquina de estados finitos**. Se muestra para cada **estado** su **salida**, y la **tabla de transiciones entre estados**.

```
# --- Grafica 1 : Tabla de estados -----
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# Se colocan los titulos de la tabla.
cv2.putText(canvas2,"Tabla de estados de la maquina de estados finitos", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Estado", (30, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Salida", (140, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Entrada", (250, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,":", (320, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Estado actual", (345, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"->", (465, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Estado futuro", (500, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# Se dibujan lineas para dividir la tablas.
cv2.rectangle(canvas2, (20,50),(620,48),(0,0,0), -1, cv2.LINE_AA)
cv2.rectangle(canvas2, (20,71),(620,73),(0,0,0), -1, cv2.LINE_AA)

ajuste_espacio = 0
str_transiciones = [""] for i in range(0,len(fsm1.tabla_transiciones_estado_futuro))]
estados_transiciones = [[[]]] for i in range(0,len(fsm1.tabla_transiciones_estado_futuro))]

for i in range(0,fsm1.num_estados):
    # Se obtiene el estado y salida correspondiente.
    str_bin_estado = calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
```

```

    str_bin_salida
calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida) =  

        # Se escriben los textos del estado y salida correspondiente.
        cv2.putText(canvas2,"q"+str(i)+":           "+str_bin_estado,          (30,
70+(i+ajuste_espacio+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1,
cv2.LINE_AA)
        cv2.putText(canvas2,str_bin_salida,                      (140,
70+(i+ajuste_espacio+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1,
cv2.LINE_AA)  

        #print("fsm1.tabla_transiciones_estados["+str(i)+"]:
"+str(fsm1.tabla_transiciones_estados[i])+"
"+str(len(fsm1.tabla_transiciones_estados[i]))) len:  

j = 0  

while j < len(fsm1.tabla_transiciones_estado_futuro[i]):  

    # Se obtiene el texto de la transicion correspondiente, si hay mas de una posible,
se agregan condiciones de no importa con "X".
    transiciones =  

calc.list_duplicates_of(fsm1.tabla_transiciones_estado_futuro[i],fsm1.tabla_transiciones_e
stado_futuro[i][j])  

    diff = calc.obtener_bits_cambiantes(fsm1,transiciones,i)
    str_transicion = calc.obtener_str_de_binario_transicion(fsm1.bits_transicion-1,fsm1.tabla_transiciones_estados[i][transiciones[0]],diff)  

y_coor = 70+(i+ajuste_espacio+1)*20  

# Se colocan los textos para la parte de transiciones de estados.
    cv2.putText(canvas2,str_transicion,          (265,
y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,":", (320, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0,
0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"q"+str(i),          (395,
y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"->", (465, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)  

cv2.putText(canvas2,"q"+str(fsm1.tabla_transiciones_estado_futuro[i][transiciones[0]]),
(542, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)  

# Se guardan los textos transiciones calculadas y los estados a los que cambia.
    str_transiciones[i].append(str_transicion)
    estados_transiciones[i].append(fsm1.tabla_transiciones_estado_futuro[i][j])  

j = j+len(transiciones)
ajuste_espacio += 1  

str_transucion = "

```

```

# Ajuste de las listas, ya que el primer dato no contiene informacion util.
str_transiciones[i].pop(0)
estados_transiciones[i].pop(0)

# Se colocan lineas que dividen cada estado.
cv2.rectangle(canvas2, (238,48),(240,68+(i+ajuste_espacio+1)*20),(0,0,0), -1,
cv2.LINE_AA)
cv2.rectangle(canvas2,
(20,65+(i+ajuste_espacio+1)*20),(620,67+(i+ajuste_espacio+1)*20),(0,0,0), -1,
cv2.LINE_AA)

# Se colocan textos de notas adicionales.
cv2.putText(canvas2,"Los estados son: "+str(fsm1.tabla_nombres_estados),(20,
390),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las salidas son: "+str(fsm1.tabla_nombres_salidas),(20,
405),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las entradas son: "+str(fsm1.tabla_nombres_entradas),(20,
420),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"La tabla de transiciones indica con cual entrada pasa de un
estado a otro estado. ",(20, 445),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Si en la entrada hay bits en X, significa que no importa el valor
de ese bit para pasar al siguiente estado. ",(20,
460),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de entrada no incluidas en la tabla indican
que el estado no cambia. ",(20, 475),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)

#print("str_transiciones: "+str(str_transiciones))
#print("estados_transiciones: "+str(estados_transiciones))

filename = "g/01_Tabla_estados_binarios" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

```

Figura 258. Fragmento de código para la **Gráfica 1: Tabla de estados binarios**

Este parte realiza lo siguiente:

1. Imprime los **títulos de la tabla**.
2. Para cada **estado** de la máquina de estados finitos realiza:
 - a. **Imprime el estado** y su **salida** correspondiente en binario.
 - b. Para cada **transición** hacia otros estados realiza:
 - **Imprime la entrada** correspondiente en binario y el **estado siguiente** al que cambiará.
3. **Imprime una leyenda** para entender mejor la tabla. Contiene los nombres de cada bit de los bits de entrada, salida y los estados.

El resultado de esta parte es la siguiente gráfica:

Tabla de estados de la máquina de estados finitos					
Estado	Salida	Entrada :	Estado actual	→	Estado futuro
q0: 00	000	10X :	q0	→	q1
q1: 01	100	0XX :	q1	→	q0
		1X1 :	q1	→	q2
q2: 10	001	0XX :	q2	→	q0
		11X :	q2	→	q3
q3: 11	011	0XX :	q3	→	q0
		100 :	q3	→	q1
		101 :	q3	→	q2

Los estados son: ['q0 Apagado', 'q1 Encendido funcionando', 'q2 Corto circuito', 'q3 Reset']

Las salidas son: ['bit 2: Relevador ON/OFF', 'bit 1: LED de Reset', 'bit 0: LED de Corto circuito']

Las entradas son: ['bit 2: Boton de Alimentación', 'bit 1: Boton de Reset', 'bit 0: Sensor de Corto Circuito']

La tabla de transiciones indica con cual entrada pasa de un estado a otro estado.

Si en la entrada hay bits en X, significa que no importa el valor de ese bit para pasar al siguiente estado.

Las combinaciones de entrada no incluidas en la tabla indican que el estado no cambia.

Figura 259. Gráfica 1: Tabla de estados binarios

Se muestra la tabla del **comportamiento de la máquina de estados finitos**. De izquierda a derecha se muestra el estado, la salida y las transiciones hacia otros estados. En la parte inferior de la tabla se muestra una leyenda con información adicional para entender la tabla.

6.4.2.4.5.4 Gráfica 2: Diagrama de estados (1)

En esta **gráfica 2** se muestra un **diagrama de estados** de la máquina de estados finitos, sin utilizar el paquete graphviz.

```
# --- Grafica 2 : Diagrama de estados -----
-----
# Opcion 1. Diagrama de estados con funciones desarrolladas en
calculadora_datos_graficas.py, sin utilizar la librería graphviz
# --- Generacion de un canvas de w = 480 x h = 640 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

# Se coloca el titulo.
cv2.putText(canvas2,"Diagrama de estados de la maquina de estados finitos", (30,
35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)

# Se colocan las coordenadas de los circulos a utilizar, uno por cada estado y otros
parametros.
xcoords = [int(640/8),int(640*7/8),320,320]
ycoords = [90,90,220,380]
length_lineas = 10
c_radio = [40,58]

for i in range(0,fsm1.num_estados):
    # Se dibuja el circulo del estado actual.
    #print("estado actual: "+str(i))
    cv2.circle(canvas2, (xcoords[i],ycoords[i]), c_radio[0], (0,0,0), 2)
    #cv2.circle(canvas2, (xcoords[i],ycoords[i]), c_radio[1], (0,0,0), 2)

    # Se obtienen los textos del estado actual y de la salida.
    estado = calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
    salida = calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida)

    #cv2.line(canvas2,(xcoords[i],ycoords[i]-50),(xcoords[i],ycoords[i]+50),(0,0,0), 3)

    # Se colocan los textos del estado actual y de la salida
    cv2.putText(canvas2,"q"+str(i)+":"+str(estado),(xcoords[i]-25,ycoords[i]-
15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Salida: ",(xcoords[i]-
26,ycoords[i]+7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(salida),(xcoords[i]-
14,ycoords[i]+22),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
```

```

# Se escribe un texto adicional para indicar si es el estado inicial.
if i == fsm1.estado_inicial:
    cv2.putText(canvas2,"Estado inicial",(xcoords[i]-60,ycoords[i]+c_radio[1]),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# Se itera sobre las transiciones de este estado actual hacia otros.
for j in range(0,len(estados_transiciones[i])):
    #print("Estado a actualizar: "+str(j))

    coords_labels = 0

    # Se calculan las coordenadas de las flechas para indicar los cambios de estado.
    lineas_transiciones =
    calc.obtener_puntos_lineas_transicion(canvas2,xcoords,ycoords,c_radio,length_lineas,i,e
    stados_transiciones[i][j])
    #print("lineas_transiciones : "+str(lineas_transiciones))

    # Se calcula la distancia de las flechas con las coordenadas anteriores.
    d = calc.distancia_entre_puntos(lineas_transiciones[0],lineas_transiciones[2])

    # Se calcula el ajuste para la cabeza de la flecha (para que todas las flechas
    tengan el mismo tamano de flecha).
    tip_l = 10 / d
    #print(d)

    # Puesto que se calcularon coordenadas para dos flechas (una de ida y otra de
    vuelta), se elige alguna de las dos.
    # Si el estado actual es menor que el estado al que cambia, se utiliza la primera
    flecha, de lo contrario, se utiliza la segunda.
    # Se dibuja la flecha.
    if i > estados_transiciones[i][j]:
        canvas2 =
        cv2.arrowedLine(canvas2,tuple(lineas_transiciones[0]),tuple(lineas_transiciones[2]),(0,0,0)
        , 3, tipLength = tip_l)
        coords_labels = 0
    else:
        canvas2 =
        cv2.arrowedLine(canvas2,tuple(lineas_transiciones[1]),tuple(lineas_transiciones[3]),(0,0,0)
        , 3, tipLength = tip_l)
        coords_labels = 1

    # Se coloca el texto correspondiente al numero binario con el que cambia de
    estado.

    cv2.putText(canvas2,str_transiciones[i][j],calc.obtener_coordenadas_labels(lineas_transici
    ones[coords_labels],lineas_transiciones[coords_labels+2]),cv2.FONT_HERSHEY_SIMPL
    EX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# Se escriben textos de notas adicionales.

```

```

cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio
de estado, indicado por el numero adyacente. " ,(20,
445),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de transiciones entre estados que no
aparecen en el diagrama indica que no hay cambio" ,(20,
460),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"de estado cuando ocurren.Si hay 'X' en el numero de una flecha
significa que no importa el valor de ese bit. " ,(20,
475),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/02_Diagrama_estados_binarios" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

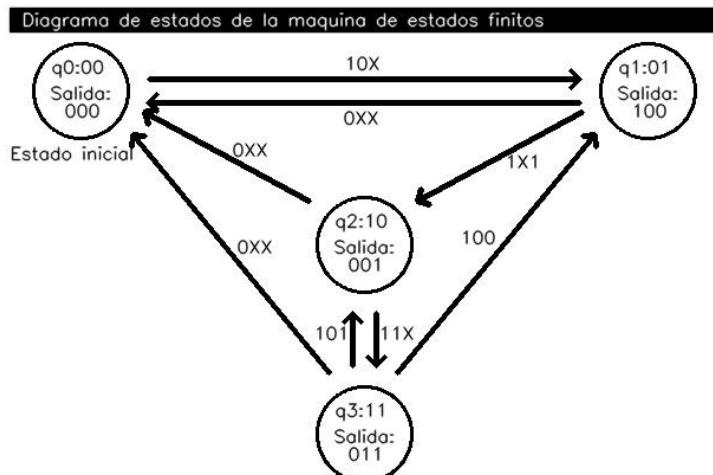
```

Figura 260. Fragmento de código para la **Gráfica 2: Diagrama de estados 1.**

Esta función realiza lo siguiente:

1. Se obtienen las **coordenadas x y y** del centro de cada uno de los **círculos** en donde se colocará el estado y su salida.
2. Para cada **estado** se realiza lo siguiente:
 - a. Se **imprimen** los textos del **estado** y de la **salida** en binario en cada círculo. Si es el estado inicial, se indica por medio de un texto.
 - b. Por cada **transición** hacia otro estado se realiza lo siguiente:
 - i. Se **obtienen** las **coordenadas de la línea** que une al círculo del estado actual con el círculo del estado siguiente.
 - ii. Se obtiene el **tamaño de la cabeza de la flecha**.
 - iii. Se **traza la línea** que une ambos círculos
 - iv. Se **imprime** el texto de la **entrada** correspondiente en binario con el que cambia hacia ese estado.
3. Se **imprime una leyenda**, explicando el diagrama de estados.

El resultado de esta parte es la siguiente gráfica:



Cada círculo es un estado. Cada flecha representa un cambio de estado, indicado por el número adyacente. Las combinaciones de transiciones entre estados que no aparecen en el diagrama indican que no hay cambio de estado cuando ocurren. Si hay 'X' en el número de una flecha significa que no importa el valor de ese bit.

Figura 261. Gráfica 2: Diagrama de estados 1.

Se muestra el **diagrama de estados**, en donde cada **círculo** es un **estado**. En cada circulo se muestra el **número del estado** y su **salida** correspondiente. Además, se observan las **flechas de transición entre estados**, así como el valor de **entrada** en binario correspondiente para hacer el **cambio de estado**.

6.4.2.4.5.5 Gráfica 3: Diagrama de estados (2)

En esta **gráfica 3** se muestra un **diagrama de estados** de la máquina de estados finitos utilizando el paquete graphviz.

El código para esta gráfica se explicará en 4 partes:

- i. **Parte i: Generación del diagrama de estados utilizando graphviz.**
- ii. **Parte ii. Formato de las dimensiones de la imagen del diagrama de estados.**
- iii. **Parte iii. Concatenación de título y leyenda a la imagen del diagrama de estados.**
- iv. **Parte iv. Eliminación de imágenes auxiliares.**

6.4.2.4.5.5.1 Generación del diagrama de estados utilizando graphviz (i)

```
# --- Grafica 2 : Diagrama de estados -----
-----
# Opcion 2. Diagrama de estados con funciones desarrolladas en
calculadora_datos_graficas.py, utilizando la librería graphviz

# Nombre del diagrama de estados
g = gv.Digraph('g/diagrama_estados',format='png')

# Configuracion del nodo inicial
g.node('ini', shape="point")

# Configuracion del diagrama de estados, que vaya de izquierda a derecha
g.graph_attr['rankdir'] = 'LR'

str_completo = [["] for i in range(0,fsm1.num_estados)] 

for i in range(0,fsm1.num_estados):

    # Calculo de las variables estado y salida en binario
    estado = calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
    salida = calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida)

    # Concatenacion del string completo para cada estado
    str_completo[i] = 'Estado
'+str(estado)+'\n'+str(fsm1.tabla_nombres_estados[i])+'\n'+ "Salida "+str(salida)

    # Se asignan los nombres de los estados.
    # El estado final tiene forma de doble circulo, mientras que se indica con una flecha
    con punto para el estado inicial
    if i == fsm1.estado_final:
        g.node(str_completo[i], shape="doublecircle")
```

```

else:
    g.node(str_completo[i])
    if i == fsm1.estado_inicial:
        g.edge('ini',str_completo[i])
# Se asignan las flechas de transicion entre estados
for i in range(0,fsm1.num_estados):
    for j in range(0,len(estados_transiciones[i])):
        #print('flecha      de      A:'+str(fsm1.tabla_nombres_estados[i])+'
B:'+str(fsm1.tabla_nombres_estados[estados_transiciones[i][j]])) a

g.edge(str_completo[i],str_completo[estados_transiciones[i][j]],str(str_transiciones[i][j]))

# Genera el diagrama, pero no lo muestra en el visor de fotos
g.render(view=False)

```

Figura 262. Fragmento de código para la **Gráfica 3: Diagrama de estados 2 (parte i)**.

En esta **parte i**, se utiliza el paquete graphviz para generar el diagrama de estados.

Para cada estado se realiza lo siguiente:

1. Se **definen** los **estados** por el **número de estado** y su **salida**.
2. Se añade cada **estado** como un **nodo nuevo**. Casos especiales:
 - a. Si es el estado inicial, se indica con una flecha.
 - b. Si es el estado final, se indica por un doble círculo.
3. Se asignan las **transiciones entre estados**. Para las transiciones de cada estado se realiza lo siguiente:
 - a. Se **imprime** una **flecha** del estado actual al estado al que cambia. Se utilizan los textos de las transiciones obtenidas para la gráfica 1 descrito en la sección 6.4.2.4.5.3.
4. Se genera el diagrama de estados, aunque no se visualiza en pantalla.

6.4.2.4.5.5.2 Formato de las dimensiones de la imagen del diagrama de estados (ii)

```
# Se lee la imagen del diagrama de estados generado
diagrama = cv2.imread('g/diagrama_estados.gv.png')
#height, width, channels = image.shape
scale_percent = 0
dim = 0

# Se recalculan las dimensiones de la imagen.
# Se pregunta si se recalculan los limites para que ocupe todo el largo (640) o todo el alto (320) establecidos
# Se calcula el porcentaje de escalamiento de la imagen
if(diagrama.shape[1]*320/diagrama.shape[0] <= 640):
    scale_percent = 320/diagrama.shape[0]
    dim = (int(diagrama.shape[1]*scale_percent), 320)
else:
    scale_percent = 640/diagrama.shape[1]
    dim = (640, int(diagrama.shape[0]*scale_percent))

# Se asignan las nuevas dimensiones de la imagen, de acuerdo al porcentaje de estaca calculado anteriormente
diagrama_resized = cv2.resize(diagrama, dim, interpolation = cv2.INTER_AREA)
#print('scale percent: ',scale_percent)
#print('diagrama_resized Dimensions : ',diagrama_resized.shape)

# Se guarda la imagen escalada
filename = "g/diagrama_estados_resized" + ".jpg"
cv2.imwrite(filename,diagrama_resized)

#cv2.imshow("Resized image", resized)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

# En caso de requerirlo, se concatenan partes en blanco a la imagen del diagrama de estados a los lados
if(diagrama_resized.shape[1] < 640):
    canvas2 = np.ones((diagrama_resized.shape[0],int((640-diagrama_resized.shape[1])/2),3), dtype = "uint8")*255
    filename = "g/blanco1" + ".jpg"
    cv2.imwrite(filename,canvas2)

# Se verifica si se necesitan dos imagenes de igual tamaño, esto es si las dimensiones de la imagen del diagrama de estados es par
# De lo contrario se crean dos imagenes en blanco de diferente tamano
if diagrama_resized.shape[1]%2 == 1:
    canvasaux = np.ones((diagrama_resized.shape[0],int((640-diagrama_resized.shape[1])/2)+1,3), dtype = "uint8")*255
    #print("blanco2 = blanco1 + 1")
```

```

filename = "g/blanco2" + ".jpg"
cv2.imwrite(filename,canvasaux)
else:
    #print("blanco2 = blanco1")
    filename = "g/blanco2" + ".jpg"
    cv2.imwrite(filename,canvas2)

img_blanco1 = cv2.imread('g/blanco1.jpg')
img_blanco2 = cv2.imread('g/blanco2.jpg')
#print('blanco1 Dimensions : ',img_blanco1.shape)
#print('blanco2 Dimensions : ',img_blanco2.shape)

im_h = cv2.hconcat([img_blanco1,diagrama_resized,img_blanco2])
#print('im_h Dimensions : ',im_h.shape)
else:
    im_h = diagrama_resized.copy()
    #print('im_h Dimensions : ',im_h.shape)

```

Figura 263. Fragmento de código para la **Gráfica 3: Diagrama de estados 2 (parte ii)**.

Esta **parte ii** manipula la imagen del diagrama de estados creada para que tenga las mismas dimensiones que las demás gráficas (640 (ancho) x 480 (alto) pixeles):

1. Lee la imagen del diagrama de estados.
2. Escala la imagen para que tenga 640 pixeles de ancho o 320 pixeles de alto. Se elige alguna de las dos opciones con base en el mayor porcentaje de escalado.
3. Si se escaló a 320 pixeles de alto, se concatenan dos imágenes en blanco a sus lados, para que la imagen tenga ahora dimensiones de 640 x 320 pixeles.

6.4.2.4.5.5.3 Concatenación de título y leyenda a la imagen del diagrama de estados (iii)

```
# Se crea la imagen del titulo del diagrama de estados
canvas2 = np.ones((60,640,3), dtype = "uint8")*255
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)
cv2.putText(canvas2,"Diagrama de estados de la maquina de estados finitos", (30,
35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)
filename = "g/titulo" + ".jpg"
cv2.imwrite(filename,canvas2)
img_titulo = cv2.imread('g/titulo.jpg')

# Se crea la imagen que contiene las anotaciones del diagrama de estados
canvas2 = np.ones((420-diagrama_resized.shape[0],640,3), dtype = "uint8")*255
cv2.putText(canvas2,"Los estados son: "+str(fsm1.tabla_nombres_estados),(20, 330-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las salidas son: "+str(fsm1.tabla_nombres_salidas),(20, 345-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las entradas son: "+str(fsm1.tabla_nombres_entradas),(20, 360-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)

cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio
de estado, indicado por el numero adyacente.",(20, 385-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de transiciones entre estados que no
aparecen en el diagrama indica que no hay cambio",(20, 400-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"de estado cuando ocurren.Si hay 'X' en el numero de una flecha
significa que no importa el valor de ese bit.",(20, 415-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
filename = "g/anotaciones" + ".jpg"
cv2.imwrite(filename,canvas2)

# Se leen las imagenes de las anotaciones y del titulo
img_anotaciones = cv2.imread('g/anotaciones.jpg')
img_titulo = cv2.imread('g/titulo.jpg')

# Se concatenan las imagenes del diagrama de estados, titulo y anotaciones
im_v = cv2.vconcat([img_titulo,im_h,img_anotaciones])

# Se guarda la imagen final
filename = "g/03_Diagrama_estados_binarios_alternativo" + ".jpg"
cv2.imwrite(filename,im_v)
```

```
graficas.append(filename)
```

Figura 264. Fragmento de código para la **Gráfica 3: Diagrama de estados 2 (parte iii).**

Esta **parte iii** agrega lo siguiente:

1. **Agrega** un texto con el **título** en la parte superior de la imagen de 60 x 640 pixeles.
Ahora sus dimensiones son de 380 x 640 pixeles.
2. **Agrega** un texto con la **descripción** del diagrama en la parte inferior de la imagen.
En este punto, el diagrama de estados ya tiene 480 x 640 pixeles.

6.4.2.4.5.5.4 Eliminación de imágenes auxiliares (iv)

```
# Se obtiene la ruta desde donde se ejecuta el archivo main.py
path = os.path.abspath(os.getcwd())
#print(path)

# Se eliminan las imágenes que ya no se van a usar, utilizando la ruta calculada
# anteriormente
for i in
['g/diagrama_estados.gv','g/diagrama_estados.gv.png','g/diagrama_estados_resized.jpg','
g/blanco1.jpg','g/blanco2.jpg','g/titulo.jpg','g/anotaciones.jpg']:

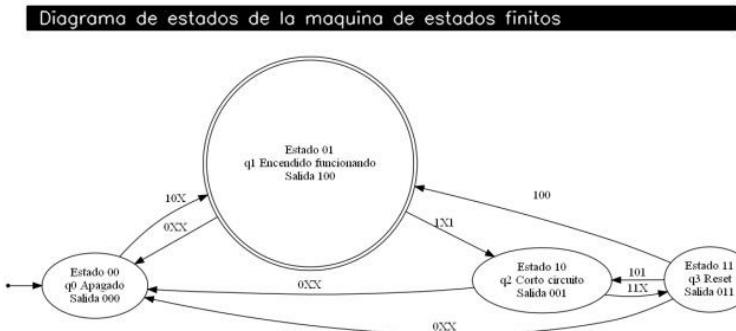
    try:
        os.remove(path+'/'+i)
    except:
        #print("Archivo "+i+" no removido")
    pass
```

Figura 265. Fragmento de código para la **Gráfica 3: Diagrama de estados 2 (parte iv).**

Puesto que se crearon imágenes adicionales que ya no se van a necesitar, se eliminan.

6.4.2.4.5.5.5 Presentación de la gráfica

El resultado de esta parte es la siguiente gráfica:



Los estados son: ['q0 Apagado', 'q1 Encendido funcionando', 'q2 Corto circuito', 'q3 Reset']

Las salidas son: ['bit 2: Relevador ON/OFF', 'bit 1: LED de Reset', 'bit 0: LED de Corto circuito']

Las entradas son: ['bit 2: Botón de Alimentación', 'bit 1: Botón de Reset', 'bit 0: Sensor de Corto Circuito']

Cada círculo es un estado. Cada flecha representa un cambio de estado, indicado por el número adyacente. Las combinaciones de transiciones entre estados que no aparecen en el diagrama indican que no hay cambio de estado cuando ocurren. Si hay 'X' en el número de una flecha significa que no importa el valor de ese bit.

Figura 266. Gráfica 3: Diagrama de estados 2.

Se muestra el **diagrama de estados**, en donde cada **círculo** es un **estado**. En cada **círculo** se muestra el **nombre del estado**, el **número del estado** y su **salida** correspondiente. Además, se observan las **flechas de transición entre estados**, así como el valor de **entrada** correspondiente para hacer el **cambio de estado**.

6.4.2.4.5.6 Gráfica 4: Diagrama de transiciones de la máquina de estados finitos

En esta **gráfica 4** se observa tanto la salida como el estado de la máquina de estados finitos al probarla con el vector de prueba definido en la sección 6.3.4.2.1.

```
# --- Grafica 3 : Voltaje y binario -----
# --- Generacion de un canvas de w = 480 x h = 640 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

# Se escribe el titulo.
cv2.putText(canvas2,"Transiciones de estados de acuerdo al vector de transiciones de
entrada utilizado", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1,
cv2.LINE_AA)
#cv2.circle(canvas2, (320,260), 215,(255,0,0),2)
#cv2.circle(canvas2, (320,260), 185,(255,0,0),2)

angle = np.linspace(0,2*math.pi,int(len(muestras[muestras_llaves[0]])/2+1))
#print("angle: "+str(angle))
#print("muestras_llaves[0]: "+str(muestras_llaves[0]))
#print("muestras_llaves[0]: "+str(muestras_llaves[0]))
#print("muestras_llaves[1]: "+str(muestras_llaves[1]))
#print("muestras_llaves[1]: "+str(muestras_llaves[1]))
#print("muestras_llaves[2]: "+str(muestras_llaves[2]))
#print("muestras_llaves[2]: "+str(muestras_llaves[2]))

# Dibuja cada uno de los circulos y la informacion del estado correspondiente que va
dentro de ella. Tambien dibuja las flechas entre circulos.
for i in range(0,int(len(muestras[muestras_llaves[0]])/2)):
    # Se obtienen las coordenadas del circulo
    xcoor = int(320+185*math.cos(angle[i]))
    ycoor = int(260+185*math.sin(angle[i]))

    # Se dibuja el circulo.
    cv2.circle(canvas2, (xcoor,ycoor), 35, (0,0,0), 2)

    # Se obtienen los textos del estado actual y de la salida a partir del vector de
muestras leido.
    str_bin_estado =
    calc.decimal_a_binario(int(muestras[muestras_llaves[1]][i]),math.ceil(math.log(fsm1.num_
estados,2)))
    str_bin_salida =
    calc.decimal_a_binario(int(muestras[muestras_llaves[2]][i]),fsm1.bits_salida)
```

```

# Se escriben los textos del estado actual y de la salida.

cv2.putText(canvas2,"q"+str(int(muestras[muestras_llaves[1]][i]))+":"+str_bin_estado,(xcoor-24,ycoor-15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Salida:",(xcoor-26,ycoor+7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,str_bin_salida,(xcoor-15,ycoor+22),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# Encuentra las coordenadas de la flecha entre dos circulos.

i1 = calc.get_intersections(320,260,180,xcoor,ycoor,40)
i2 = calc.get_intersections(320,260,180,int(320+185*math.cos(angle[i+1])),int(260+185*math.sin(angle[i+1])),40)
dist = 100
flecha_coords = [0]*4
"for j in range(0,1):
    if math.dist([i1[j]*2],i1[j*2+1],[i2[j]*2],i2[j*2+1]) < dist:
        print('1, j=' + str(j))
        dist = math.dist([i1[j]*2],i1[j*2+1],[i2[j]*2],i2[j*2+1])
        flecha_coords[0] = i1[j]*2
        flecha_coords[1] = i1[j*2+1]
        flecha_coords[2] = i2[j]*2
        flecha_coords[3] = i2[j*2+1]
    if math.dist([i1[(2-j)*2]],i1[(2-j)*2+1],[i2[j]*2],i2[j*2+1]) < dist:
        print('2, j=' + str(j))
        dist = math.dist([i1[j]*2],i1[j*2+1],[i2[j]*2],i2[j*2+1])
        flecha_coords[0] = i1[(2-j)*2]
        flecha_coords[1] = i1[(2-j)*2+1]
        flecha_coords[2] = i2[j]*2
        flecha_coords[3] = i2[j*2+1]"
# Se puede comprobar con el codigo comentado que esta es la solucion que proporciona las coordenadas correctas.

flecha_coords[0] = i1[2]
flecha_coords[1] = i1[3]
flecha_coords[2] = i2[0]
flecha_coords[3] = i2[1]

# Se calcula un ajuste a las coordenadas de las etiquetas de las flechas.
fxcoor = int((flecha_coords[0] + flecha_coords[2])/2 + int(-15+32*math.cos(angle[i])))
fycoor = int((flecha_coords[1] + flecha_coords[3])/2 + int(7+18*math.sin(angle[i])))
#print('coordenadas de la flecha: '+str(flecha_coords))
#print('distancia: '+str(dist))

# Dibuja la flecha y escribe el texto correspondiente.

canvas2 = cv2.arrowedLine(canvas2,(int(flecha_coords[0]),int(flecha_coords[1])),(int(flecha_coords[2]),int(flecha_coords[3])),(0,0,0), 3, tipLength = 0.25)
str_bin_entrada_transicion = calc.decimal_a_binario(int(muestras[muestras_llaves[0]][i+1]),fsm1.bits_salida)

```

```

cv2.putText(canvas2,str_bin_entrada_transicion,(fxcoor,fycoor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

#print("str_bin_estado: "+str(str_bin_estado)+str(str_bin_entrada_transicion))
# Si es el primer estado el que se dibuja, se anade el texto que indique que es el
inicio.
if i == 0:
    canvas2
    cv2.arrowedLine(canvas2,(xcoor+40+60,ycoor),(xcoor+40,ycoor),(0,0,0), 3, tipLength = 0.25)
    cv2.putText(canvas2,"Inicio",(xcoor+55,ycoor-7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio de estado, indicado por el numero adyacente.",(20,477),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/04_Trancisiones_fsm" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Auxiliar para depuracion ---
plt.show()

```

Figura 267. Fragmento de código para la **Gráfica 4: Diagrama de transiciones de la máquina de estados finitos.**

En esta parte, se realiza lo siguiente:

1. Se genera una circunferencia de radio grande del tamaño de la imagen a mostrar.
2. Se obtienen las **coordenadas x y** y del **centro** de uno de los círculos en donde se colocarán los diferentes estados, basado en la circunferencia grande.
3. Para cada muestra se realiza lo siguiente:
 - a. Se **imprimen** los textos del **estado** y de la **salida** en binario en cada círculo. Si es el estado inicial, se indica por medio de una flecha.
 - b. Se **calculan** las **coordenadas** del **siguiente círculo** asociada a la siguiente muestra.
 - c. Se **imprime** la **flecha** que une al círculo actual con el siguiente y el texto de entrada correspondiente a la siguiente muestra.
 - d. Si es el primer círculo que se dibuja, se agrega el texto de inicio y se agrega una flecha.

El resultado de esta parte es la siguiente gráfica:

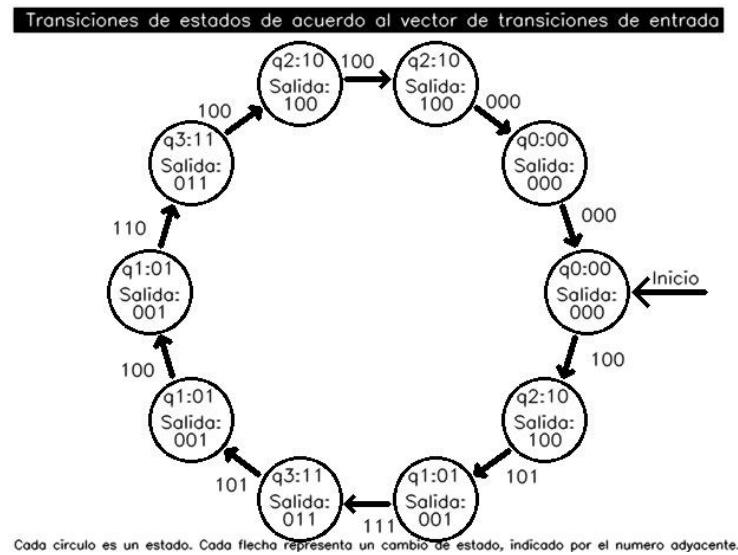


Figura 268. Gráfica de transiciones de la máquina de estados finitos.

Se muestra el **diagrama** con el **vector de prueba** aplicado a la máquina de estados finitos. En cada círculo se muestra el **estado actual** de la máquina de estados y su **salida** correspondiente. Cada **flecha** con su respectivo **número** significa la **entrada** con la que la máquina **cambia al siguiente estado**.

7. Resultados

Para esta sección se presentarán las capturas de pantalla de las gráficas sobre la imagen de cada uno de los 4 circuitos desarrollados:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

7.1. Trazador de Curvas (a)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

7.1.1 Presentación de la imagen del circuito con las gráficas

Para el **circuito Trazador de Curvas (a)**, se presentan 7 gráficas. Al ejecutar los programas de Arduino y Python, se abre la siguiente ventana:

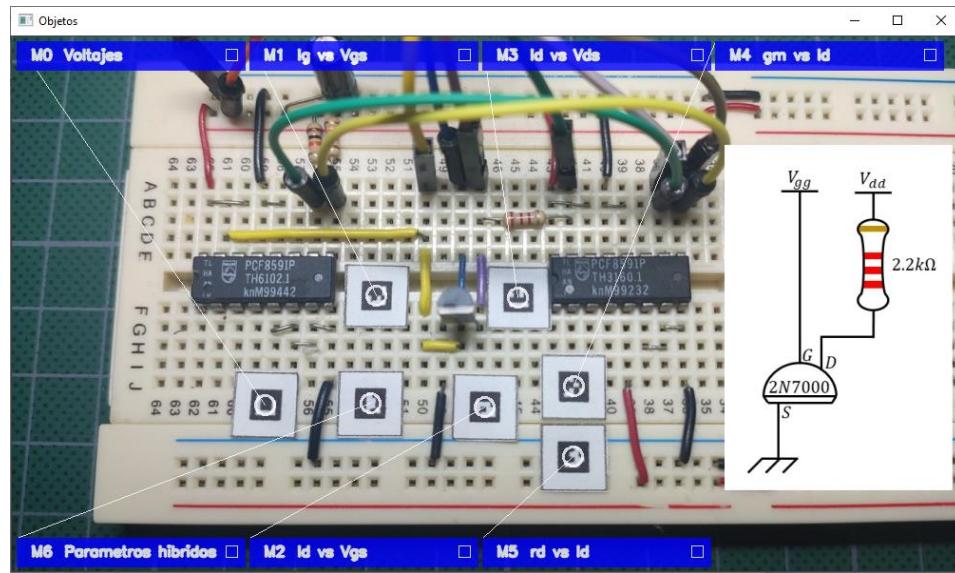


Figura 269. Ventana inicial con la imagen del circuito.

Se muestra la imagen del circuito con los marcadores y sobre de ella se muestran 7 pestanas.

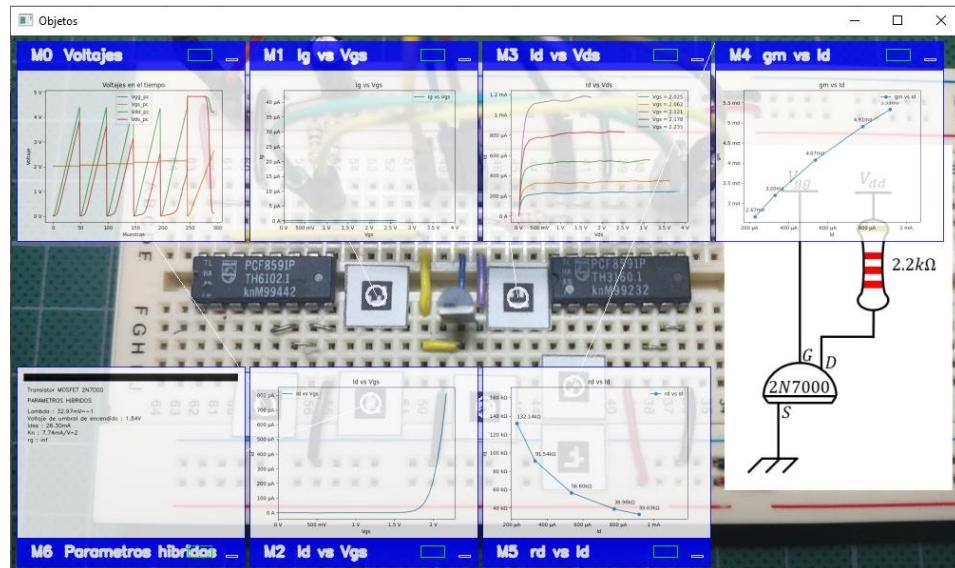


Figura 270. Ventanas desplegadas de todas las gráficas del circuito.

Al abrir cada una de las **7 pestañas**, se despliegan las **7 gráficas** anteriormente descritas.

7.1.2. Gráfica 0: Voltajes en el tiempo

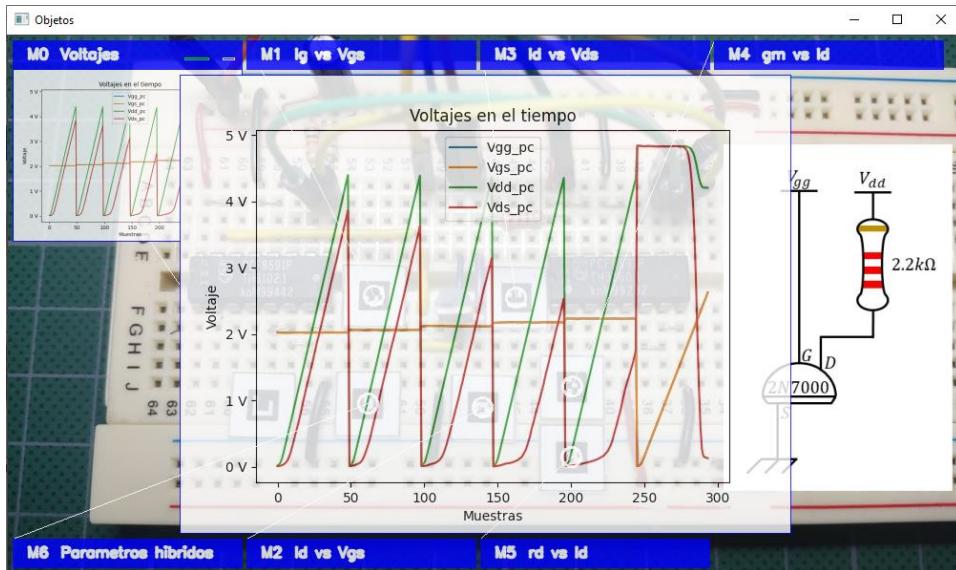


Figura 271. Presentación de la **Gráfica 0: Voltajes en el tiempo**.

Se observan los **voltajes** V_{gg} (azul), V_{gs} (naranja), V_{dd} (verde) y V_{ds} (rojo). Se tomaron **300 muestras** divididas en **6 series** de 50 muestras.

Esta **gráfica 0** permite observar el comportamiento de cada voltaje en el tiempo y si se toman correctamente las muestras del circuito.

Se observa para las primeras 250 muestras o primeras 5 series que la forma de onda de este voltaje V_{ds} es similar al voltaje V_{dd} . Con cada incremento del voltaje V_{gs} , el voltaje V_{ds} tarda más en seguir a la forma de onda del voltaje V_{dd} .

Por otro lado, en la 6ta serie de muestras tomadas (250 a 299) se observa que al incrementar V_{gs} desde 0V, hay un momento en el que el voltaje V_{ds} disminuye. También se observa que el voltaje V_{dd} también disminuye, aunque no debería de suceder por ser un voltaje de alimentación.

7.1.3. Gráfica 1: Ig vs Vgs

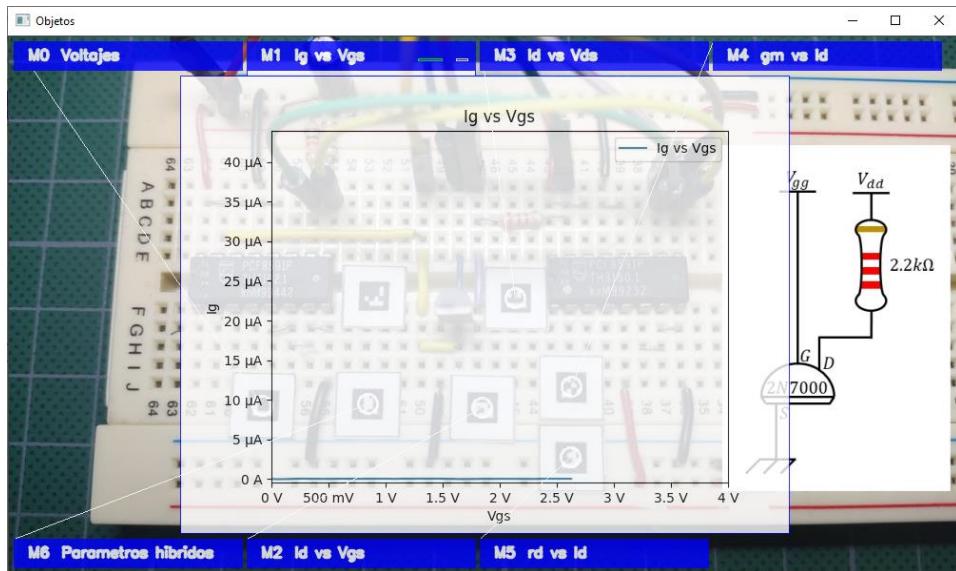


Figura 272. Presentación de la **Gráfica 1: Ig vs Vgs**.

En esta **gráfica 1** se observa en azul la **curva característica de entrada I_g vs V_{gs}** del transistor MOSFET de Enriquecimiento. Puesto que es un transistor controlado por voltaje y no por corriente, la **gráfica 1** muestra que se requiere con un valor muy pequeño de corriente I_g para que funcione el transistor MOSFET.

7.1.4. Gráfica 2: Id vs Vgs

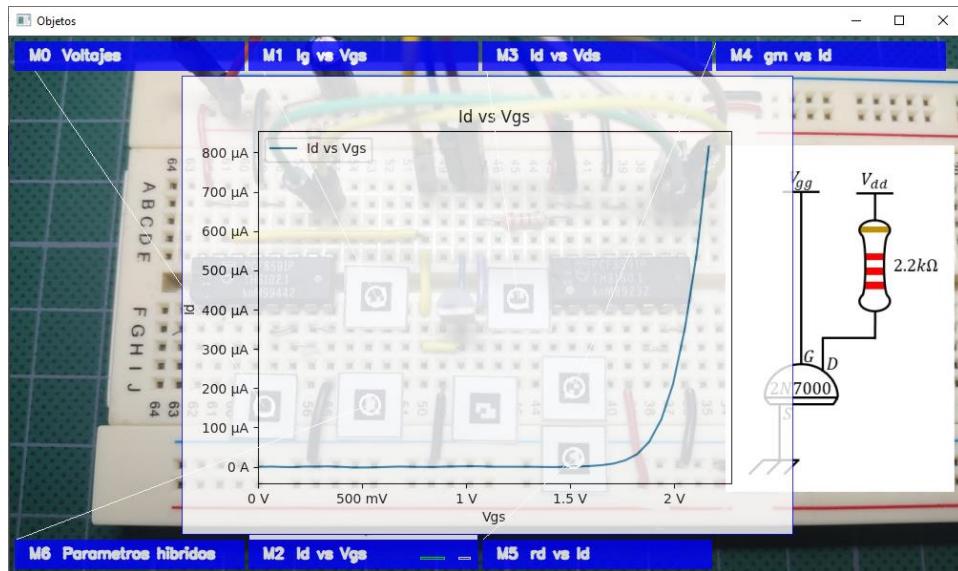


Figura 273. Presentación de la **Gráfica 2: Id vs Vds.**

En esta **gráfica 2** se observa en azul la **curva característica de transferencia I_d vs V_{gs}** del transistor MOSFET de enriquecimiento. Se observa que a partir de un cierto valor de voltaje V_{gs} empieza a circular una corriente I_d .

7.1.5. Gráfica 3: Id vs Vds

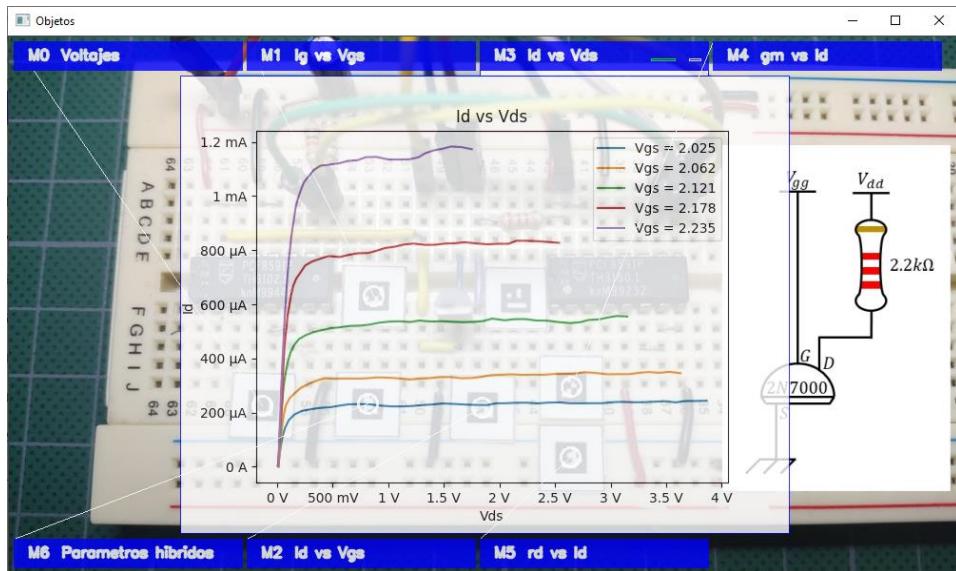


Figura 274. Presentación de la **Gráfica 3: Id vs Vds**.

En esta **gráfica 3** se observa un conjunto de **5 curvas características de salida I_d vs V_{ds}** del transistor MOSFET en azul, naranja, verde, rojo y morado. Cada curva se obtuvo al fijar un voltaje V_{gs} (mostrado en la leyenda de la gráfica) y se observa que cada curva tiene una corriente I_d diferente más o menos constante.

7.1.6. Gráfica 4: gm vs Id

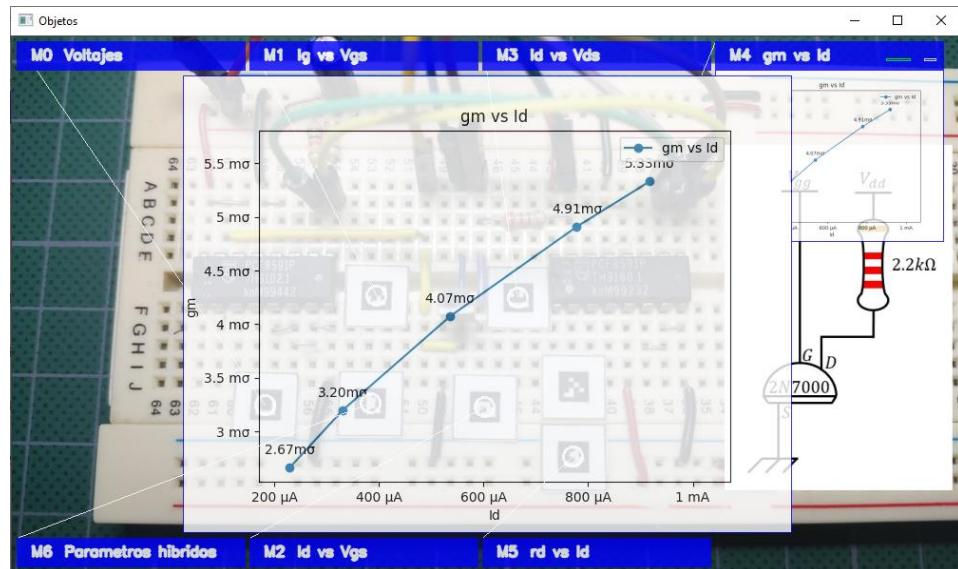


Figura 275. Presentación de la **Gráfica 4: gm vs Id**

En esta **gráfica 4** se observa en azul el comportamiento del parámetro de transconductancia g_m . Se calcularon 5 valores de g_m correspondientes a cada una de las 5 curvas I_d vs V_{ds} presentadas en la figura 275.

7.1.7. Gráfica 5: r_d vs I_d

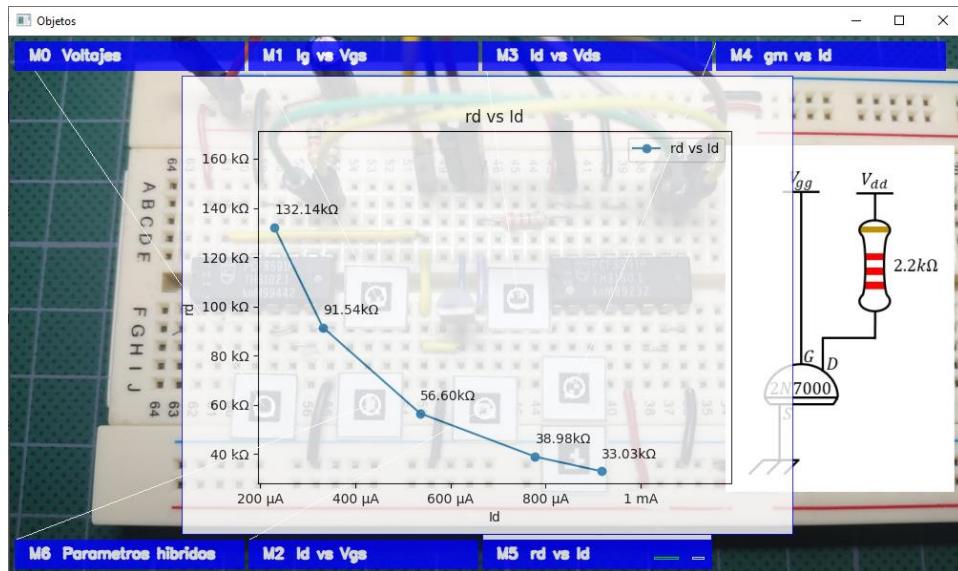


Figura 276. Presentación de la **Gráfica 5: r_d vs I_d** .

En esta **gráfica 5** se observa en azul el comportamiento de la **resistencia r_d** del modelo híbrido pi. Se calcularon 5 valores de r_d correspondientes a cada una de las 5 curvas I_d vs V_{ds} presentadas en la figura 275.

7.1.8. Gráfica 6: Parámetros híbridos

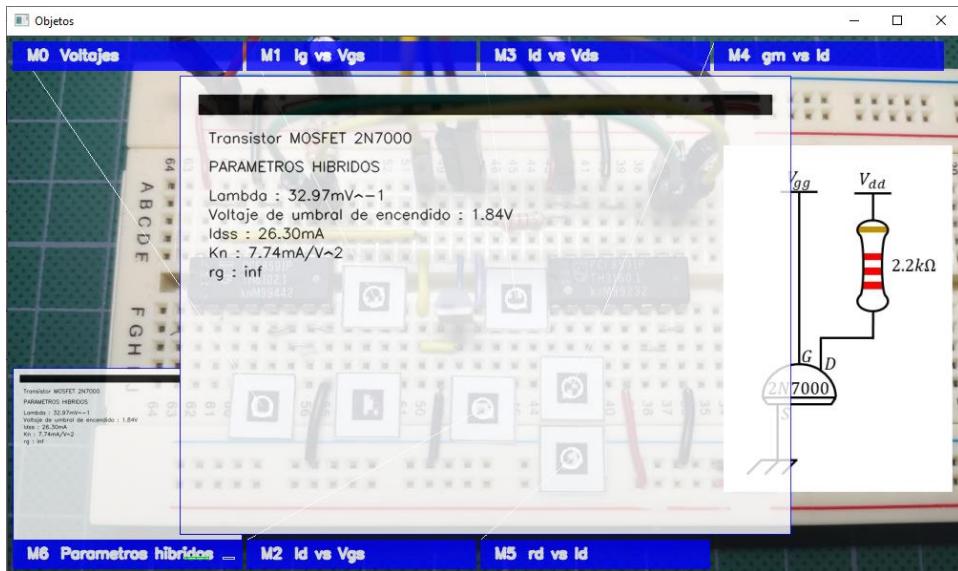


Figura 277. Presentación de la Gráfica 6: Parámetros híbridos.

En esta **gráfica 6** se muestra un resumen de los **parámetros híbridos** calculados del transistor. Se presentan los valores de los parámetros **lambda**, **voltaje de umbral de encendido** V_{th} , I_{dss} , K_n y r_g del transistor MOSFET.

Los valores resultantes de los parámetros híbridos son los siguientes:

Parámetro	Valor resultante
λ	$32.97 \frac{1}{\text{mV}}$
V_{th}	1.84 V
I_{dss}	26.3 mA
K_n	$7.74 \frac{\text{mA}}{\text{V}^2}$
r_g	∞

Tabla 36. Resultados de los parámetros híbridos del transistor MOSFET 2N7000.

7.2. Circuito generador de señal senoidal (b)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

7.2.1 Presentación de la imagen del circuito con las gráficas

Para el **circuito generador de señal senoidal (b)**, se presentan 4 gráficas. Al ejecutar los programas de Arduino y Python, se abre la siguiente ventana:

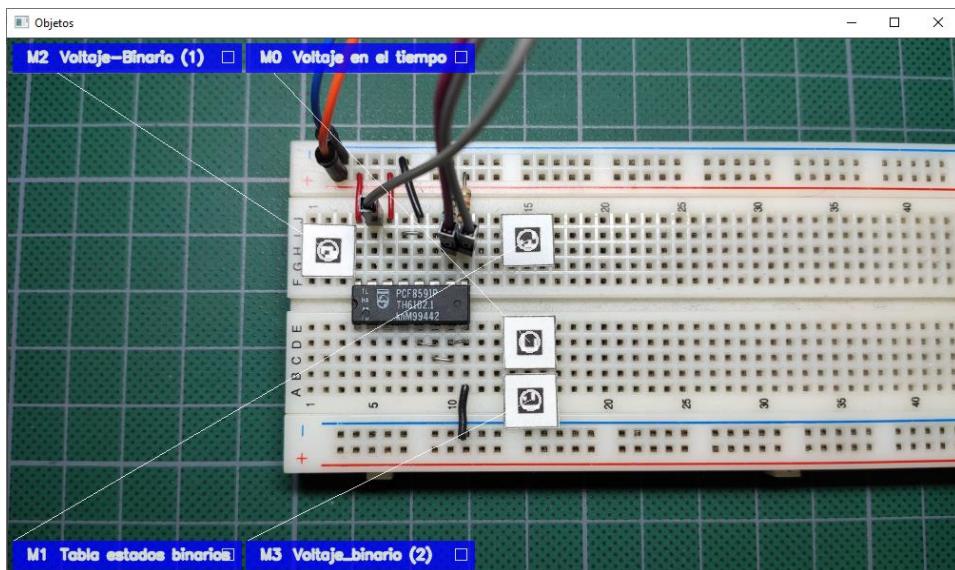


Figura 278. Ventana inicial con la imagen del circuito.

Se muestra la imagen del circuito con los marcadores y sobre de ella se muestran 4 pestanas.

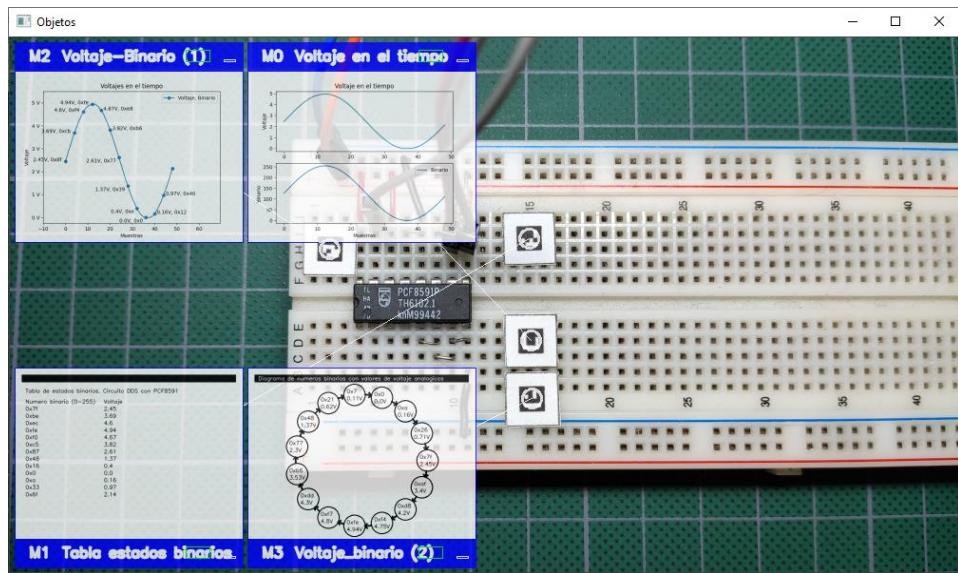


Figura 279. Ventanas desplegadas de todas las gráficas del circuito.

Al abrir cada una de las 4 pestñas, se despliegan las 4 gráficas anteriormente descritas.

7.2.2. Gráfica 0: Comparación entre voltaje y binario 1

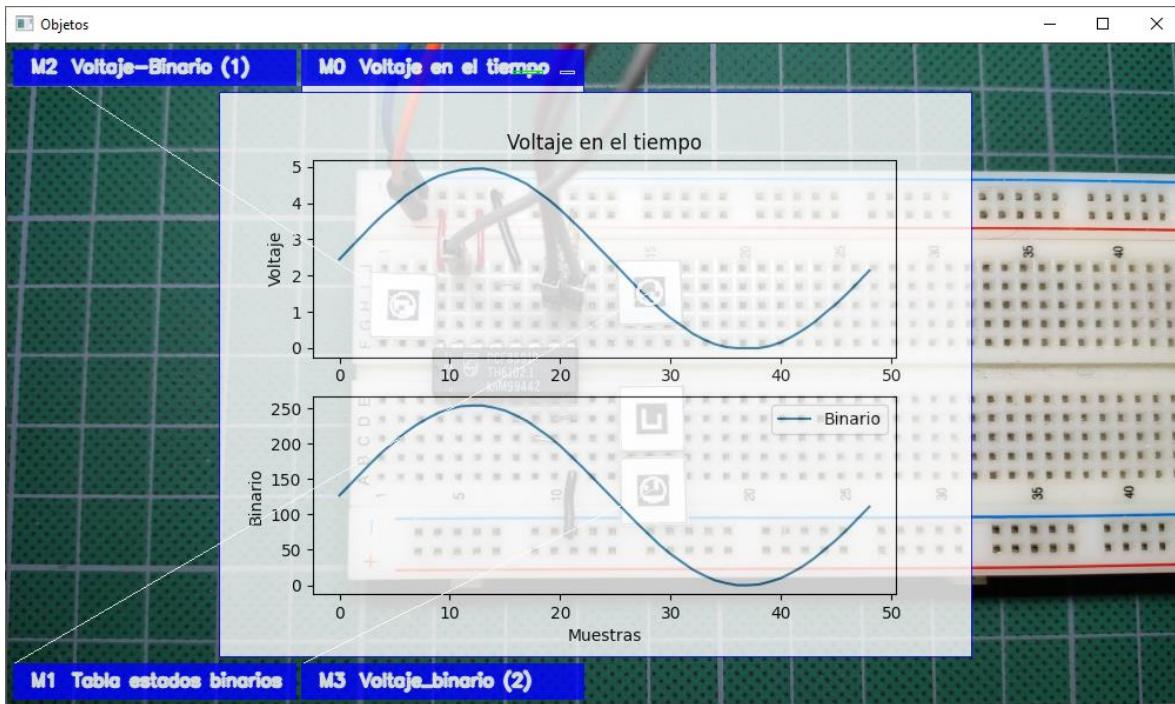


Figura 280. Presentación de la Gráfica 0: Comparación entre voltaje y binario 1.

En esta **gráfica 0** se muestra tanto la curva del **voltaje a la salida** del circuito PCF8591 en azul en la parte superior como la curva de los **valores binarios** enviados a este circuito PCF8591 en azul en la parte inferior.

Se observa que el voltaje a la salida del circuito PCF8591 es directamente proporcional al valor en binario enviado. Esto es que para 5V, se tiene que enviar un valor de 255; para 0V se envía 0 y para 2.5V se envía 127 aproximadamente.

Se observa que la secuencia enviada al circuito es para producir una señal senoidal. Esto indica que también se puede programar el microcontrolador para que produzca otros tipos de señales, como la triangular o cuadrada, por ejemplo.

7.2.3. Gráfica 1: Comparación entre voltaje y binario 2

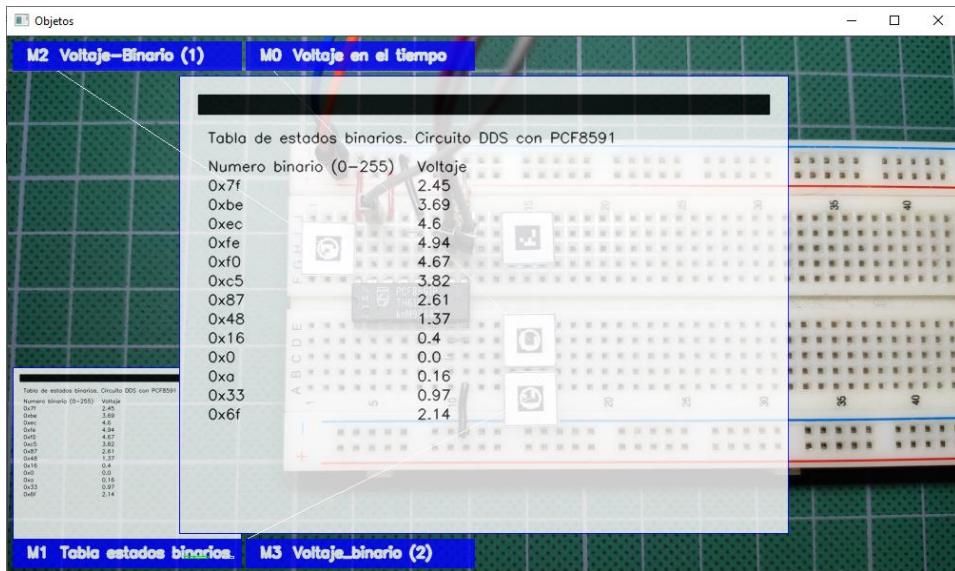


Figura 281. Presentación de la Gráfica 1: Comparación entre voltaje y binario 2

En la **gráfica 1** se presenta una tabla comparativa de algunos **valores binarios** enviados al PCF8591 en formato hexadecimal y el **voltaje a la salida** de este circuito PCF8591. Se observa que un valor de casi 5V es producido con un valor binario de 255 (en hexadecimal 0xff), mientras que con 0V se necesita un valor binario de 0 (en hexadecimal 0x00). Por otro lado, para un valor de alrededor de 2.5V se necesita un valor binario de 128 aproximadamente.

Los valores seleccionados son los siguientes:

Valor binario (en hexadecimal)	Voltaje a la salida (V)
0x7F	2.45
0xBE	3.69
0xEC	4.6
0xFE	4.94
0XF0	4.67
0xC5	3.82
0x87	2.61
0x48	1.37
0x16	0.4
0x00	0
0x0A	0.16
0x33	0.97
0x6F	2.14

Tabla 37. Valores seleccionados de valor binario y voltaje a la salida.

7.2.4. Gráfica 2: Comparación entre voltaje y binario 3

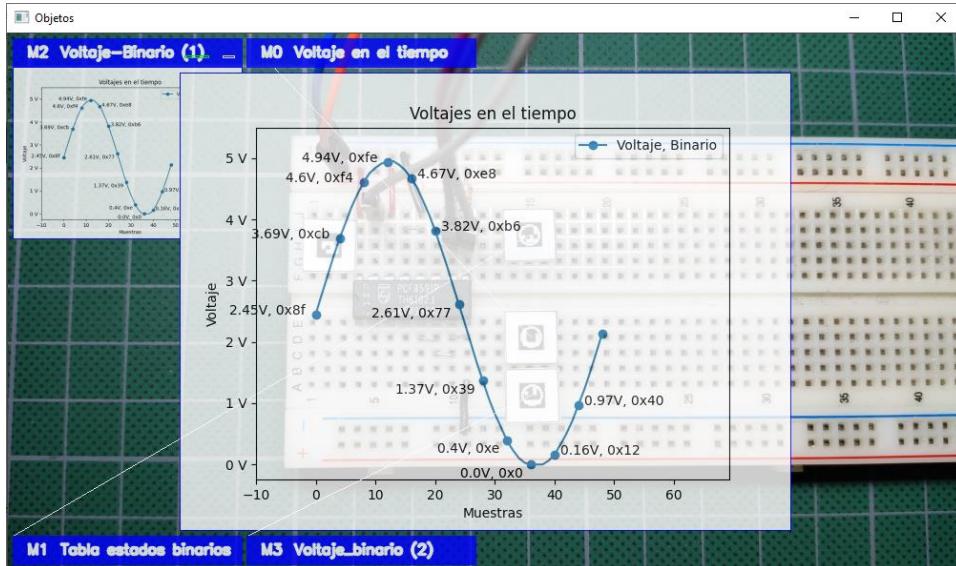


Figura 282. Presentación de la **Gráfica 2: Comparación entre voltaje y binario 3**.

En esta **gráfica 2** también se puede comparar el **voltaje a la salida** del circuito PCF8591 y los **valores binarios** enviados a este circuito. Se muestra en azul la gráfica de **voltaje a la salida** del circuito PCF8591. Esta vez se observa que hay etiquetas sobre algunos puntos de la gráfica, mostrando el valor del voltaje de salida y el valor binario enviado en formato hexadecimal. Los valores seleccionados son los mismos que los mostrados en la tabla 37 de la sección 7.2.3.

7.2.5 Gráfica 3: Comparación entre voltaje y binario 4

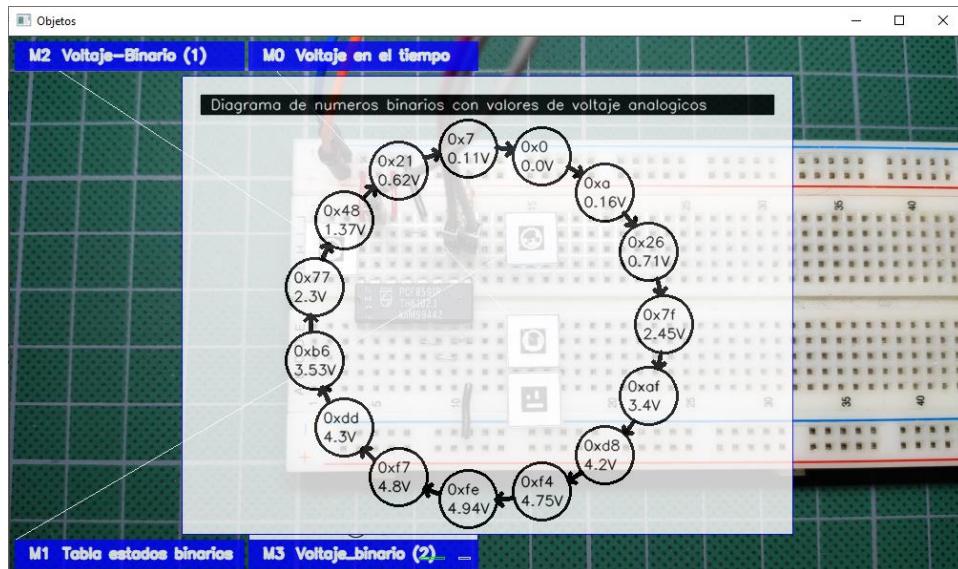


Figura 283. Presentación de la Gráfica 3: Comparación entre voltaje y binario 4

En la **gráfica 3** se presenta un diagrama que permite también la comparación de **voltaje a la salida** del circuito PCF8591 y los **valores en binario** enviados a este circuito. El diagrama tiene 15 círculos, en donde cada uno de estos círculos se muestra tanto el **valor en binario** como el **voltaje a la salida** del circuito PCF8591.

En este diagrama se observa en cada círculo el valor de voltaje y el valor en binario asociado para producir ese voltaje. Estos valores seleccionados son los mismos que los mostrados en la tabla 37 de la sección 7.2.3.

7.3 Amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

7.3.1 Presentación de la imagen del circuito con las gráficas

Para el **circuito Amplificador de una etapa (c)**, se presentan 8 gráficas. Al ejecutar los programas de Arduino y Python, se abre la siguiente ventana:

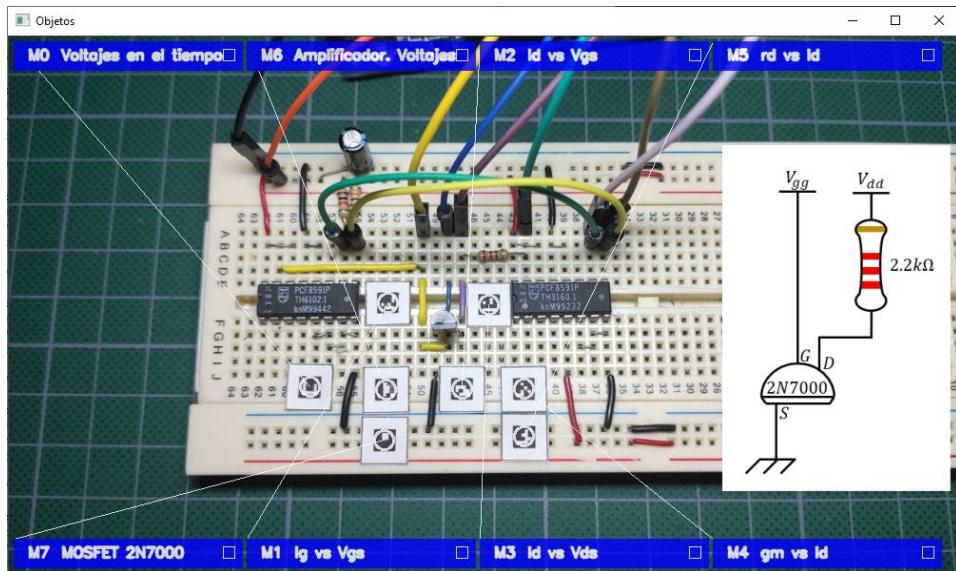


Figura 284. Ventana inicial con la imagen del circuito.

Se muestra la imagen del circuito con los marcadores y sobre de ella se muestran 8 pestanas.

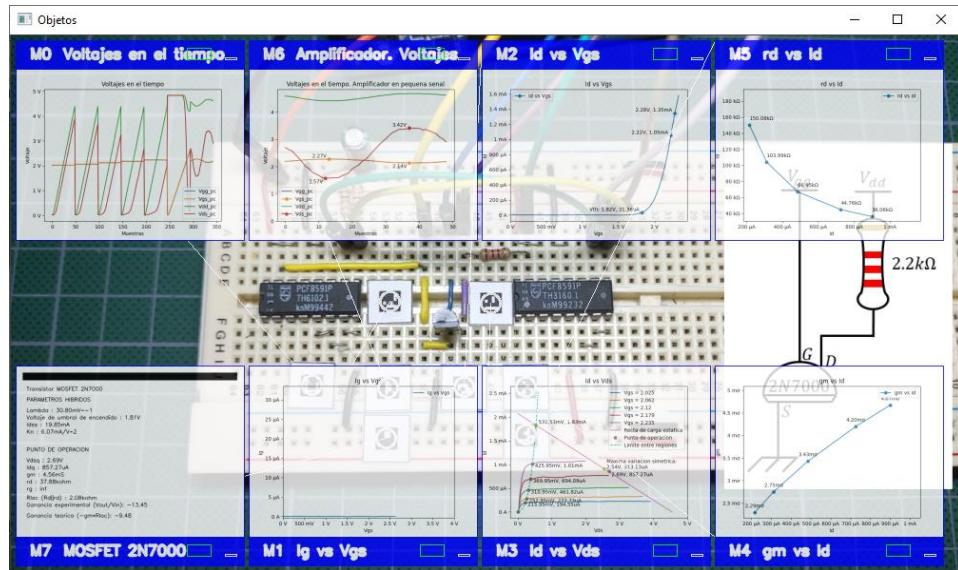


Figura 285. Ventanas desplegadas de todas las gráficas del circuito.

Al abrir cada una de las 8 pestanas, se despliegan las 8 gráficas anteriormente descritas.

7.3.2 Gráfica 0: Voltajes en el tiempo

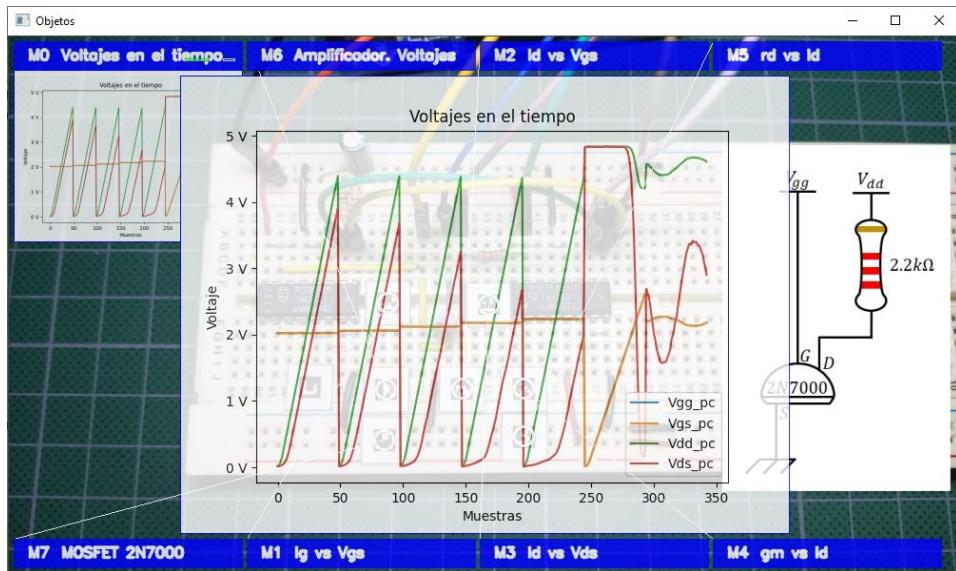


Figura 286. Presentación de la Gráfica 0: Voltajes en el tiempo.

Se observan los **voltajes** V_{gg} (azul), V_{gs} (naranja), V_{dd} (verde) y V_{ds} (rojo). Se tomaron 300 muestras divididas en 7 series de 50 muestras.

Esta **gráfica 0** permite observar el comportamiento de cada voltaje en el tiempo y si se toman correctamente las muestras del circuito.

Se observa para las primeras 250 muestras o primeras 5 series que la forma de onda de este voltaje V_{ds} es similar al voltaje V_{dd} . Con cada incremento del voltaje V_{gs} , el voltaje V_{ds} tarda más en seguir a la forma de onda del voltaje V_{dd} .

Por otro lado, en la 6ta serie de muestras tomadas (250 a 299) se observa que al incrementar V_{gs} desde 0V, hay un momento en el que el voltaje V_{ds} disminuye. También se observa que el voltaje V_{dd} también disminuye, aunque no debería por ser un voltaje de alimentación.

Para la 7ma serie de muestras tomadas (300 a 349), el transistor funciona como amplificador en pequeña señal. Se observa que el voltaje de entrada V_{gs} tiene forma de una señal senoidal con una amplitud pequeña. En la salida, que es voltaje V_{ds} , se observa que también tiene forma de señal senoidal, aunque está invertida y tiene una amplitud mayor con respecto al voltaje de entrada V_{gs} . Por otro lado, se observan variaciones en el voltaje V_{dd} , aunque no debería de suceder por ser un voltaje de alimentación.

7.3.3 Gráfica 1: Ig vs Vgs

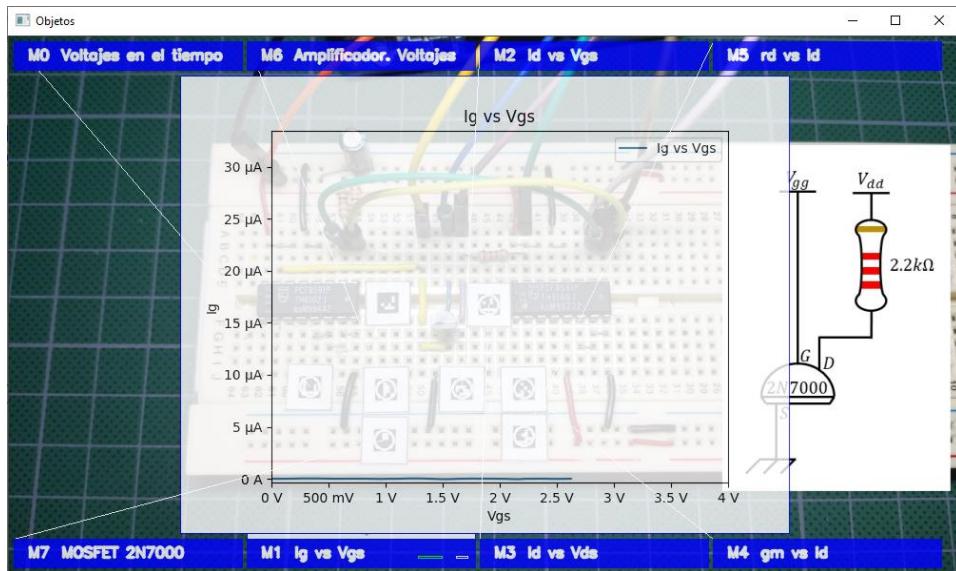


Figura 287. Presentación de la **Gráfica 1: Ig vs Vgs**.

En esta **gráfica 1** se observa en azul la **curva característica de entrada I_g vs V_{gs}** del transistor MOSFET de enriquecimiento. Puesto que es un transistor controlado por voltaje y no por corriente, **la gráfica 1** muestra que se requiere con un valor muy pequeño de corriente I_g para que funcione el transistor MOSFET.

7.3.4 Gráfica 2: Id vs Vgs

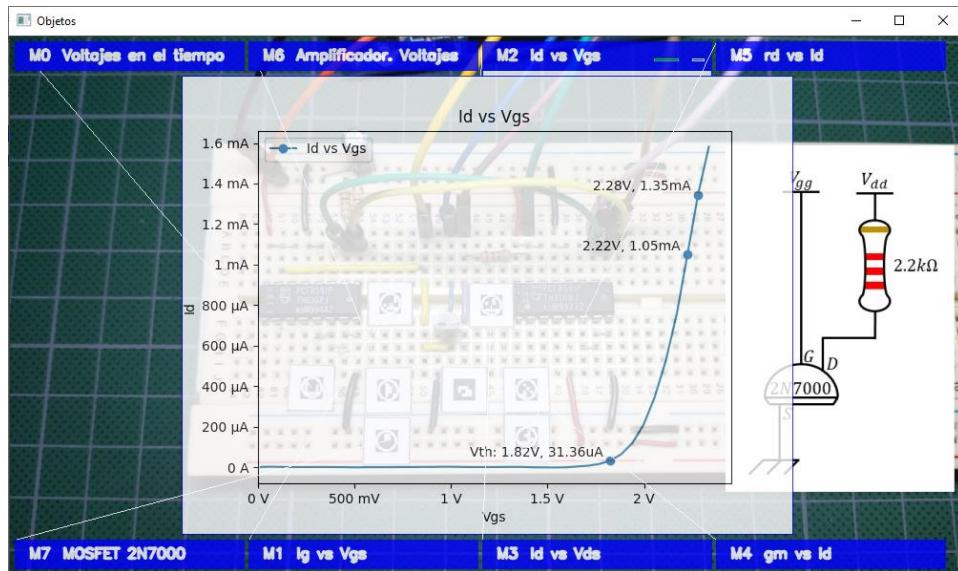


Figura 288. Presentación de la Gráfica 2: Id vs Vgs.

En esta **gráfica 2** se observa en azul la **curva característica de transferencia I_d vs V_{gs}** del transistor MOSFET de enriquecimiento. Se observa que a partir de un cierto valor de voltaje V_{gs} empieza a circular una corriente I_d . Se muestra el voltaje de umbral de encendido V_{th} , así como dos muestras las cuales se utilizaron para obtener algunos parámetros del transistor, como K_n , V_{th} e I_{dss} .

7.3.5 Gráfica 3: Id vs Vds

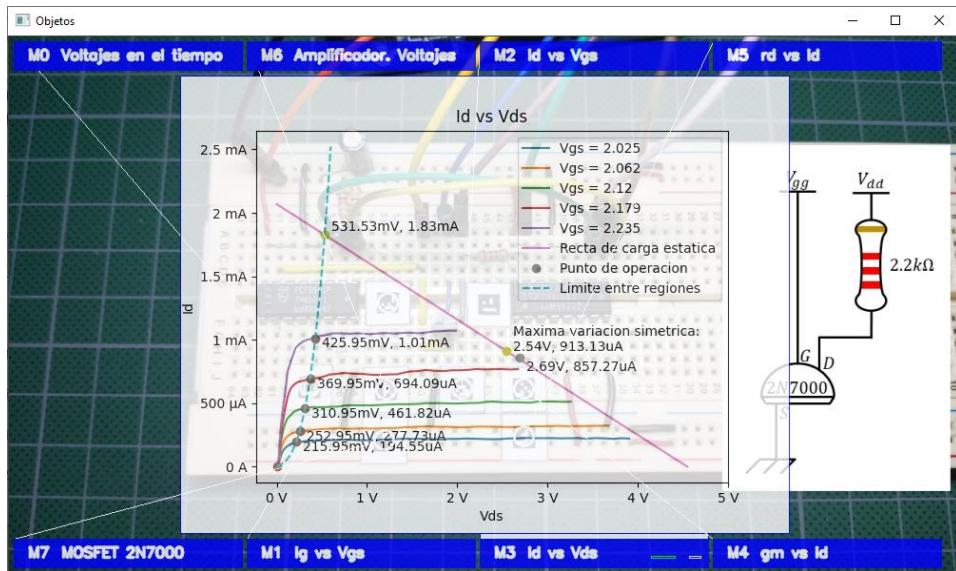


Figura 289. Presentación de la Gráfica 3: Id vs Vds.

En esta **gráfica 3** se observa un conjunto de **5 curvas características de salida I_d vs V_{ds}** del transistor MOSFET en azul, naranja, verde, rojo y morado. Cada curva se obtuvo al fijar un voltaje V_{gs} (mostrado en la leyenda de la gráfica) y se observa que cada curva tiene una corriente I_d diferente más o menos constante.

Además, se observa la **curva que delimita las regiones de operación óhmica y de saturación** del transistor MOSFET. También se observa el **voltaje de saturación $V_{ds(sat)}$** para cada una de las cinco curvas en donde cambia de región de operación.

Se observa que está la **recta de carga estática**, el **punto de máxima variación simétrica** y el **punto de operación** cuando el transistor se comporta como amplificador.

Se observa que tanto el punto de máxima variación simétrica y el punto de operación están muy cerca uno del otro.

7.3.6 Gráfica 4: gm vs Id

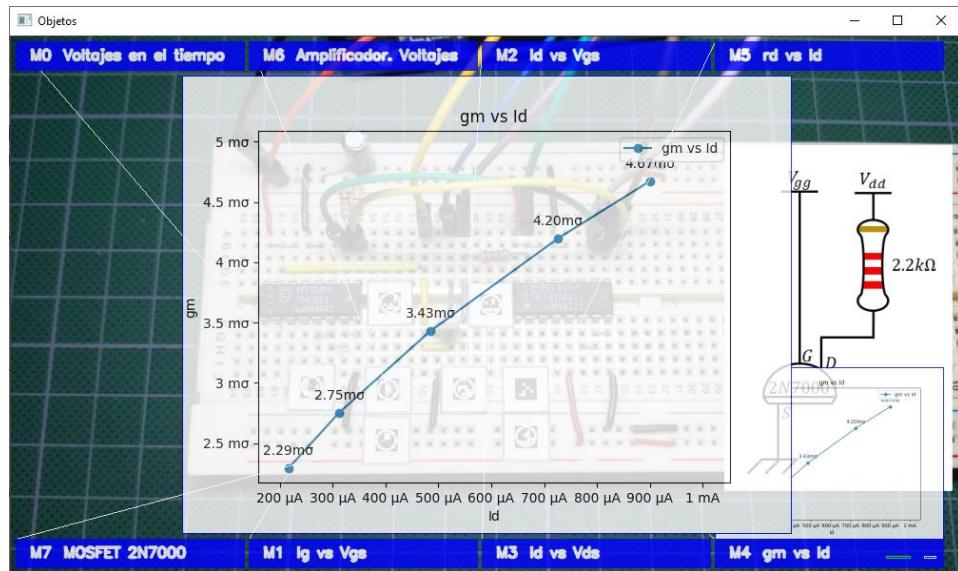


Figura 290. Presentación de la Gráfica 4: gm vs Id.

En esta **gráfica 4** se observa en azul el comportamiento del parámetro de transconductancia g_m . Se calcularon 5 valores de g_m correspondientes a cada una de las 5 curvas I_d vs V_{ds} presentadas en la figura 290.

7.3.7 Gráfica 5: r_d vs I_d

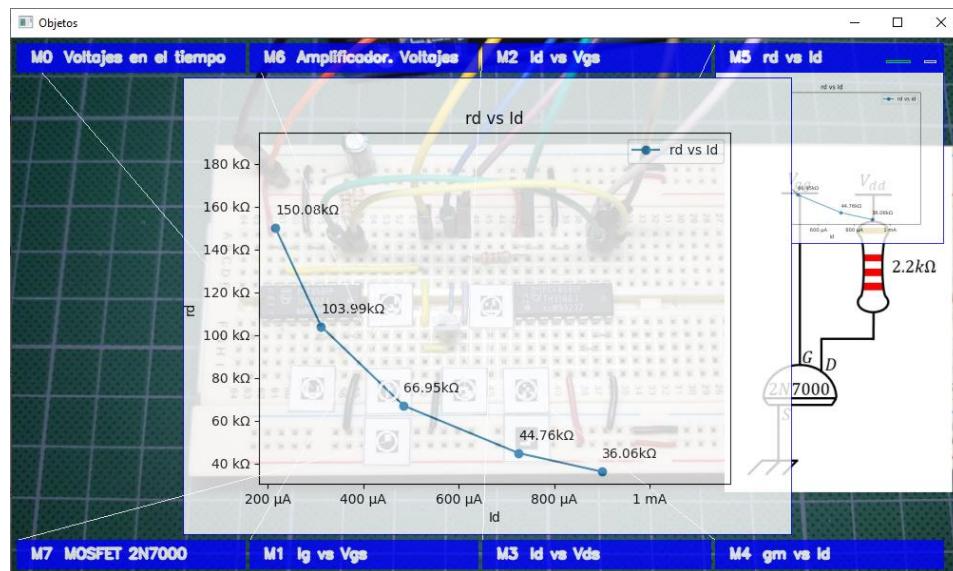


Figura 291. Presentación de la Gráfica 5: r_d vs I_d .

En esta **gráfica 5** se observa en azul el comportamiento de la **resistencia r_d** del modelo híbrido pi. Se calcularon 5 valores de r_d correspondientes a cada una de las 5 curvas I_d vs V_{ds} presentadas en la figura 275.

7.3.8 Gráfica 6: Voltajes del amplificador en fuente común

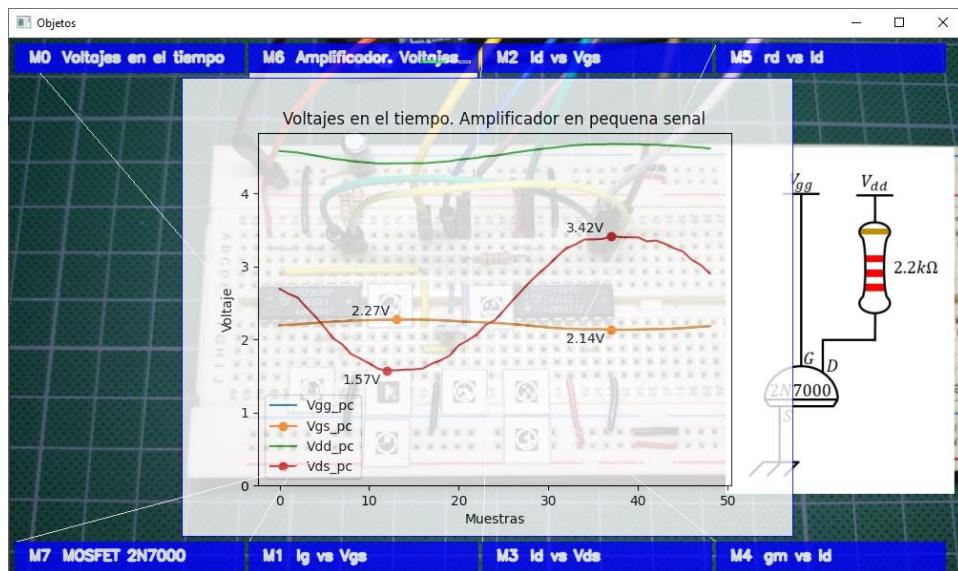


Figura 292. Presentación de la **Gráfica 6: Voltajes del amplificador en fuente común**.

En esta **gráfica 6** se observa el comportamiento del **transistor como amplificador en fuente común**. El **voltaje V_{gs}** (en amarillo) es la **entrada** y tiene una forma de onda senoidal con una amplitud de voltaje pequeña (alrededor de 150 mV). El **voltaje V_{ds}** (en rojo) es la **salida** y se observa la misma forma de onda senoidal, pero con una amplitud de voltaje mayor y está invertida con respecto a la forma de onda del voltaje V_{gs} de entrada.

Además, se observa que están marcados los máximos y mínimos de los voltajes de entrada V_{gs} y salida V_{ds} .

7.3.9 Gráfica 7: Parámetros híbridos y punto de operación

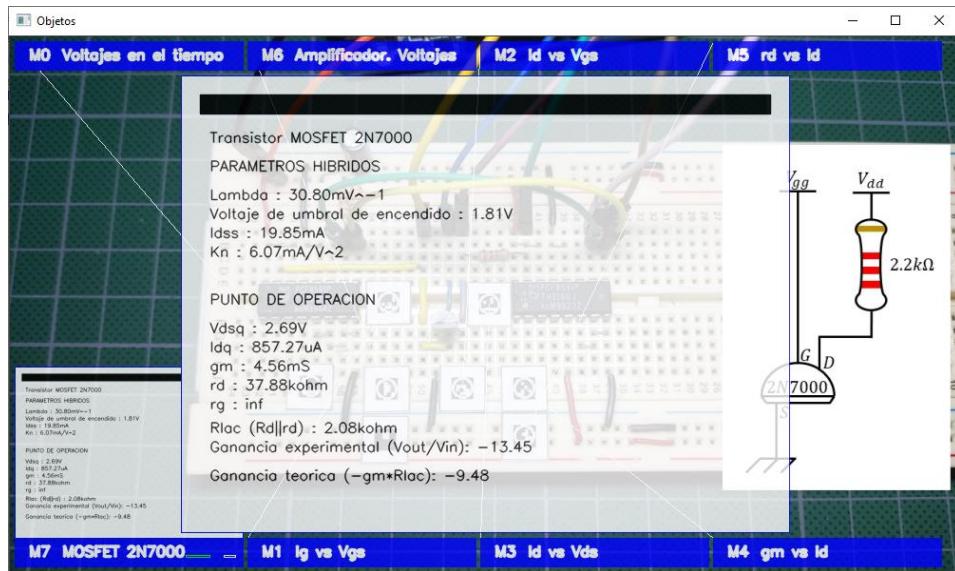


Figura 293. Presentación de la **Gráfica 7: Parámetros híbridos y punto de operación**.

En esta **gráfica 7** se muestra un resumen de los **parámetros híbridos** calculados del transistor. Estos parámetros incluyen:

- **Lambda**.
- El **voltaje de umbral de encendido** V_{th} del transistor MOSFET.
- Los parámetros I_{dss} , K_n y la **resistencia** r_d del modelo híbrido pi.

Los valores de los parámetros híbridos son los siguientes:

Parámetro	Valor resultante
λ	$30.8 \frac{1}{mV}$
V_{th}	1.81 V
I_{dss}	19.85 mA
K_n	$6.07 \frac{mA}{V^2}$
r_g	∞

Tabla 38. Resultados de los parámetros híbridos del transistor MOSFET 2N7000.

También se muestran los **parámetros** del **punto de operación** cuando el transistor MOSFET funciona como **amplificador**. Los parámetros asociados al punto de operación incluyen:

- El **voltaje entre drenaje y fuente del punto de operación** V_{dsq} .
- La **corriente de drenaje en el punto de operación** I_{dq} .
- Las **resistencias** r_g y r_d del modelo híbrido pi.
- La **resistencia equivalente** R_{lac} de salida utilizando la resistencia del circuito R_d en paralelo con la resistencia híbrida r_d .
- La **ganancia del amplificador** calculada de dos maneras, tanto **teórica** como **experimental**.

Los valores de los parámetros en el punto de operación son los siguientes:

Parámetro	Valor resultante
V_{dsq}	2.69 V
I_{dq}	857.27 μA
g_m	4.56 mS
r_d	37.88 k Ω
r_g	∞
R_{lac}	2.08 k Ω
Ganancia experimental	-13.45
Ganancia teórica	-9.48

Tabla 39. Resultados de los parámetros en el punto de operación del transistor MOSFET 2N7000.

7.4. Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

7.4.1 Presentación de la imagen del circuito con las gráficas

Para el **circuito secuencial (d)**, se presentan 5 gráficas. Al ejecutar los programas de Arduino y Python, se abre la siguiente ventana:

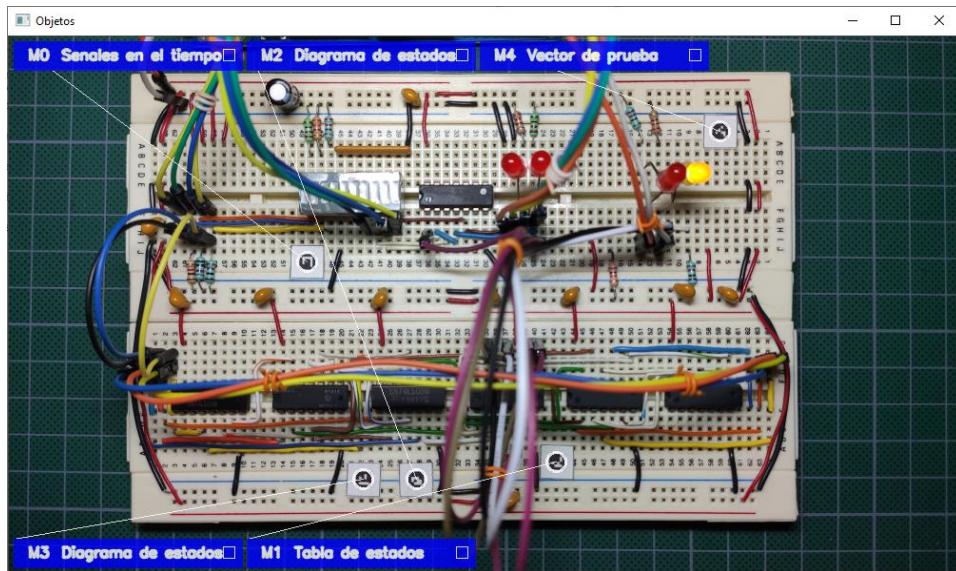


Figura 294. Ventana inicial con la imagen del circuito (circuito secuencial implementado por hardware).

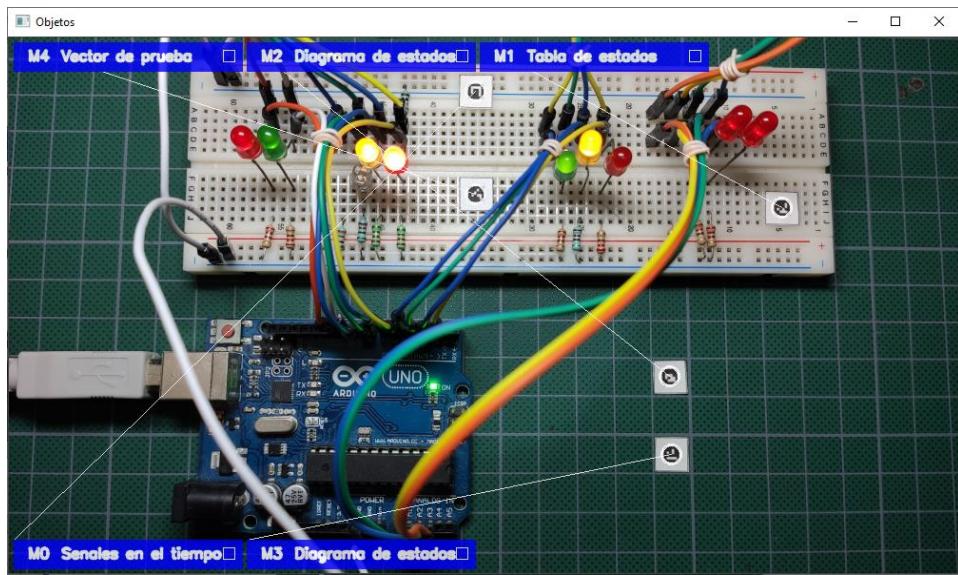


Figura 295. Ventana inicial con la imagen del circuito (circuito secuencial implementado por software).

Se muestra la imagen del circuito con flip flops (implementado por hardware) en la figura 294 e implementado con un microcontrolador (implementado por software) en la figura 295. Cada una de las imágenes contiene 5 marcadores y sobre de ella se muestran 5 pestanas.

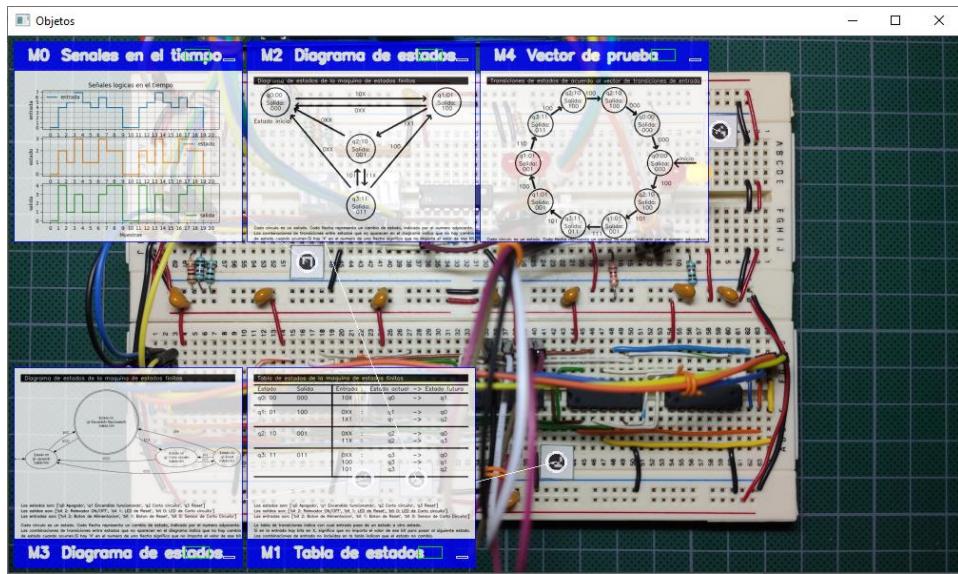


Figura 296. Ventanas desplegadas de todas las gráficas del circuito (circuito secuencial implementado por hardware).

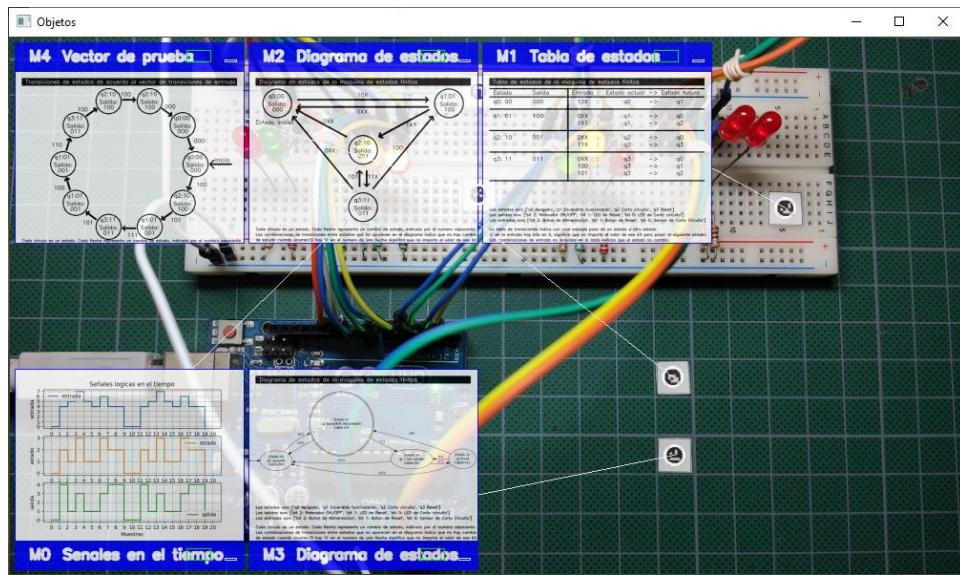


Figura 297. Ventanas desplegadas de todas las gráficas del circuito (circuito secuencial implementado por software).

Al abrir cada una de las 5 pestañas, se despliegan las 5 gráficas anteriormente descritas.

7.4.2 Gráfica 0: Señales digitales binarias en el tiempo

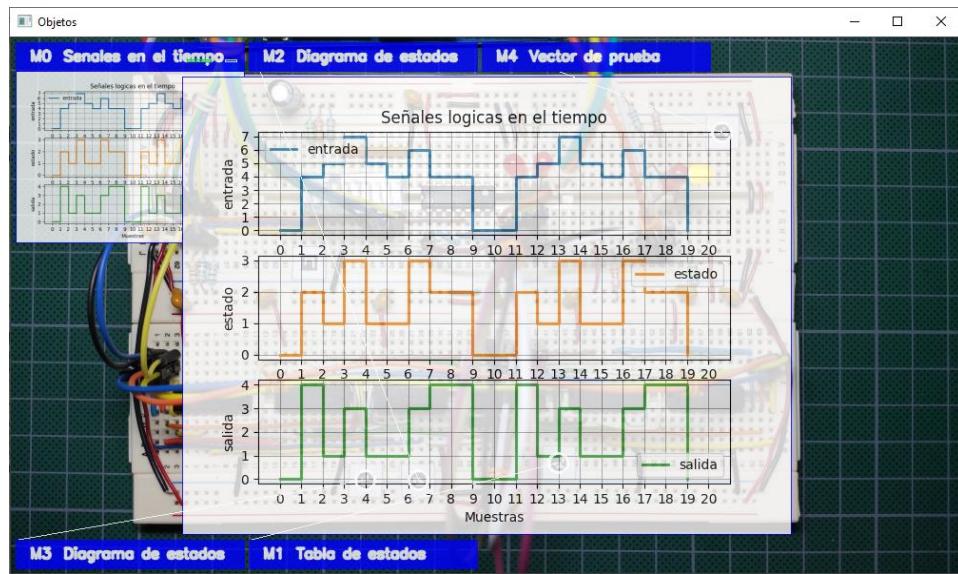


Figura 298. Presentación de la **Gráfica 0: Señales digitales en el tiempo** (circuito secuencial implementado por hardware).

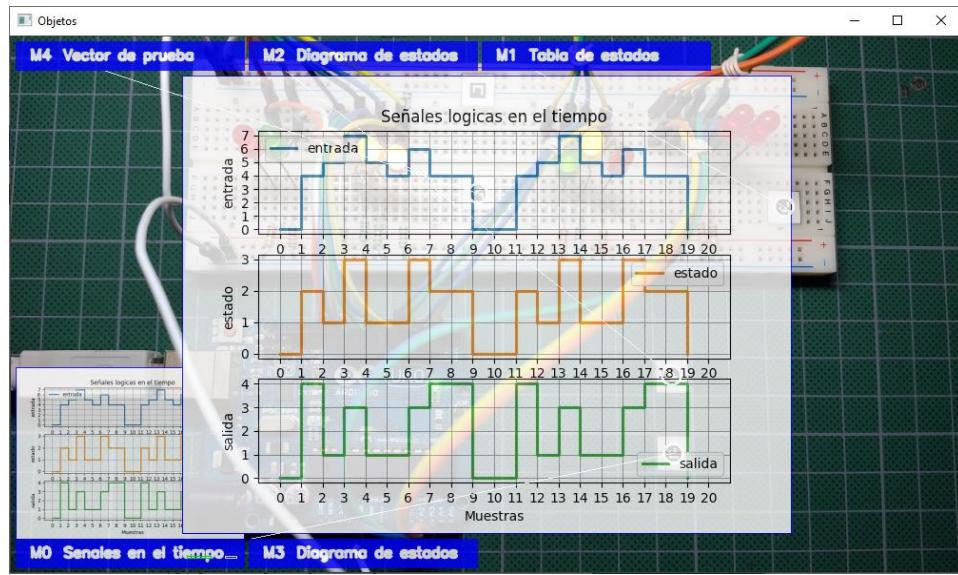


Figura 299. Presentación de la **Gráfica 0: Señales digitales en el tiempo** (circuito secuencial implementado por software).

En esta **gráfica 0** se pueden comparar tres señales, que son la **señal de entrada** (en azul en la parte superior) de la máquina de estados, la **señal de estado** (en amarillo en la parte de en medio) y la **señal de salida** (en verde en la parte inferior). Cada conjunto de los bits de entrada, estado y salida se visualiza como un número entero.

La máquina de estado pasa por los siguientes estados, mostrando la salida y la entrada correspondiente (en binario):

Entrada	Estado	Salida
000	00	000
100	10	100
101	01	001
110	11	011
101	01	001
100	01	001
110	11	011
100	10	100
100	10	100
000	00	000

Tabla 40. Estados por los que pasa la máquina de estados finitos, mostrando entrada y salida correspondiente a cada estado.

Esta secuencia de cambios de estados mostrada en la tabla 40 se repite dos veces en la gráfica. Se observa que la salida cambia junto con el estado. Estos cambios de estado son dados por la señal entrada.

7.4.3 Gráfica 1: Tabla de estados binarios

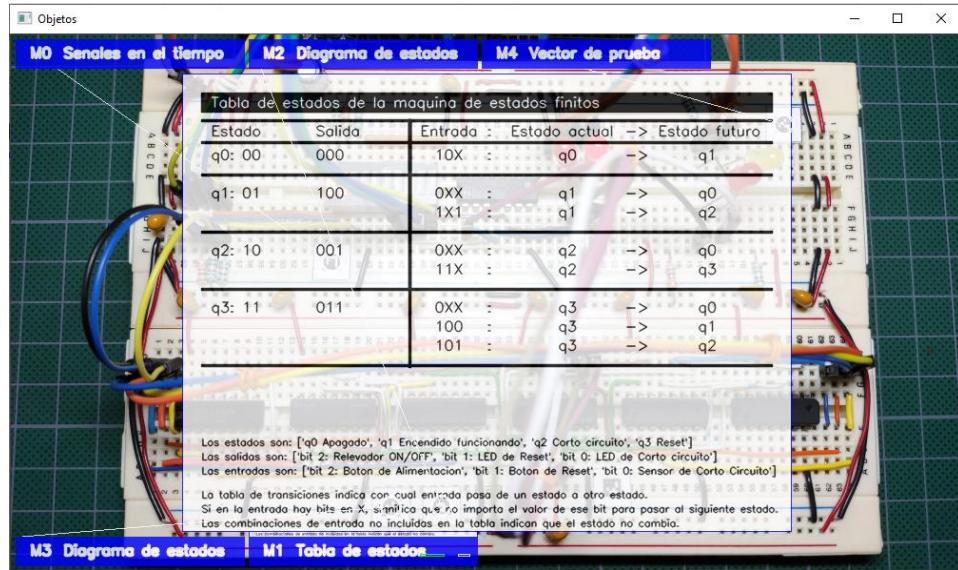


Figura 300. Presentación de la **Gráfica 1: Tabla de estados binarios** (circuito secuencial implementado por hardware).

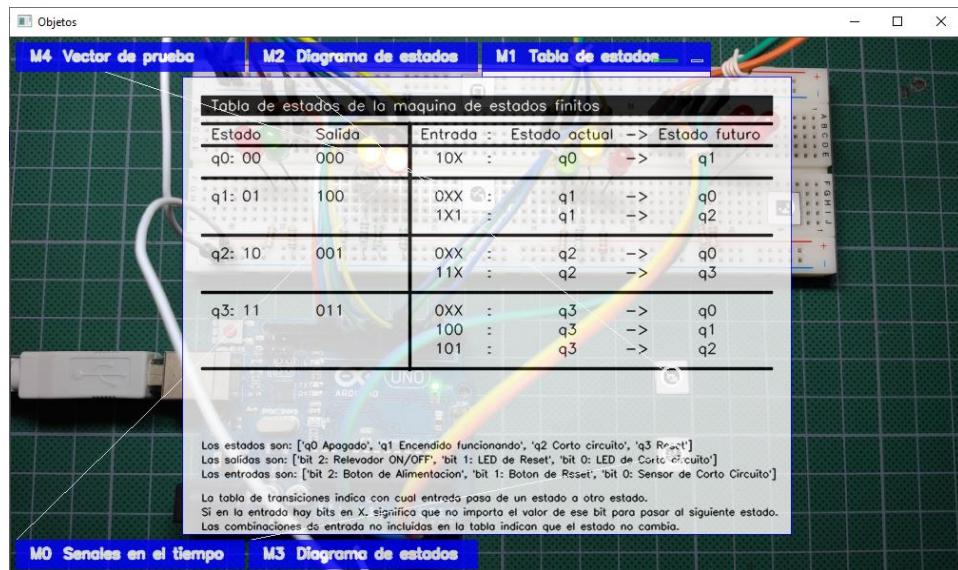


Figura 301. Presentación de la **Gráfica 1: Tabla de estados binarios** (circuito secuencial implementado por software).

Esta **gráfica 1** se muestra una tabla en donde se observa el **comportamiento de la máquina de estados**, mostrando para cada **estado** su **salida** correspondiente y las **transiciones** que tiene **hacia otros estados** con la **entrada** correspondiente. En la parte inferior de la tabla se muestra una leyenda con información adicional para entender la tabla.

Hay 4 estados, los cuales el estado 0 tiene 1 posible transición hacia el estado 1. El estado 1 tiene dos transiciones hacia los estados 0 y 2. El estado 2 tiene dos transiciones hacia el estado 0 y 3. El estado 3 tiene tres transiciones hacia los estados 0,1 y 2

Se observa en la parte inferior que hay notas que amplían la explicación de la tabla.

7.4.4 Gráfica 2: Diagrama de estados 1

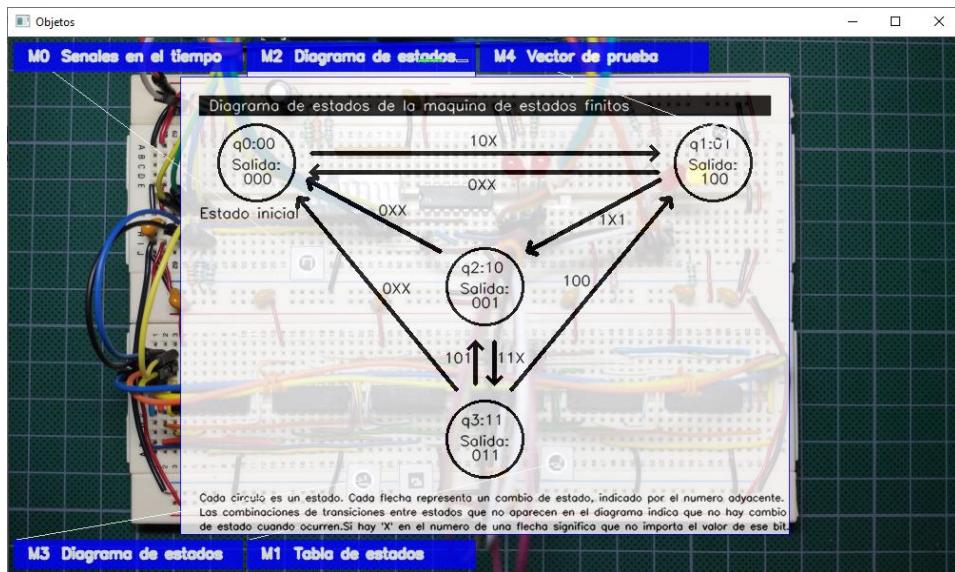


Figura 302. Presentación de la **Gráfica 2: Diagrama de estados 1** (circuito secuencial implementado por hardware).

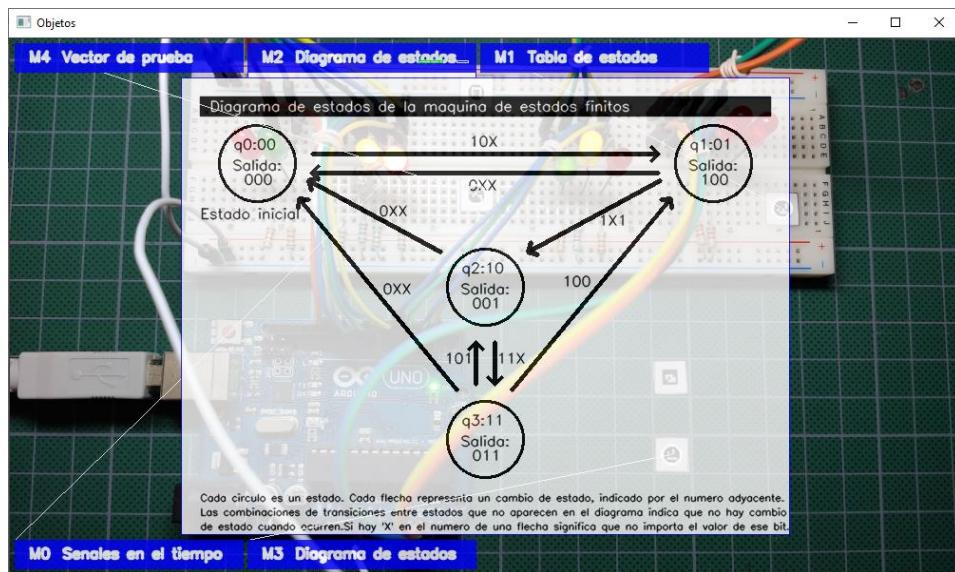


Figura 303. Presentación de la **Gráfica 2: Diagrama de estados 1** (circuito secuencial implementado por software).

En la **gráfica 2** se observa este **diagrama de estados** que también muestra el **comportamiento de la máquina de estados**. Cada **círculo** representa un **estado** diferente en donde muestra tanto el **número de estado** y su **salida**. Las **flechas** entre estados son las **transiciones** del estado actual hacia otros estados. El **número sobre la flecha** indica la **entrada** con la cuál entra la máquina de estados finitos **cambia de estado**. Este diagrama es equivalente a la tabla anterior mostrada en las figuras 302 y 303.

7.4.5 Gráfica 3: Diagrama de estados 2

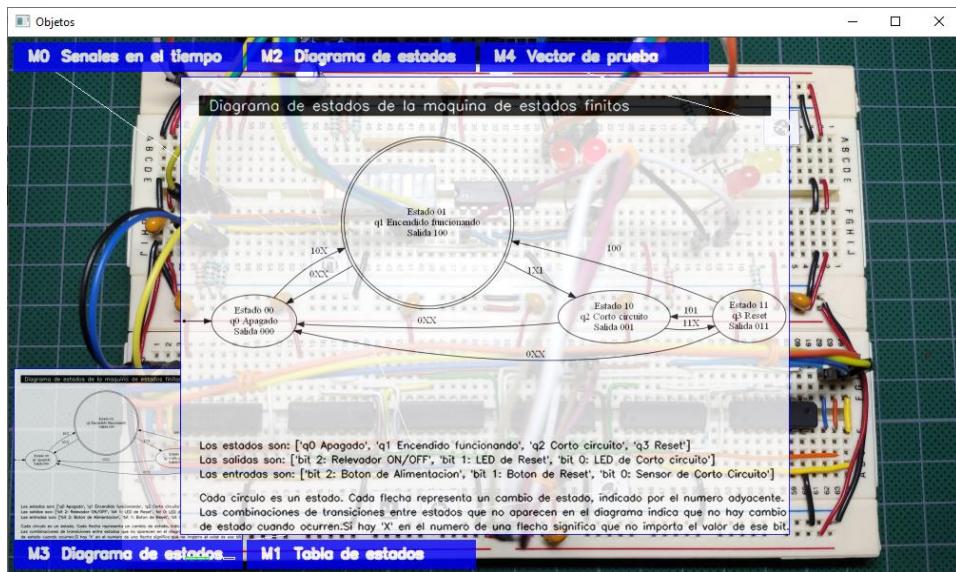


Figura 304. Presentación de la **Gráfica 3: Diagrama de estados 2** (circuito secuencial implementado por hardware).

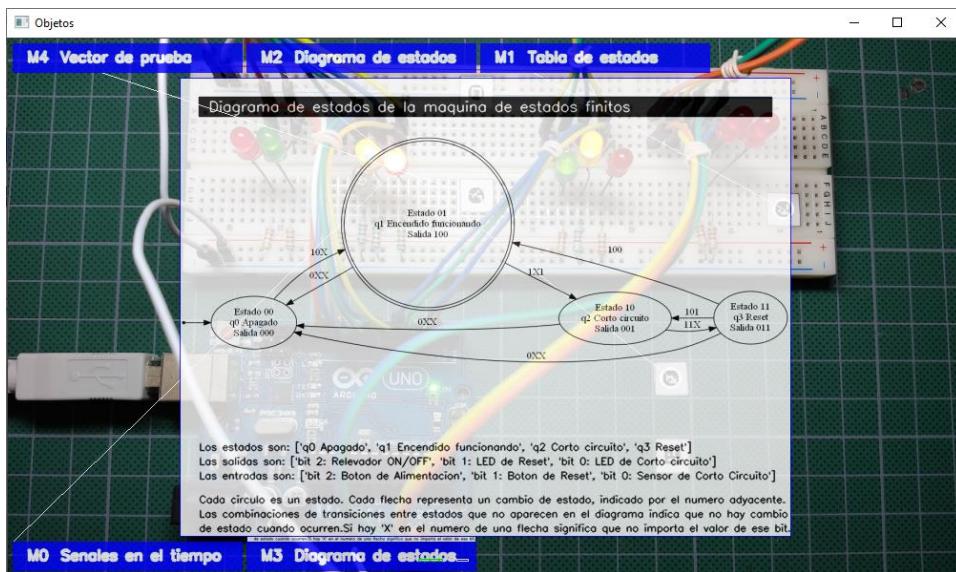


Figura 305. Presentación de la **Gráfica 3: Diagrama de estados 2** (circuito secuencial implementado por software).

La **gráfica 3** presenta el mismo **diagrama de estados** de las figuras 304 y 305 de manera diferente utilizando el paquete graphviz. Es otra forma de representar el comportamiento de la máquina de estados finitos.

7.4.6 Gráfica 4: Diagrama de transiciones de la máquina de estados finitos

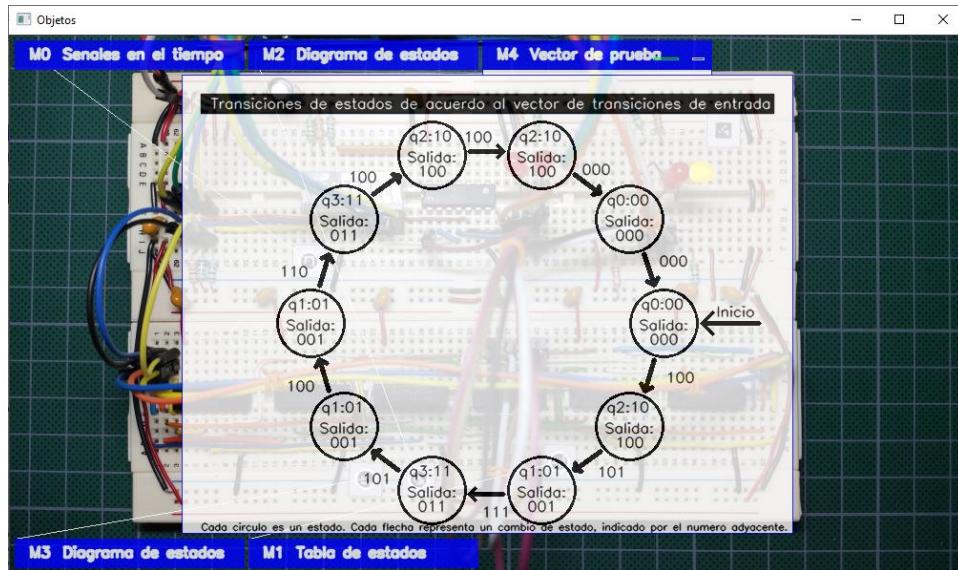


Figura 306. Presentación de la **Gráfica 4: Diagrama de transiciones de la máquina de estados finitos** (circuito secuencial implementado por hardware).

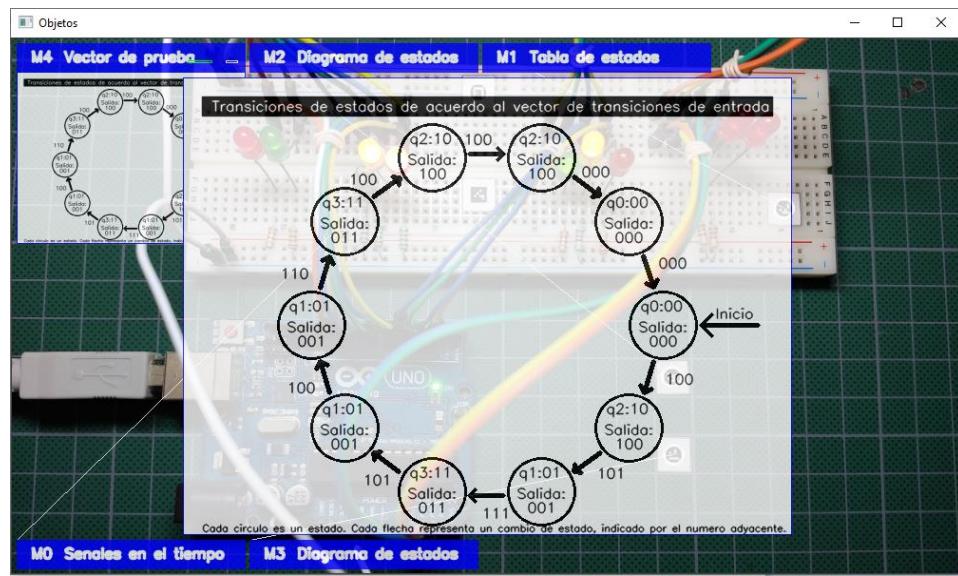


Figura 307. Presentación de la **Gráfica 4: Diagrama de transiciones de la máquina de estados finitos** (circuito secuencial implementado por software).

En la **gráfica 4** se presenta un **diagrama** en donde se muestra el **vector de prueba** aplicado a la máquina de estados finitos. En cada círculo se muestra el **estado actual** de la máquina de estados finitos y su **salida**. Cada **flecha** con su respectivo **número** significa la **entrada** con la que la máquina **cambia al siguiente estado**.

Este **diagrama** de transiciones presentado en las figuras 306 y 307 contiene las mismas transiciones de estados que las mostradas en la tabla 40 de la sección 7.4.2.

Se observa que el vector de prueba hace que la máquina de estados inicie y termine en el mismo estado.

8. Análisis y discusión de resultados

En esta sección, los resultados obtenidos en la sección **7 Resultados**, se comparan con las hojas de datos, gráficas y parámetros que se presentan, para cada circuito, en la sección **5 Marco Teórico**.

Se compararán los resultados de los 4 circuitos desarrollados:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

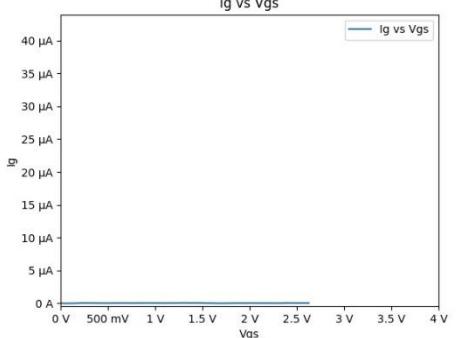
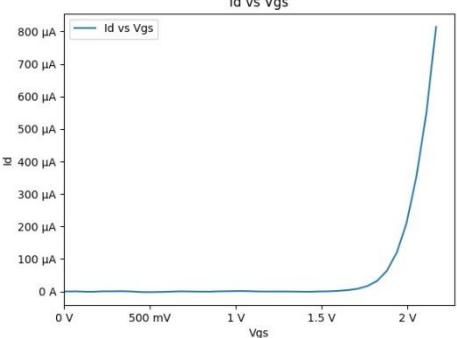
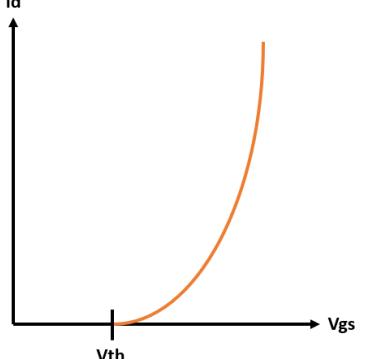
8.1. Trazador de Curvas (a)

Ubicación de los circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

8.1.1 Comparación de las gráficas resultantes de las curvas características de entrada, transferencia y salida.

Comparando las gráficas de las **curvas características de entrada, transferencia y de salida**:

Tipo de gráfica	Gráfica resultante	Gráfica del marco teórico																				
Curva característica de entrada	 <p>Ig vs Vgs</p> <p>Y-axis: Ig (0 A to 40 μA) X-axis: Vgs (0 V to 4 V)</p> <table border="1"><caption>Data points for Ig vs Vgs</caption><thead><tr><th>Vgs (V)</th><th>Ig (μA)</th></tr></thead><tbody><tr><td>0</td><td>0</td></tr><tr><td>0.5</td><td>10</td></tr><tr><td>1.0</td><td>20</td></tr><tr><td>1.5</td><td>30</td></tr><tr><td>2.0</td><td>40</td></tr><tr><td>2.5</td><td>40</td></tr><tr><td>3.0</td><td>40</td></tr><tr><td>3.5</td><td>40</td></tr><tr><td>4.0</td><td>40</td></tr></tbody></table>	Vgs (V)	Ig (μA)	0	0	0.5	10	1.0	20	1.5	30	2.0	40	2.5	40	3.0	40	3.5	40	4.0	40	 <p>Ig</p> <p>Y-axis: Ig (0 A to 40 μA) X-axis: Vgs (0 V to 4 V)</p> <p>Threshold voltage: Vth</p>
Vgs (V)	Ig (μA)																					
0	0																					
0.5	10																					
1.0	20																					
1.5	30																					
2.0	40																					
2.5	40																					
3.0	40																					
3.5	40																					
4.0	40																					
Curva característica de transferencia	 <p>Id vs Vgs</p> <p>Y-axis: Id (0 A to 800 μA) X-axis: Vgs (0 V to 2 V)</p> <table border="1"><caption>Data points for Id vs Vgs</caption><thead><tr><th>Vgs (V)</th><th>Id (μA)</th></tr></thead><tbody><tr><td>0</td><td>0</td></tr><tr><td>0.5</td><td>0</td></tr><tr><td>1.0</td><td>0</td></tr><tr><td>1.5</td><td>0</td></tr><tr><td>1.8</td><td>100</td></tr><tr><td>2.0</td><td>800</td></tr></tbody></table>	Vgs (V)	Id (μA)	0	0	0.5	0	1.0	0	1.5	0	1.8	100	2.0	800	 <p>Id</p> <p>Y-axis: Id (0 A to 800 μA) X-axis: Vgs (0 V to 2 V)</p> <p>Threshold voltage: Vth</p>						
Vgs (V)	Id (μA)																					
0	0																					
0.5	0																					
1.0	0																					
1.5	0																					
1.8	100																					
2.0	800																					

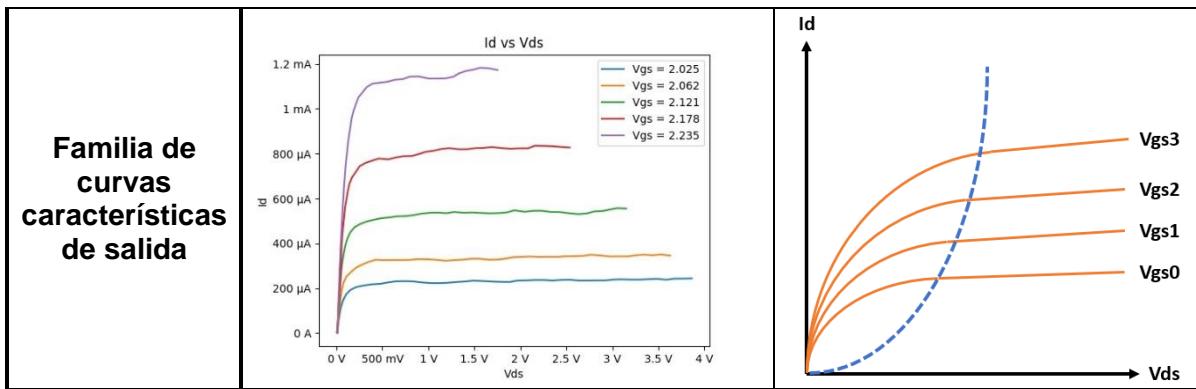


Tabla 41. Tabla comparativa de las gráficas resultantes.

Se observa que las formas de onda de las gráficas resultantes mostradas en la tabla 41 en la columna “gráfica resultante” son muy parecidas a las que se esperaban, mostradas en la tabla 41 en la columna “gráfica del marco teórico”.

En el caso de la **curva característica de entrada**, fue de esperar que la corriente de compuerta I_g fuera muy cercana a 0A.

En el caso de la **curva característica de transferencia**, se observa en la gráfica resultante que en algún valor del voltaje V_{gs} alrededor de 1.8V, la corriente I_d es diferente de 0A, lo cual coincide con la gráfica esperada mostrada en el marco teórico.

En el caso de la **familia de curvas características de salida**, se observa en la gráfica resultante que para cada voltaje V_{gs} elegido, se forma una curva I_d vs V_{ds} diferente. Este comportamiento es el esperado al comparar la gráfica resultante con la gráfica mostrada en el marco teórico. Sin embargo, en esta gráfica se observa un poco de ruido en cada una de las curvas I_d vs V_{ds} , a pesar de que se utiliza el filtro circular sobre las muestras tomadas. Este ruido puede ser producido por las fuentes de alimentación utilizadas, los cables usados y el protoboard en donde se armó el circuito.

Conforme el voltaje V_{gs} aumenta, la curva I_d vs V_{ds} es más corta. Es decir, cada vez es posible graficarla para un menor rango de voltaje V_{ds} . Se debe a la limitación de corriente del circuito PCF8591. Cuando se le exige una mayor corriente a su salida analógica, el voltaje se cae.

8.1.2 Comparación de las gráficas resultantes para los parámetros g_m y r_d

Comparando los resultados de los parámetros g_m y r_d :

Tipo de gráfica	Gráfica resultante	Ecuación de la gráfica en el marco teórico												
Curva del comportamiento del parámetro g_m	<table border="1"> <caption>Data points for gm vs Id graph</caption> <thead> <tr> <th>Id (mA)</th> <th>gm (mS)</th> </tr> </thead> <tbody> <tr><td>0.2</td><td>2.67</td></tr> <tr><td>0.4</td><td>3.20</td></tr> <tr><td>0.6</td><td>4.07</td></tr> <tr><td>0.8</td><td>4.91</td></tr> <tr><td>1.0</td><td>5.33</td></tr> </tbody> </table>	Id (mA)	gm (mS)	0.2	2.67	0.4	3.20	0.6	4.07	0.8	4.91	1.0	5.33	$g_m = 2\sqrt{K_n I_{dq}}$
Id (mA)	gm (mS)													
0.2	2.67													
0.4	3.20													
0.6	4.07													
0.8	4.91													
1.0	5.33													
Curva del comportamiento del parámetro r_d	<table border="1"> <caption>Data points for rd vs Id graph</caption> <thead> <tr> <th>Id (mA)</th> <th>rd (kΩ)</th> </tr> </thead> <tbody> <tr><td>0.2</td><td>132.14</td></tr> <tr><td>0.4</td><td>91.54</td></tr> <tr><td>0.6</td><td>56.60</td></tr> <tr><td>0.8</td><td>38.98</td></tr> <tr><td>1.0</td><td>33.03</td></tr> </tbody> </table>	Id (mA)	rd (kΩ)	0.2	132.14	0.4	91.54	0.6	56.60	0.8	38.98	1.0	33.03	$r_d = \frac{1}{\lambda I_{dq}}$
Id (mA)	rd (kΩ)													
0.2	132.14													
0.4	91.54													
0.6	56.60													
0.8	38.98													
1.0	33.03													

Tabla 42. Tabla comparativa de las gráficas de los parámetros g_m y r_d

En la gráfica del **parámetro g_m** se observa que la ecuación $g_m = 2\sqrt{K_n I_{dq}}$ es la de una parábola horizontal que abre hacia la derecha, con variable independiente I_{dq} y variable dependiente g_m . La gráfica resultante en la tabla 42 muestra una parte de esta parábola horizontal, por lo que es el comportamiento esperado para este parámetro g_m .

Sin embargo, al comparar los valores resultantes de g_m con la hoja de especificación de datos, estos valores resultantes están por debajo del mínimo (100 mS). Sin embargo, las condiciones de prueba especificadas en la hoja de datos ($V_{ds} = 10V$ e $I_d = 200mA$) son muy diferentes a las condiciones realizadas en este proyecto ($V_{ds} \sim 2V$ e $I_d [200\mu A \sim 1.15 mA]$).

Para la gráfica del **parámetro r_d** , se observa que la ecuación $r_d = \frac{1}{\lambda I_{dq}}$ es la de una hipérbola, con variable independiente I_{dq} y variable dependiente r_d . La gráfica resultante

en la tabla 42 muestra una parte de esta hipérbola, por lo que es el comportamiento esperado para este parámetro r_d .

8.1.3 Comparación de resultados de los parámetros del transistor MOSFET 2N7000

Comparando el voltaje de umbral V_{th} con lo que especifican las hojas de datos, se observa que

$$0.8V < V_{th} = 1.84V < 3V$$

El voltaje de umbral $V_{th} = 1.84V$ está dentro del rango especificado.

No se pueden comparar los parámetros K_n , I_{dss} y lambda puesto que no están especificados en las hojas de datos. Sin embargo, los parámetros K_n , V_{th} e I_{dss} están relacionados:

$$I_{dss} = K_n V_{th}^2 = 7.74 \frac{mA}{V^2} * 1.84V^2$$

$$I_{dss} = 26.3 mA$$

8.2. Circuito generador de señal senoidal (b)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

8.2.1 Comparación de la gráfica resultante con la forma de onda senoidal

Comparando la gráfica con la forma de onda senoidal:

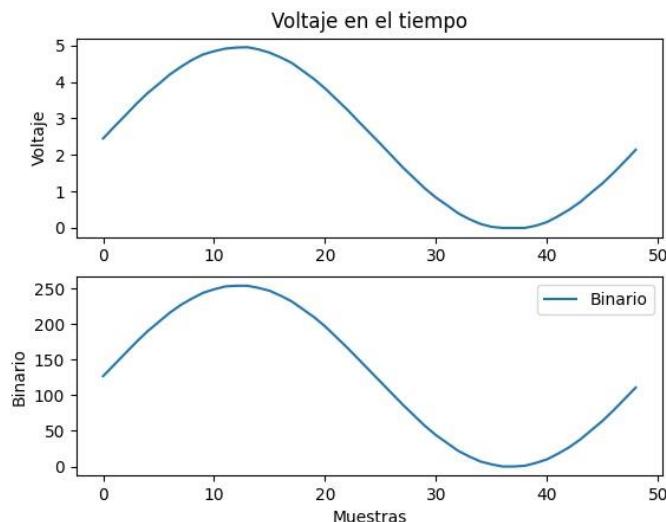


Figura 308. Comparación de la forma de onda del valor binario enviado con el voltaje de salida

La **gráfica 0: Voltajes en el tiempo** muestra que la forma de onda senoidal tanto para valores binarios como para el voltaje a la salida tienen forma senoidal.

Valor binario (en hexadecimal)	Valor binario (en decimal)	Voltaje a la salida (V)
0x7F	127	2.45
0xBE	190	3.69
0xEC	236	4.6
0xFE	254	4.94
0XF0	240	4.67
0xC5	197	3.82
0x87	135	2.61
0x48	72	1.37
0x16	22	0.4
0x00	0	0
0x0A	10	0.16
0x33	51	0.97
0x6F	111	2.14

Tabla 43. Valores seleccionados de valor binario y voltaje a la salida.

De la tabla 43 se observa que los valores en binario sí generan una amplitud de voltaje de 0V (valor binario de 0) a 5V (valor binario de 255) y que cualquier valor intermedio de voltaje a la salida es directamente proporcional al valor en binario enviado, como se verifica en la siguiente gráfica de la figura 309:

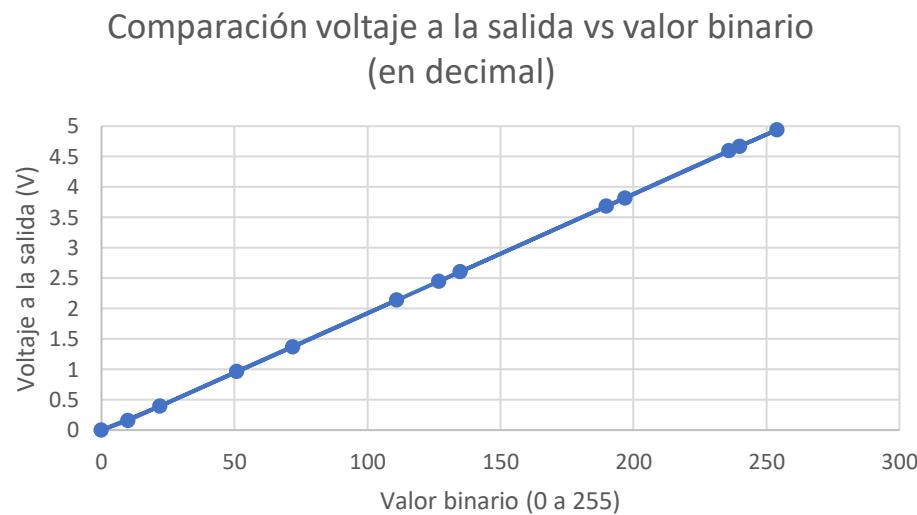


Figura 309. Comparación voltaje a la salida vs valor binario.

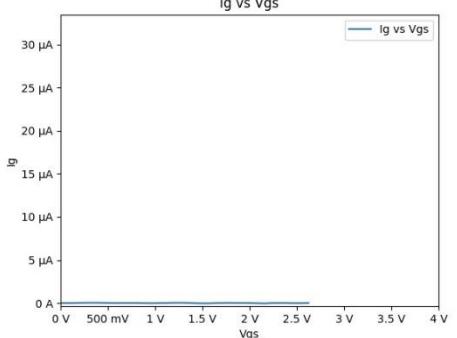
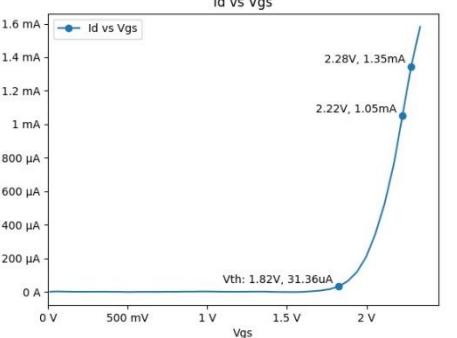
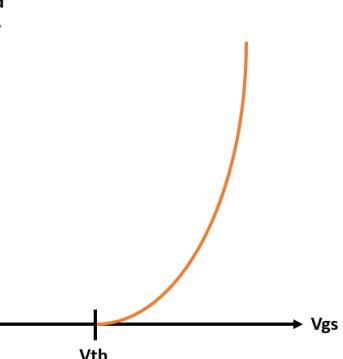
8.3. Amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos

- Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- Amplificador de una etapa (c)** con un transistor MOSFET.
- Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

8.3.1 Comparación de las gráficas resultantes para las curvas características de entrada, transferencia y salida

Comparando las gráficas de las **curvas características de entrada, transferencia y de salida**:

Tipo de gráfica	Gráfica resultante	Gráfica del marco teórico
Curva característica de entrada	 <p>Graph of I_g vs V_{gs} (labeled Ig vs Vgs). The y-axis ranges from 0 A to 30 μA with increments of 5 μA. The x-axis ranges from 0 V to 4 V with increments of 0.5 V. The curve is a straight line starting at (0,0) and ending at approximately (4, 30).</p>	 <p>Theoretical graph of I_g vs V_{gs}. The current is constant at a positive value for $V_{gs} > V_{th}$.</p>
Curva característica de transferencia	 <p>Graph of I_d vs V_{gs} (labeled Id vs Vgs). The y-axis ranges from 0 A to 1.6 mA with increments of 0.2 mA. The x-axis ranges from 0 V to 2 V with increments of 0.5 V. The curve starts at $(V_{th}, 31.36 \mu A)$ and increases rapidly, reaching $(2.28V, 1.35mA)$ and $(2.22V, 1.05mA)$.</p>	 <p>Theoretical graph of I_d vs V_{gs}. The current is zero for $V_{gs} < V_{th}$ and increases sharply for $V_{gs} > V_{th}$.</p>

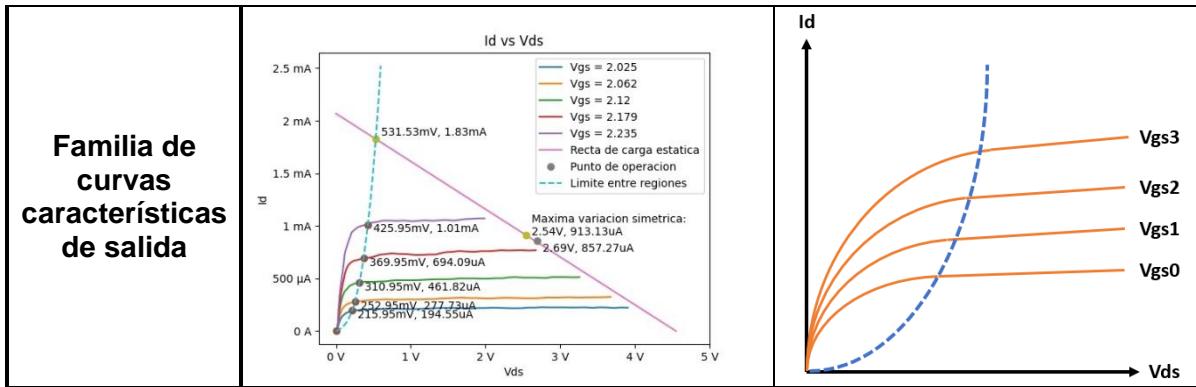


Tabla 44. Tabla comparativa de las gráficas resultantes.

Se observa que las formas de onda de las gráficas resultantes mostradas en la tabla 41 en la columna “gráfica resultante” son muy parecidas a las que se esperaban, mostradas en la tabla 41 en la columna “gráfica del marco teórico”.

En el caso de la **curva característica de entrada**, fue de esperar que la corriente de compuerta I_g fuera muy cercana a 0A.

En el caso de la **curva característica de transferencia**, se observa en la gráfica resultante que a partir del valor del voltaje V_{gs} de 1.82V, la corriente I_d es diferente de 0A. Esta forma de onda de la gráfica resultante coincide con la gráfica esperada mostrada en el marco teórico.

En el caso de la **familia de curvas características de salida**, se observa en la gráfica resultante que para cada voltaje V_{gs} elegido, se forma una curva I_d vs V_{ds} diferente. Este comportamiento es el esperado al comparar la gráfica resultante con la gráfica mostrada en el marco teórico. Sin embargo, en esta gráfica se observa un poco de ruido en cada una de las curvas I_d vs V_{ds} , a pesar de que se utiliza el filtro circular sobre las muestras tomadas. Este ruido puede ser producido por las fuentes de alimentación utilizadas, los cables usados y el protoboard en donde se armó el circuito.

Conforme el voltaje V_{gs} aumenta, la curva I_d vs V_{ds} es más corta. Es decir, cada vez es posible graficarla para un menor rango de voltaje V_{ds} . Se debe a la limitación de corriente del circuito PCF8591. Cuando se le exige una mayor corriente a su salida analógica, el voltaje se cae.

Relacionando el voltaje V_{gs} con el punto de saturación asociado a cada curva I_d vs V_{ds} se observa:

Voltaje V_{gs} (V)	Voltaje $V_{ds(sat)}$ (mV)	Corriente $I_{d(sat)}$ (uA)
2.025 V	215.95 mV	194.55 μ A
2.062 V	252.95 mV	277.73 μ A
2.12 V	310.95 mV	461.82 μ A
2.179 V	369.95 mV	694.09 μ A
2.235 V	425.93 mV	1010 μ A

Tabla 45. Comparación del voltaje V_{gs} con el voltaje $V_{ds(sat)}$ y la corriente $I_{ds(sat)}$ para las curvas I_d vs V_{ds}

Se observa que en casi todos los casos el voltaje $V_{ds(sat)}$ es menor al máximo establecido en las hojas de datos de 0.4V (excepto para $V_{gs} = 2.235$ V).

8.3.2 Comparación de las gráficas resultantes de los parámetros g_m y r_d

Comparando los resultados de los parámetros g_m y r_d :

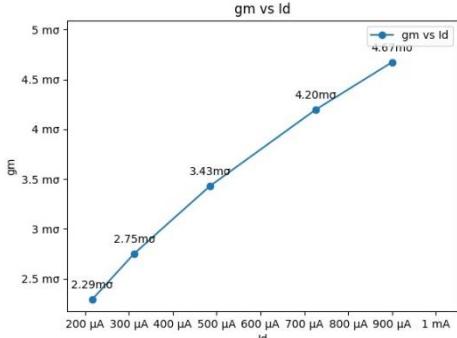
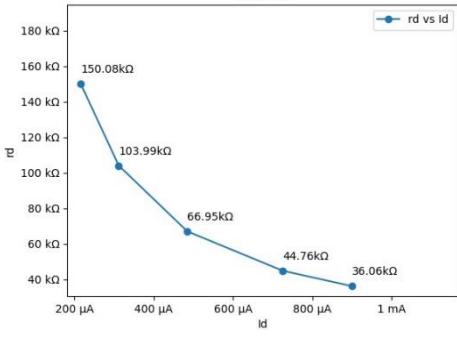
Tipo de gráfica	Gráfica resultante	Ecuación de la gráfica en el marco teórico																		
Curva del comportamiento del parámetro g_m	 <p>The graph plots g_m (mS) against I_d (μA). The x-axis ranges from 200 μA to 1 mA, and the y-axis ranges from 2.5 mS to 5 mS. The data points show a linear increase of g_m with I_d.</p> <table border="1"> <thead> <tr> <th>I_d (μA)</th> <th>g_m (mS)</th> </tr> </thead> <tbody> <tr><td>200</td><td>2.29</td></tr> <tr><td>300</td><td>2.75</td></tr> <tr><td>400</td><td>3.21</td></tr> <tr><td>500</td><td>3.43</td></tr> <tr><td>600</td><td>3.65</td></tr> <tr><td>700</td><td>3.87</td></tr> <tr><td>800</td><td>4.09</td></tr> <tr><td>900</td><td>4.31</td></tr> </tbody> </table>	I_d (μA)	g_m (mS)	200	2.29	300	2.75	400	3.21	500	3.43	600	3.65	700	3.87	800	4.09	900	4.31	$g_m = 2\sqrt{K_n I_{dq}}$
I_d (μA)	g_m (mS)																			
200	2.29																			
300	2.75																			
400	3.21																			
500	3.43																			
600	3.65																			
700	3.87																			
800	4.09																			
900	4.31																			
Curva del comportamiento del parámetro r_d	 <p>The graph plots r_d (kΩ) against I_d (μA). The x-axis ranges from 200 μA to 1 mA, and the y-axis ranges from 40 kΩ to 180 kΩ. The data points show a linear decrease of r_d with I_d.</p> <table border="1"> <thead> <tr> <th>I_d (μA)</th> <th>r_d (kΩ)</th> </tr> </thead> <tbody> <tr><td>200</td><td>150.08</td></tr> <tr><td>300</td><td>103.99</td></tr> <tr><td>400</td><td>66.95</td></tr> <tr><td>500</td><td>44.76</td></tr> <tr><td>600</td><td>36.06</td></tr> </tbody> </table>	I_d (μA)	r_d (k Ω)	200	150.08	300	103.99	400	66.95	500	44.76	600	36.06	$r_d = \frac{1}{\lambda I_{dq}}$						
I_d (μA)	r_d (k Ω)																			
200	150.08																			
300	103.99																			
400	66.95																			
500	44.76																			
600	36.06																			

Tabla 46. Tabla comparativa de las gráficas de los parámetros g_m y r_d

Para la gráfica del parámetro g_m , se observa que la ecuación $g_m = 2\sqrt{K_n I_{dq}}$ es la de una parábola horizontal que abre hacia la derecha, con variable independiente I_{dq} y variable dependiente g_m . La gráfica resultante en la tabla 46 muestra una parte de esta parábola horizontal, por lo que es el comportamiento esperado para este parámetro g_m .

Sin embargo, al comparar los valores resultantes de g_m con la hoja de especificación de datos, estos valores resultantes están por debajo del mínimo (100 mS). Sin embargo, las condiciones de prueba especificadas en la hoja de datos ($V_{ds} = 10V$ e $I_d = 200mA$) son muy diferentes a las condiciones realizadas en este proyecto ($V_{ds} \sim 2V$ e $I_d = [200\mu A \sim 1.15 mA]$).

Para la gráfica del **parámetro r_d** , se observa que la ecuación $r_d = \frac{1}{\lambda I_{dq}}$ es la de una hipérbola, con variable independiente I_{dq} y variable dependiente r_d . La gráfica resultante en la tabla 46 muestra una parte de esta hipérbola, por lo que es el comportamiento esperado para este parámetro r_d .

8.3.3 Comparación de resultados de los parámetros del transistor MOSFET 2N7000

Comparando el voltaje de umbral V_{th} con lo que especifican las hojas de datos, se observa que

$$0.8V < V_{th} = 1.81V < 3V$$

El voltaje de umbral $V_{th} = 1.81V$ está dentro del rango especificado.

No se pueden comparar los parámetros K_n , I_{dss} y lambda puesto que no están especificados en las hojas de datos. Sin embargo, los parámetros K_n , V_{th} e I_{dss} están relacionados:

$$I_{dss} = K_n V_{th}^2 = 6.07 \frac{mA}{V^2} * 1.81^2$$

$$I_{dss} = 19.86 mA$$

8.3.3.1 Comparación de resultados de los parámetros híbridos entre los resultados para circuitos trazador de curvas (a) y los resultados para el amplificador de una etapa (c)

Se observó que, aunque se midan los parámetros el mismo transistor bajo las mismas condiciones de prueba, se obtienen resultados ligeramente diferentes.

Comparando los resultados de los dos circuitos:

Parámetro	Valor resultante (Circuito trazador de curvas (a))	Valor resultante (Amplificador de una etapa (c))
λ	$32.97 \frac{1}{mV}$	$30.8 \frac{1}{mV}$
V_{th}	1.84 V	1.81 V
I_{dss}	26.3 mA	19.85 mA
K_n	$7.74 \frac{mA}{V^2}$	$6.07 \frac{mA}{V^2}$
r_g	∞	∞

Tabla 47. Comparación de resultados

Se observa que los parámetros I_{dss} y K_n son los que tienen mayores diferencias entre los resultados de ambos circuitos. Se puede mejorar la lectura y cálculo de estos parámetros, puesto que se deben de resultar las mismas mediciones para el mismo transistor bajo las mismas condiciones de prueba.

Estas diferencias se deben principalmente a los cambios de voltaje en la fuente de alimentación el ruido proveniente de los cables utilizados, los mismos circuitos y el protoboard donde se armó el circuito.

8.3.4 Comparación de resultados de los parámetros en el punto de operación del amplificador de una etapa

Comparando los parámetros del punto de operación con los presentados en el marco teórico:

Parámetro	Análisis teórico	Análisis con el proyecto propuesto	Porcentaje de error
V_{dsq}	2.87 V	2.69 V	6.27%
V_{gsq}	2.18 V	2.205 V	1.15%
I_{dq}	830.983 μA	857.27 μA	3.16%
g_m	4.56 mS	4.56 mS	0%
r_d	40.113 k Ω	37.88 k Ω	5.56%
r_g	∞	∞	-
R_{lac}	2.08 k Ω	2.08 k Ω	0%
Ganancia del amplificador	-9.48	-13.45	41.87%

Tabla 48. Comparación de resultados para el amplificador en fuente común.

Se observa que los porcentajes de error para todos los parámetros (excepto ganancia del amplificador) son menores al 10%. Esto quiere decir que el sistema desarrollado en este proyecto en estas mediciones es fiable.

Sin embargo, en la ganancia del amplificador es la que presenta una mayor diferencia con respecto a la ganancia teórica, por lo que se tienen que realizar ajustes para mejorar este cálculo. Este error tan grande se debe a que la fuente de alimentación presenta cambios grandes en el voltaje de alimentación.

8.4. Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Como se observó en la sección de resultados 7.4, tanto para el **circuito secuencial implementado por hardware** como para el **circuito secuencial implementado por software** se obtuvieron los mismos resultados.

8.4.1 Comparación de los resultados del funcionamiento de la máquina de estados finitos

Comparando los resultados de la máquina de estados con lo presentado en la sección 6.3.4.2.1:

Entrada $I_2 I_1 I_0$	Estado esperado $S_1 S_0$	Salida esperada $O_2 O_1 O_0$	Estado resultante $S_1 S_0$	Salida resultante $O_2 O_1 O_0$
000	00	000	00	000
100	10	100	10	100
101	01	001	01	001
111	11	011	11	011
101	01	001	01	001
100	01	001	01	001
110	11	011	11	011
100	10	100	10	100
100	10	100	10	100
000	00	000	00	000

Tabla 49. Comparación del funcionamiento de la máquina de estados finitos aplicándole el vector de prueba. (El estado inicial es 00)

Se observa tanto el **estado resultante** de la máquina de estados finitos como la **salida resultante** de la máquina de estados son iguales que el **estado esperado** y la **salida esperada** respectivamente.

El funcionamiento de la máquina de estados finitos es también comprobable con la **gráfica 1: Tabla de estados binarios**, presentada en las figuras 300 y 301.

8.4.2 Comparación de las gráficas resultantes que contienen los diagramas de estados

Comparando los dos diagramas de estados con el que se muestra en el marco teórico:

Diagrama de estados	Gráfica del diagrama de estados
Diagrama de estados presentado en el marco teórico	
Diagrama de estados 1 resultante	<p>Cada círculo es un estado. Cada flecha representa un cambio de estado, indicado por el número adyacente. Las combinaciones de transiciones entre estados que no aparecen en el diagrama indican que no hay cambio de estado cuando ocurren. Si hay 'X' en el número de una flecha significa que no importa el valor de ese bit.</p>
Diagrama de estados 2 resultante	<p>Los estados son: ['q0 Apagado', 'q1 Encendido funcionando', 'q2 Corto circuito', 'q3 Reset'] Los salidas son: ['bit 2: Relevador ON/OFF', 'bit 1: LED de Reset', 'bit 0: LED de Corto circuito'] Los entradas son: ['bit 2: Botón de Alimentación', 'bit 1: Botón de Reset', 'bit 0: Sensor de Corto Circuito']</p> <p>Cada círculo es un estado. Cada flecha representa un cambio de estado, indicado por el número adyacente. Las combinaciones de transiciones entre estados que no aparecen en el diagrama indican que no hay cambio de estado cuando ocurren. Si hay 'X' en el número de una flecha significa que no importa el valor de ese bit.</p>

Tabla 50. Comparación de las gráficas resultantes que contienen diagramas de estados.

Se observa que los tres diagramas de estados de la tabla 50 son equivalentes. Puesto que estos diagramas de estados también representan el comportamiento de la máquina de estados, se puede comprobar que estos tres diagramas son equivalentes con la **gráfica 1: Tabla de estados binarios**, presentada en las figuras 300 y 301.

9. Conclusiones

Con base en los resultados observados, el presente proyecto “Realidad Aumentada como una Herramienta para el Análisis y Diseño de circuitos Electrónicos Analógicos utilizando tecnología MOS y Circuitos Digitales Secuenciales” cumple el objetivo de mostrar la información más relevante del circuito, como parámetros eléctricos, gráficas y funcionamiento del circuito sobre la misma imagen del circuito.

Se puede mejorar este proyecto, sobre todo en la parte de la adquisición de datos para obtener lecturas más estables y mediciones más exactas y precisas de los parámetros del circuito, sobre todo en circuitos analógicos.

10. Referencias

- [1] Baran, B., Yecan, E., Kaptan y Paşayığit O., "Using augmented reality to teach fifth grade students about electrical circuits", *Education and Information Technologies*, vol. 25, pp. 1371–1385, marzo 2020.
- [2] Restivo T., Chouzal F., Rodrigues J., Menezes P. y Lopes J. B., "Augmented reality to improve STEM motivation", 2014 IEEE Global Engineering Education Conference (EDUCON), Estambul, 2014, pp. 803-806, doi: 10.1109/EDUCON.2014.6826187.
- [3] Urbano D., Fátima Chouzal M., Teresa Restivo M., "Evaluating an online augmented reality puzzle for DC circuits: Students' feedback and conceptual knowledge gain", Computer Applications in Engineering Education, vol. 28, no. 5, pp. 1355-1368, septiembre 2020.
- [4] Reyes F., "Realidad aumentada móvil aplicada a mediciones eléctricas", Tesis MCC, Universidad Autónoma Metropolitana Unidad Azcapotzalco, CDMX, Ciudad de México, 2017.
- [5] Ramírez F., "19.2 Convenciones y Características de los Transistores FET", febrero 20, 2014. Accedido el 9 de octubre de 2021. [Archivo de video]. Disponible: <https://youtu.be/otxMxj02M5g>.
- [6] Ramírez F., "20.2 Ejemplo de Análisis de un Circuitos con Transistor NMOS", febrero 22, 2014. Accedido el 9 de octubre de 2021. [Archivo de video]. Disponible: <https://youtu.be/PbWXZbC8yqg>.
- [7] Ramírez F., "22.2 Amplificador FET - Concepto de Pequeña Señal", febrero 27, 2014. Accedido el 9 de octubre de 2021. [Archivo de video]. Disponible: <https://youtu.be/FyMwiQEEt10>.
- [8] Ramírez F., "22.3 Amplificador FET - Modelo Híbridod PI", febrero 27, 2014. Accedido el 9 de octubre de 2021. [Archivo de video]. Disponible: <https://youtu.be/zW8IMDKaXTw>.
- [9] Ramírez F., "23.1 Amplificador MOSFET en Fuente Común", marzo 1, 2014. Accedido el 9 de octubre de 2021. [Archivo de video]. Disponible: <https://youtu.be/QUSLGWWr2Ow>.
- [10] Murphy E. y Slattery C., "Ask The Application Engineer—33: All About Direct Digital Synthesis", *Analog Dialogue*, vol. 38, no. 3, pp. 8-12, agosto 2004. [Online]. Disponible: <https://www.analog.com/media/en/analog-dialogue/volume-38/number-3/articles/volume38-number3.pdf>.

[11] Kuphaldt T., Crunkilton D., Papaliakos G et. al., *Lessons in Electric Circuits: Vol IV – Digital*, Idaho: EETech Media LLC., 2014. Accedido el 20 de julio de 2021. [Online]. Disponible: <https://www.allaboutcircuits.com/textbook/digital/>

[12] Aaronson, S., Conferencia de clase, Tema: “Automata, Computability, and Complexity: Lecture 3”, 6.045J, Massachusetts Institute of Technology, Cambridge, MA, Estados Unidos, marzo 2011. [Online] Accedido el 2 de octubre de 2021. Disponible: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/lecture-notes/MIT6_045JS11_lec03.pdf.

[13] OnSemi LLC., “N-Channel Enhancement Mode Field Effect Transistor 2N7000, 2N7002, NDS7002A”, NDS7002A/D datasheet, agosto 2021. [Online]. Disponible: <https://www.onsemi.com/pdf/datasheet/nds7002a-d.pdf>.

[14] NXP Semiconductors, “PCF8591 8-bit A/D and D/A converter”, junio 2013. [Online]. Disponible: <https://www.nxp.com/docs/en/data-sheet/PCF8591.pdf>.

[15] Texas Instrumens “SN5474, SN54LS74A, SN54S74, SN7474, SN74LS74A, SN74S74 Dual D-Type Positive-Edge-Trigereed Flip-Flops with Reset and Clear”, SDLS119 datasheet, marzo 1988. [Online]

10.1 Software utilizado

Para realizar los programas del microcontrolador, se utilizó **Arduino IDE**.

Para realizar los programas en Python, se utilizó **Python IDLE** para Python 3.

Para realizar los diagramas esquemáticos, se utilizó **Multisim** y **Kicad Eeschema**.

Para realizar los diagramas de clases, se utilizó **Pycharm 2021.2.2 (Professional Edition)**.

Para realizar la edición de imágenes y figuras, se utilizó **Microsoft PowerPoint**.

Para realizar la edición de este documento, se utilizó **Microsoft Word**.

11. Entregables

11.1 Programas base

En esta sección se presentan los programas proporcionados por el asesor, al inicio del proyecto terminal. Estos programas incluyen:

1. Programa de Arduino (1)
2. Programas en Python (2)
 - a. main.py (2a)
 - b. modelo.py (2b)
 - i. comunicación.py (2bi)
 - ii. muestras.py (2bii)
 - iii. marcadores.py (2biii)
 - iv. clasificador.py (2biv)
 - v. graficador.py (2bv)
 - vi. display.py (2bvi)
 - vii. asignador.py (2bvii)
 - c. vista.py (2c)
 - d. controlador.py (2d)
 - i. eventomouse.py (2di)

11.1.1 Programa base de Arduino (1)

Programa base de Arduino



Figura 310. Diagrama a bloques del programa base de Arduino.

```
*****
PROGRAMA: 22032021-A

Sisyema de Adquisicion de 4 CANALES Analogicos

Este programa se ejecuta en conjunto con:

ARDUINO: Completo AR 02.ino      (entradas) 22032021-A
PYTHON:  Completo AR 02.py (monitor)  22032021-P

CIRCUITO DE PRUEBA:
PROYECTO TECNOLOGICO: DDS basado en un registro de corrimiento
4015B

30 / marzo / 2021

Adaptado de: https://www.youtube.com/watch?v=8YEvU2Zi\_LU
***** /



// --- Definicion de pines para los 4 canales del ADC ---
const int PinCanal0 = A0; // Pin de entrada analogica canal 0
const int PinCanal1 = A1; // Pin de entrada analogica canal 1
const int PinCanal2 = A2; // Pin de entrada analogica canal 2
const int PinCanal3 = A3; // Pin de entrada analogica canal 3

// -----
// Declaracion de los siguientes arreglos:
// canales[4] -> valores analógicos leidos en cada canal
// voltaje[4] -> voltajes escalados de los valores analógicos
//           escalamiento(analogicos: 0,1023 -> voltajes: 0,5)
// -----
int canales[4];
float voltajes[4];

// --- Variables para control tiempo de muestreo ---
```

```

unsigned long lastTime = 0;
unsigned long sampleTime = 50; //200;

void setup() {
    // --- Inicializa velocidad de transmision serial ---
    Serial.begin(9600);
}

void loop() {
    // --- Toma muestras cada 200 ms ---
    if (millis()-lastTime > sampleTime) {
        lastTime = millis();

        // --- Lectura de los canales ---
        canales[0] = analogRead(PinCanal0);
        canales[1] = analogRead(PinCanal1);
        canales[2] = analogRead(PinCanal2);
        canales[3] = analogRead(PinCanal3);

        // --- Escalamiento de canales ---
        voltajes[0] = scaling(canales[0], 0, 1023, 0, 5);
        voltajes[1] = scaling(canales[1], 0, 1023, 0, 5);
        voltajes[2] = scaling(canales[2], 0, 1023, 0, 5);
        voltajes[3] = scaling(canales[3], 0, 1023, 0, 5);

        // --- Transmision de cada valor al programa en Python ---
        Serial.println(voltajes[0]);
        Serial.println(voltajes[1]);
        Serial.println(voltajes[2]);
        Serial.println(voltajes[3]);
    }
}

float scaling(float x, float in_min, float in_max, float out_min, float
out_max)
{
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

```

Figura 311. Programa completo base de Arduino

11.1.2 Programa base de Python (2)

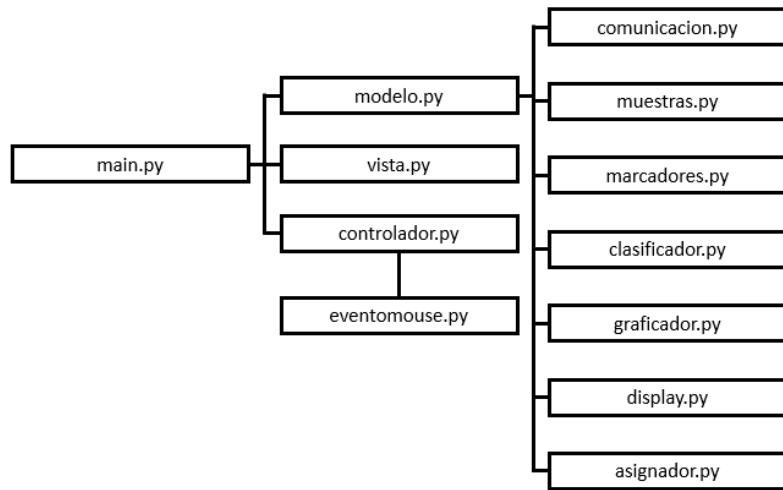


Figura 312. Diagrama de clases completo del programa base de Python.

11.1.2.1 Clase `main.py` (2a)

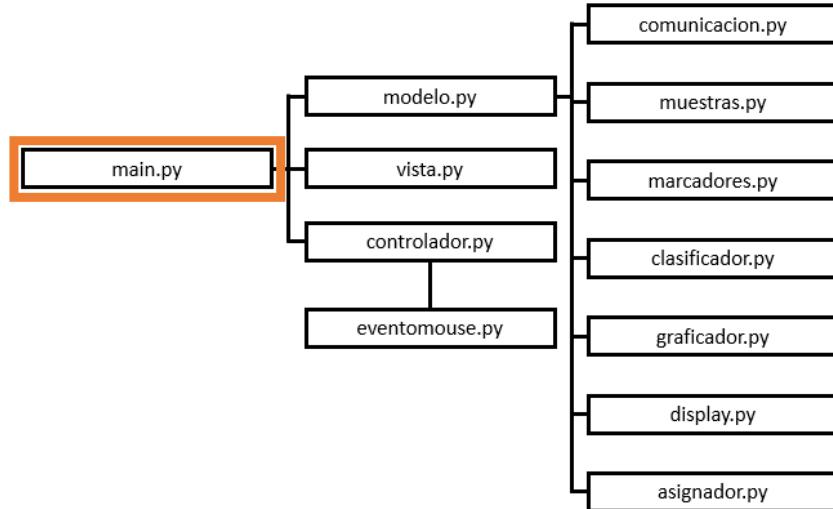


Figura 313. Ubicación de la clase `main.py`.

```

# *****
#
# PROYECTO de Realidad Aumentada - Version 3.0
#
# Programa main.py
#
# Descripción:
# Identifica los marcadores ArUco colocados sobre la imagen
# de un circuito en protoboard y genera ventanas flotantes
# ["display" + "graficas"], mostrando la información recibida
# de ARDUNIO para cada marcador
#
# Clases de apoyo:
# MODELO DE PROGRAMACION: Modelo - Vista - Controlador (aproximacion)
#
# Fecha: junio 8/2021
#
# *****

import imutils
import cv2

from modelo import Modelo
from controlador import Controlador
from vista import Vista

# *** LECTURA de imagen y ACONDICIONAMIENTO *****

# --- Proto con marcadores a 10% de su tamaño con marco ---
image = cv2.imread("i/marcador 8B.jpg") # Imagen con 8 marcadores

print("")
print("En main:: imagen -> marcador 8B.jpg")
print("")

# --- Sin importar el tamaño de la imagen, se escala a width = 600 ---
image = imutils.resize(image, width=1000)
# *** FIN de LECTURA de imagen y ACONDICIONAMIENTO *****

# *** PROGRAMA PRINCIPAL *****

# --- MODELO propuesto para AR ---
model = Modelo(image)

```

```
# --- CONTROL de eventos de mouse ---
control = Controlador(image,model)
# --- LOOP infinito ---
vista = Vista(image,control.mse)
# *** FIN de Programa Principal ****
cv2.destroyAllWindows()
```

Figura 314. Código completo de la clase main.py

11.1.2.2 Clase *modelo.py* (2b)

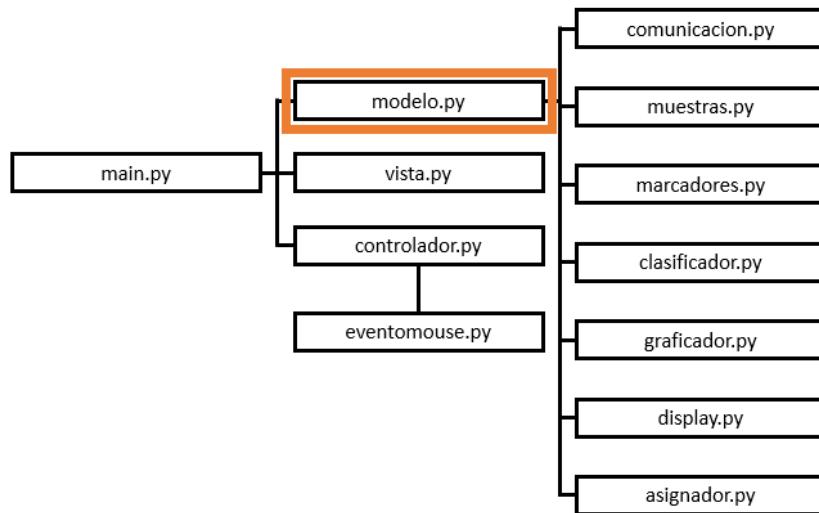


Figura 315. Ubicación de la clase *modelo.py*.

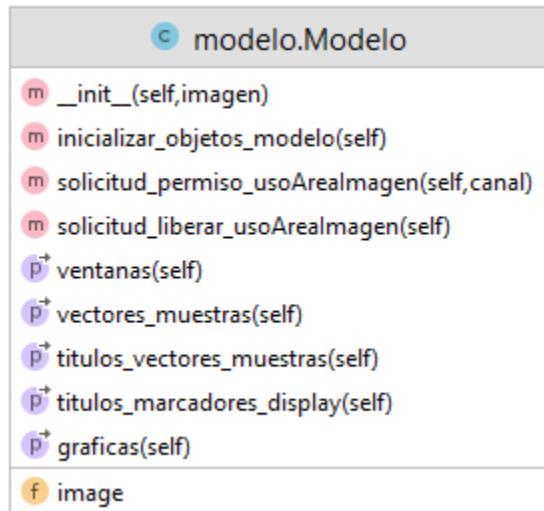


Figura 316. Métodos, atributos y propiedades de la clase *modelo.py*.

```

# ****
#
# Clase: --> Modelo
# Modulo: -> modelo.py
#
# Descripción:
#   Genera el modelo utilizado para Realidad Aumentada
#
# Clases de apoyo:
#
#   - Clase: Comunicación <-- Comunicacion Serial con ARDUINO
#   - Clase: Muestras   <-- Lectura de muestras procedentes de ARDUINO
#   - Clase: Marcadores <-- Detección de marcadores ArUco
#   - Clase: Clasificador <-- Clasificador de marcadores
#   - Clase: Display    <-- Genera "ventanas" para cada marcador [display+grafica]
#   - Clase: Graficador <-- Genera las gráficas para cada marcador
#
# Fecha: junio 8/2021
#
# ****

import imutils

from m.comunicacion import Comunicacion
from m.muestras import Muestras
from m.marcadores import Marcadores
from m.clasificador import Clasificador
from m.graficador import Graficador
from m.display import Display as disp
from m.asignador import Asignador

ventanas = []
vectores_muestras = {}
titulos_vectores_muestras = {}
titulos_marcadores_display = {}
graficas = []

class Modelo:

    # -----
    #      CONSTRUCTOR
    # -----
    def __init__(self,imagen):

```

```

print(" CONSTRUCTOR: Clase: Modelo")
self.image = imagen
self.inicializar_objetos_modelo()

def inicializar_objetos_modelo(self):

    h, w, channels = self.image.shape
    print(" Modelo: --> self.image.shape = ",self.image.shape)

    # --- 1 - Comunicacion con ARDUINO ---
    arduino = Comunicacion()
    conexionSerial = arduino.iniciarConexion_arduino()

    # --- 2 - Lectura de valores enviados por ARDUINO
    global titulos_vectores_muestras
    titulos_vectores_muestras = ["A0","A1","A2","A3"]
    titulos_vectores_muestras = ["VBB","Vbe","Ib","VCC","Vce","Ic"]
    param_muestras = {'numMuestras': 300, 'numCanales': len(titulos_vectores_muestras),
                      'llave': titulos_vectores_muestras}
    muestras = Muestras(conexionSerial,param_muestras)
    global vectores_muestras
    vectores_muestras = muestras.muestras
    # --- Imprimir diccionario 'muestras' ---
    #muestras.imprimir_valores_recibidos()
    #muestras.imprimir_valores_procesados()

    # --- 3 - Manejo de marcadores ArUco ---
    aruco = Marcadores(self.image)
    #aruco.dibujar_cuadros_marcadores()
    # --- Imprimir diccionario ---
    #aruco.imprimir_diccionario()
    # --- diccionario --> Arreglo de marcadores ArUco ---
    diccionario = aruco.diccionario

    # --- 4 - Clasificar marcadores ArUCo ---
    # --- Constantes DISPLAY: Estructura de Datos ---
    parametros_display = {
        'ancho_display': 240, # AUTOMATICO
        'alto_display': 30,
        'h_gap_display': 5,
        'offset_x': 5, # 65 Offset en X debido al movimiento 'easing'
        'v_gap_display': 5,
        'offset': 20,
    }

```

```

    'alto_imagen': h,
    'ancho_imagen': w,
}
c = Clasificador(diccionario,parametros_display)
diccionario = c.diccionario
# --- Imprimir diccionario ---
#c.imprimir_diccionario()

# --- 5 - Manejo de ventanas: display + gráfica ---
global ventanas
global titulos_marcadores_display
titulos_marcadores_display = {'m0': "Ic vs Vbe",
'm2': "Ib vs Vbe",
'm3': "Ic vs Vce",
'm4': "Ic vs Vbe pc",
'm5': "Ib vs Vbe pc",
'm6': "Ic vs Vce pc",
'm7': "Ic vs Vce Ip",
'm10': " Ib vs Vbe Ip"}

# --- Titulos MARCADORES ---
print("")
print(" list(titulos_marcadores_display): ",list(titulos_marcadores_display))

for i in range(len(diccionario)):
    # --- Crear Display --
    ventana = disp(self,self.image,diccionario[i],
                  titulos_marcadores_display,parametros_display)
    ventanas.append(ventana)

# --- 6 - Genera graficas con muestras leidas ---
graf = Graficador(self)

# --- Trazador de curvas para BJT: BC547B
#   En el programa de ARDUINO se debe seleccionar:
#   'Transistor = true'
# -----
graf.graficar_curvas_BJT()  # 8 marcadores y 8 graficas

# --- Trazador de curvas para MOSFET: 2N7000
#   En el programa de ARDUINO se debe seleccionar:
#   'Transistor = false'
# -----
#graf.graficar_curvas_MOSFET()  # 8 marcadores y 8 graficas

```

```

# --- Servicios de impresion ---
#graf.imprimir_archivos_graficas()
#graf.imprimir_muestras_canal()
#graf.imprimir_vectores_muestras()

# --- Obtener graficas ---
global graficas
graficas = graf.graficas

# --- 7 - Asignar gráficas a cada ventana ---
for i in range(len(ventanas)):
    ventanas[i].asignar_imagen_sobrepuerta(graf.graficas[i])

def solicitud_permiso_usoArealImagen(self,canal):

    # --- Se activa desde la clase Display
    #   metodo: actualizar_imagen_display(self,img)
    #
    for v in ventanas:
        if v.diccionario['id'] == canal:
            v.permitir_ocupar_arealmagen = True
        else:
            v.permitir_abrir_ventana = False
            v.permitir_ocupar_arealmagen = False

def solicitud_liberar_usoArealImagen(self):

    # --- Se activa desde la clase Display
    #   metodo: actualizar_imagen_display(self,img)
    #
    for v in ventanas:
        v.permitir_abrir_ventana = True
        v.permitir_ocupar_arealmagen = True

@property
def ventanas(self):
    return ventanas

@property
def vectores_muestras(self):
    return vectores_muestras

@property

```

```
def titulos_vectores_muestras(self):
    return titulos_vectores_muestras

@property
def titulos_marcadores_display(self):
    return titulos_marcadores_display

@property
def graficas(self):
    return graficas
```

Figura 317. Código completo de la clase modelo.py

11.1.2.2.1 Clase *comunicacion.py* (2bi)

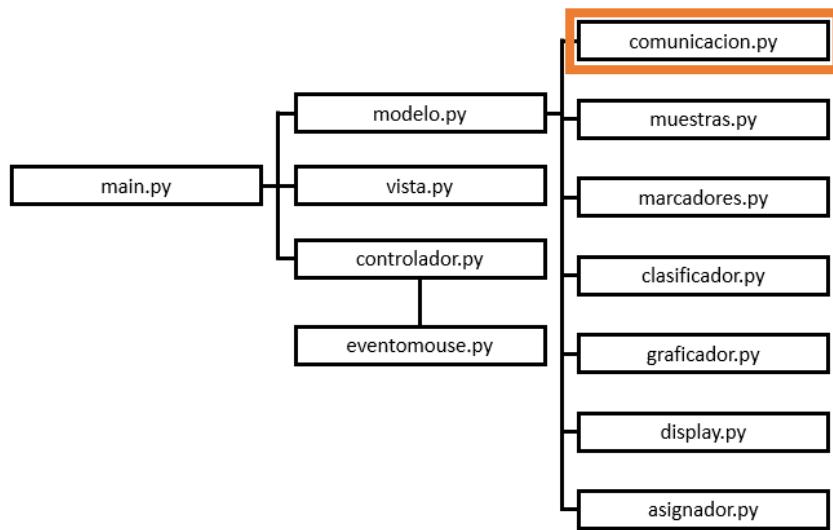


Figura 318. Ubicación de la clase *comunicacion.py*.

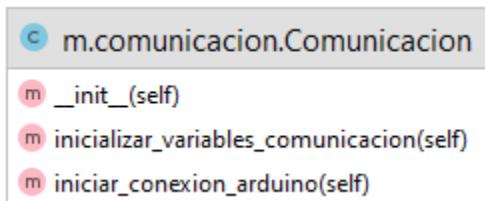


Figura 319. Métodos, atributos y propiedades de la clase *comunicación.py*.

```

# ****
#
# Clase: --> Comunicación
# Modulo: -> comunicacion.py
#
# Descripción:
# Intenta la conexión con ARDUINO a través
# del puerto Seral USB
#
# Fecha: marzo 25/2021
#
# ****

import serial
import time

class Comunicacion:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self):

        self.inicializar_variables_comunicacion()

    def inicializar_variables_comunicacion(self):
        # -----
        # Definicion de variables de comunicacion serial (USB)
        # -----
        global puertoSerie
        global baudRate
        puertoSerie = 'COM5'  # Puerto serial de conexión con Arduino (USB)
        baudRate = 9600       # Velocidad de transmisión: 9600 baudios

    def iniciarConexion_arduino(self):
        # -----
        # Manejo del puerto serial (USB)
        # -----
        # --- Conexión con el puerto serial asignado a Arduino ---
        print("")
        print(" En CLASE Comunicacion::")
        print(" Comunicacion:: puertoSerie = ",puertoSerie)
        try:

```

```
# Instance serial object
conexionSerial = serial.Serial(puertoSerie, baudRate, timeout = 1)
#time.sleep(1) # (5) Retardo de espera para establecer comunicacion serial
print("*****")
print(" --- CONECTADO CON ARDUINO ---")
print("*****")
return conexionSerial
except:
    print("*****")
    print(" FALLO LA CONEXION CON ARDUINO")
    print("*****")
```

Figura 320. Código completo de la clase comunicacion.py

11.1.2.2.2 Clase muestras.py (2bii)

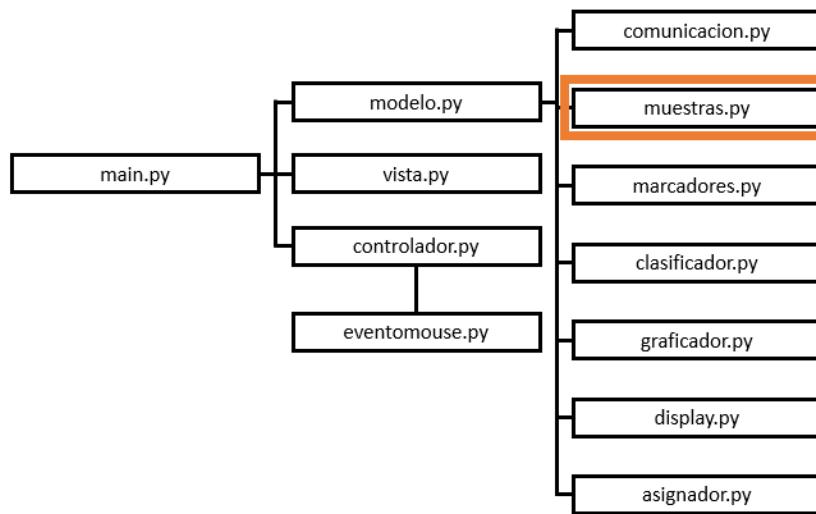


Figura 321. Ubicación de la clase muestras.py.

```
c m.muestras.Muestras
m __init__(self,conexionSerial,param_muestras)
m inicializar_variables_adquisicion(self)
m inicializar_arreglos_valores(self)
m iniciar_recepcion_datos(self)
m muestrear_datos_arduino(self)
m leer_muestras_arduino(self,numMuestras,numCanales,conexionSerial,           lineas,n)
m separar_muestras_canal(self)
m filtrar_muestras_obtenidas(self)
m promediador_datos_circular(self, vector_muestras, numero_muestras_promediar)
m promediar_circular(self,buffer)
m promediador_datos_lowpass(self, vector_muestras, alpha)
p+ muestras(self)
p+ valores(self)
m imprimir_valores_recibidos(self)
m imprimir_valores_procesados(self)
f param_muestras
f conexionSerial
```

Figura 322. Métodos, atributos y propiedades de la clase muestras.py.

```

# ****
#
# Clase: --> Muestras
# Modulo: -> muestras.py
#
# Descripción:
# Sigue el envío de valores capturados por Arduino:
#
# Fecha: junio 8/2021
#
# ****

import serial
import collections

from matplotlib.lines import Line2D
from collections import deque

muestas = {} # Diccionario de muestras: [{canal}: ch, datos: []}]

class Muestras:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,conexionSerial,param_muestas):

        self.conexionSerial = conexionSerial
        self.param_muestas = param_muestas

        self.inicializar_variables_adquisicion()
        self.inicializar_arreglos_valores()
        if self.iniciar_recepcion_datos():
            # --- Leer datos procedentes de Arduino ---
            self.muestrear_datos_arduino()
            # --- Separar muestras en grupo de 'paquetes de muestras'
            #   Actualizar diccionario 'muestas'
            #
            self.separar_muestras_canal()
            # --- Filtrar muestras 'circular' 'lowpass':
            #   Actualizar diccionario 'muestas'
            #
            self.filtrar_muestras_obtenidas()

```

```

# --- Imprimir diccionario 'muestras' ---
#self.imprimir_valores_recibidos()
#self.imprimir_valores_procesados()

def inicializar_variables_adquisicion(self):

    # -----
    #     Definicion de variables de muestreo
    # -----
    global numMuestras
    global numCanales
    global llaveMuestras
    global contadorMuestra
    global valores      # Arreglo
    global lineas       # Arreglo

    numMuestras = self.param_muestras['numMuestras'] # Número de numMuestra
    numCanales = self.param_muestras['numCanales']   # Numero de canales (arduino)
    llaveMuestras = self.param_muestras['llave']      # "llave" de las muestras
    contadorMuestra = numMuestras-1 # Contador de numMuestra
    lineas = []           # Arreglo de lineas de graficación
    valores = []          # Arreglo numCanalesXnumMuestras.
                           # Almacen de valores recibidos tipo COLA

def inicializar_arreglos_valores(self):

    # --- Inicializacion con ceros "0" de los vectores de cada canal de
    #     de valores recibidos de Arduino, almacenados en el arreglo "valores[]"
    #     Cantidad de valores por canal: "numMuestra"
    #     Numero de canales: "numCanales"
    # -----
    for i in range(numCanales):
        valores.append(collections.deque([0]*numMuestras, maxlen = numMuestras))
        lineas.append(Line2D([], [], color='blue'))

def iniciar_recepcion_datos(self):

    # --- En este METODO, Python recibe los datos enviados por ARDUINO y
    #     espera hasta recibir la palabra 'inicio', la cual es la clave
    #     para comenzar a recibir datos válidos de ARDUINO
    #     Esto es así porque ARDUINO tiene un BUFFER de datos de transmisión
    #     de aproximadamente 64 bytes de longitud. Puede ser que el buffer
    #     contenga 'basura' de transmisiones anteriores y es por ésto,
    #     que este método intenta leer el contenido del buffer hasta 64

```

```

#    veces. Si después de 64 intentos no encuentra la palabra 'inicio'
#    se reportará un problema de NO-SINCRONIZACION con ARDUINO
# -----
print("")
print(" En CLASE Muestras:: ")
print("")
print("*****")
print(" --- INICIANDO SINCRONIZACION DE DATOS CON ARDUINO ---")
print("*****")
print("")
numero_maximo_intentos = 64
inmensaje_recibido_arduino = ""

contador_intentos = 0
while (inmensaje_recibido_arduino != b'inicio'):
    inmensaje_recibido_arduino = self.conexionSerial.readline().strip()
    #print(" inmensaje_recibido_arduino = ",inmensaje_recibido_arduino)
    contador_intentos += 1
    #print("    contador_intentos = ",contador_intentos)
    if (contador_intentos >= numero_maximo_intentos):
        print("")
        print(" *** NO ES POSIBLE ESTABLECER UNA COMUNICACION ***")
        print("      SINCRONIZADA CON ARDUINO: ")
        print("      SE EXCDIO EL NUMERO DE INTENTOS -> 64 <-")
        print("*****")
        return False

print("")
print("*****")
print(" --- SINCRONIZACION ETABLECIDA CON ARDUINO ---")
print("*****")
print("")
return True

def muestrear_datos_arduino(self):

    # --- Recepcion de los valores enviados por Arduino, correspondientes
    #   a cada canal. Almacenados en el arraglo "valores[]"
    #   Cantidad de valores por canal: "numMuestra"
    #   Numero de canales: "numCanales"
    # -----
    global contadorMuestra
    for n in range(numMuestras):
        self.leer_muestras_arduino(numMuestras,numCanales,

```

```

        self.conexionSerial, lineas, n)
    contadorMuestra = contadorMuestra -1

def leer_muestras_arduino(self,numMuestras,numCanales,conexionSerial,
                         lineas,n):

    # -----
    #   Función de adquisición de valores MULTICANAL
    # -----
    for i in range(numCanales):
        # --- Lectura de cada canal de datos enviado por Arduino ---
        valor = float(conexionSerial.readline().strip())
        # --- Guardado de lecturas en la última posición de datos[i] ---
        valores[i].append(valor)
        # --- Guardado de lineas de graficacion [NO SE USA] ---
        lineas[i].set_data(range(numMuestras),valores[i])
        # --- Imprimir datos recibidos --
        #print(" valores[",i,"][numMuestras]= ",valores[i][numMuestras-1])
        print(" ---> muestra: ",n," canal: ",i," valor: ",valor)

def separar_muestras_canal(self):

    paquete_muestras_barrido = 50
    datos = {} # Diccionario parcial

    for i in range(len(valores)):
        renglon = []
        for j in range(len(valores[i])):
            renglon.append(int(1000*valores[i][j])/1000)
        datos = {llaveMuestras[i]: renglon}
        # --- Actualización del Diccionario principal: "muestras"
        muestras.update(datos)

        # --- Si el numero de muestras es mayor al tamaño del paquete
        #     de muestras (50) definido en la Clase Modelo, se procede
        #     a separar las muestras de cada renglon en paquetes de
        #     50 muestras en 'arreglo'
        # -----
        if len(renglon) > paquete_muestras_barrido:
            arreglo = []
            longitud_renglon = len(renglon)
            for j in range(0,int((longitud_renglon/paquete_muestras_barrido))):
                j_min = paquete_muestras_barrido*j
                j_max = paquete_muestras_barrido*(j+1) - 1

```

```

        arreglo.append(renglon[j_min:j_max])
# --- Se catualiza nuevamente el diccionario 'muestras' ---
datos = {llaveMuestras[i]: arreglo}
muestras.update(datos)

def filtrar_muestras_obtenidas(self):

    # --- LLaves actuales del diccionario 'muestras' ---
    llaves = list(muestras)

    # --- FILTRO DE PROMEDIO CIRCULAR ---
    #   'num_muestras_promediar'
    #   'pc' -> Promedio circular
    #
    # -----
    # --- Numero de muestras a promediar en el filtro circular ---
    num_muestras_promediar = 4

    for k in range(len(llaves)):
        # --- Construccion de la nueva 'pc' para el dicionario 'muestras'
        #   pc = llaves[k] + '_pc'
        #
        # -----
        pc = llaves[k] + '_pc'
        # --- Arreglo para el guardado de paquetes de muestras del filtro circular ---
        arreglo = []

        for r in range(len(muestras[llaves[k]])):
            # --- Promediador de datos circular ---
            arreglo.append(self.promediador_datos_circular(muestras[llaves[k]][r],
                                                               num_muestras_promediar))
        # --- Actualizacion del diccionario 'muestras'
        muestras.update({pc:arreglo})

    # --- FILTRO PASA BAJAS ---
    #   'alpha' -> Peso asignado al promedio de muestras pasadas
    #   'Ip' -> Filtro Pasa Bajas
    #
    # -----
    # --- Peso asignado al promedio de muestras pasadas ---
    alpha = 0.5

    for k in range(len(llaves)):
        # --- Construccion de la nueva 'pc' para el dicionario 'muestras'
        #   Ip = llaves[k] + '_Ip'
        #
        # -----
        Ip = llaves[k] + '_Ip'

```

```

# --- Arreglo para el guardado de paquetes de muestras del filtro circular ---
arreglo = []

for r in range(len(muestras[llaves[k]])):
    # --- Promediado y actualizacion del diccionario 'muestras' ---
    arreglo.append(self.promediador_datos_lowpass(muestras[llaves[k]][r],
                                                    alpha))
    # --- Actualizacion del diccionario 'muestras'
    muestras.update({lp:arreglo})

def promediador_datos_circular(self, vector_muestras, numero_muestras_promediar):

    # --- Promediar "vector_muestras" mediante un buffer circular de
    #           "numero_muestras_promediar"
    # -----
    # --- Definicion del buffer on estructura 'FIFO' ---
    buffer = deque(maxlen = numero_muestras_promediar)
    # --- Inicializacion del buffer ---
    for n in range(numero_muestras_promediar):
        buffer.append(vector_muestras[0])

    prom = []

    for d in range(len(vector_muestras)):
        buffer.append(vector_muestras[d])
        prom.append(self.promediar_circular(buffer))
    return prom

def promediar_circular(self,buffer):

    suma = 0
    for b in list(buffer):
        suma += b
    return int(1000*suma/len(buffer))/1000

def promediador_datos_lowpass(self, vector_muestras, alpha):

    # --- Definicion del vector de salida ---
    prom = []
    # --- Definicion del valor promediado ---
    valor_promediado = vector_muestras[0]
    # --- Generacion del vector promediado ---
    for n in range(len(vector_muestras)):
```

```

        valor_promediado = alpha*valor_promediado+(1 - alpha)*vector_muestras[n]
        prom.append(int(1000*valor_promediado)/1000)
    return prom

@property
def muestras(self):
    return muestras # Diccionario

@property
def valores(self):
    return valores # Array

def imprimir_valores_recibidos(self):

    print(" ")
    print(" En CLASE Muestras::")
    print("*****")
    print("-- Muestras recibidas de Arduino por cada canal --")
    print("*****")
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k],": ",muestras[llaves[k]])

def imprimir_valores_procesados(self):

    print(" ")
    print(" En CLASE Muestras::")
    print("*****")
    print(" -- Muestras procesadas por cada canal --")
    print("*****")
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        grupo = 0
        print("")
        print("           ===           ",llaves[k],"")
        =====
        for m in muestras[llaves[k]]:
            print("")
            print(" --- ",llaves[k],"[",grupo,"] = ",m)
            grupo += 1

```

```
print(" ")
print(" En CLASE Muestras::")
print("*****")
print(" -- Muestras ACCESADAS COMO ARREGLOS --")
print("*****")
for k in range(len(llaves)):
    print("")
    print("           === ",llaves[k],"")
    =====
    for r in range(len(muestras[llaves[k]])):
        print("")
        print(" --- ",llaves[k],"[",r,"] = ",muestras[llaves[k]][r])
```

Figura 323. Código completo de la clase muestras.py

11.1.2.2.3 Clase marcadores.py (2biii)

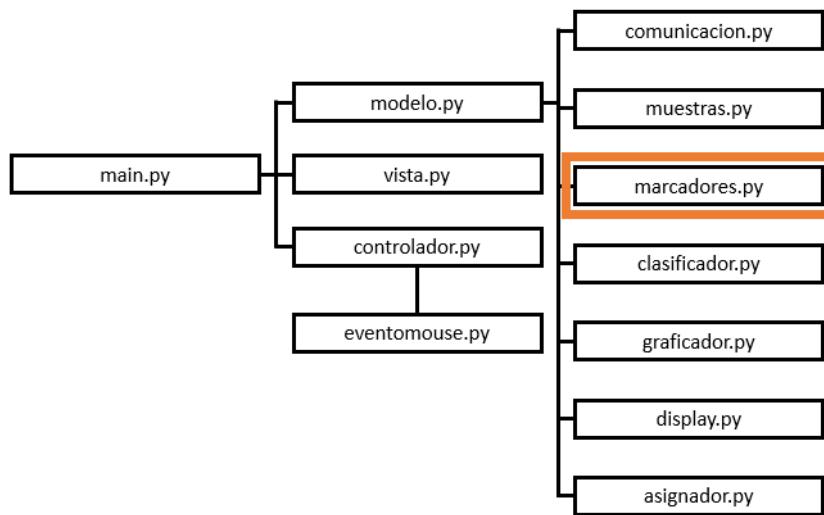


Figura 324. Ubicación de la clase marcadores.py.

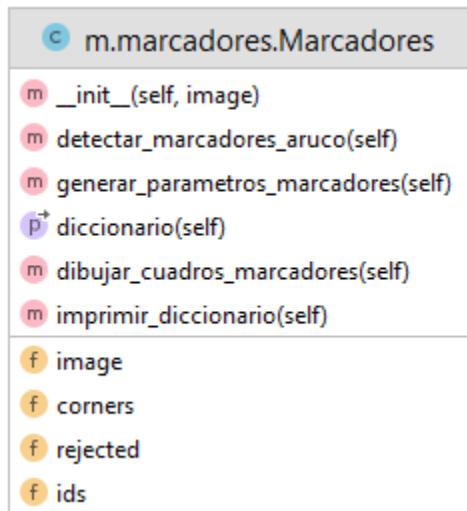


Figura 325. Métodos, atributos y propiedades de la clase marcadores.py.

```

# ****
#
# Clase: --> Muestras
# Modulo: -> muestras.py
#
# Descripción:
# Sigue el envío de valores capturados por Arduino:
#
# Fecha: junio 8/2021
#
# ****

import serial
import collections

from matplotlib.lines import Line2D
from collections import deque

muestas = {} # Diccionario de muestras: [{canal': ch, datos: []}]

class Muestras:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,conexionSerial,param_muestas):

        self.conexionSerial = conexionSerial
        self.param_muestas = param_muestas

        self.inicializar_variables_adquisicion()
        self.inicializar_arreglos_valores()
        if self.iniciar_recepcion_datos():
            # --- Leer datos procedentes de Arduino ---
            self.muestrear_datos_arduino()
            # --- Separar muestras en grupo de 'paquetes de muestras'
            #   Actualizar diccionario 'muestas'
            #
            self.separar_muestras_canal()
            # --- Filtrar muestras 'circular' 'lowpass':
            #   Actualizar diccionario 'muestas'
            #
            self.filtrar_muestras_obtenidas()

```

```

# --- Imprimir diccionario 'muestras' ---
#self.imprimir_valores_recibidos()
#self.imprimir_valores_procesados()

def inicializar_variables_adquisicion(self):

    # -----
    #     Definicion de variables de muestreo
    # -----
    global numMuestras
    global numCanales
    global llaveMuestras
    global contadorMuestra
    global valores      # Arreglo
    global lineas       # Arreglo

    numMuestras = self.param_muestras['numMuestras'] # Número de numMuestra
    numCanales = self.param_muestras['numCanales']   # Numero de canales (arduino)
    llaveMuestras = self.param_muestras['llave']      # "llave" de las muestras
    contadorMuestra = numMuestras-1 # Contador de numMuestra
    lineas = []           # Arreglo de lineas de graficación
    valores = []          # Arreglo numCanalesXnumMuestras.
                           # Almacen de valores recibidos tipo COLA

def inicializar_arreglos_valores(self):

    # --- Inicializacion con ceros "0" de los vectores de cada canal de
    #     de valores recibidos de Arduino, almacenados en el arreglo "valores[]"
    #     Cantidad de valores por canal: "numMuestra"
    #     Numero de canales: "numCanales"
    # -----
    for i in range(numCanales):
        valores.append(collections.deque([0]*numMuestras, maxlen = numMuestras))
        lineas.append(Line2D([], [], color='blue'))

def iniciar_recepcion_datos(self):

    # --- En este METODO, Python recibe los datos enviados por ARDUINO y
    #     espera hasta recibir la palabra 'inicio', la cual es la clave
    #     para comenzar a recibir datos válidos de ARDUINO
    #     Esto es así porque ARDUINO tiene un BUFFER de datos de transmisión
    #     de aproximadamente 64 bytes de longitud. Puede ser que el buffer
    #     contenga 'basura' de transmisiones anteriores y es por ésto,
    #     que este método intenta leer el contenido del buffer hasta 64

```

```

#    veces. Si después de 64 intentos no encuentra la palabra 'inicio'
#    se reportará un problema de NO-SINCRONIZACION con ARDUINO
# -----
print("")
print(" En CLASE Muestras:: ")
print("")
print("*****")
print(" --- INICIANDO SINCRONIZACION DE DATOS CON ARDUINO ---")
print("*****")
print("")
numero_maximo_intentos = 64
inmensaje_recibido_arduino = ""

contador_intentos = 0
while (inmensaje_recibido_arduino != b'inicio'):
    inmensaje_recibido_arduino = self.conexionSerial.readline().strip()
    #print(" inmensaje_recibido_arduino = ",inmensaje_recibido_arduino)
    contador_intentos += 1
    #print("    contador_intentos = ",contador_intentos)
    if (contador_intentos >= numero_maximo_intentos):
        print("")
        print(" *** NO ES POSIBLE ESTABLECER UNA COMUNICACION ***")
        print("      SINCRONIZADA CON ARDUINO: ")
        print("      SE EXCDIO EL NUMERO DE INTENTOS -> 64 <-")
        print("*****")
        return False

print("")
print("*****")
print(" --- SINCRONIZACION ETABLECIDA CON ARDUINO ---")
print("*****")
print("")
return True

def muestrear_datos_arduino(self):

    # --- Recepcion de los valores enviados por Arduino, correspondientes
    #   a cada canal. Almacenados en el arraglo "valores[]"
    #   Cantidad de valores por canal: "numMuestra"
    #   Numero de canales: "numCanales"
    # -----
    global contadorMuestra
    for n in range(numMuestras):
        self.leer_muestras_arduino(numMuestras,numCanales,

```

```

        self.conexionSerial, lineas, n)
    contadorMuestra = contadorMuestra -1

def leer_muestras_arduino(self,numMuestras,numCanales,conexionSerial,
                         lineas,n):

    # -----
    #   Función de adquisición de valores MULTICANAL
    # -----
    for i in range(numCanales):
        # --- Lectura de cada canal de datos enviado por Arduino ---
        valor = float(conexionSerial.readline().strip())
        # --- Guardado de lecturas en la última posición de datos[i] ---
        valores[i].append(valor)
        # --- Guardado de lineas de graficacion [NO SE USA] ---
        lineas[i].set_data(range(numMuestras),valores[i])
        # --- Imprimir datos recibidos --
        #print(" valores[",i,"][numMuestras]= ",valores[i][numMuestras-1])
        print(" ---> muestra: ",n," canal: ",i," valor: ",valor)

def separar_muestras_canal(self):

    paquete_muestras_barrido = 50
    datos = {} # Diccionario parcial

    for i in range(len(valores)):
        renglon = []
        for j in range(len(valores[i])):
            renglon.append(int(1000*valores[i][j])/1000)
        datos = {llaveMuestras[i]: renglon}
        # --- Actualización del Diccionario principal: "muestras"
        muestras.update(datos)

        # --- Si el numero de muestras es mayor al tamaño del paquete
        #     de muestras (50) definido en la Clase Modelo, se procede
        #     a separar las muestras de cada renglon en paquetes de
        #     50 muestras en 'arreglo'
        # -----
        if len(renglon) > paquete_muestras_barrido:
            arreglo = []
            longitud_renglon = len(renglon)
            for j in range(0,int((longitud_renglon/paquete_muestras_barrido))):
                j_min = paquete_muestras_barrido*j
                j_max = paquete_muestras_barrido*(j+1) - 1

```

```

        arreglo.append(renglon[j_min:j_max])
# --- Se actualiza nuevamente el diccionario 'muestras' ---
datos = {llaveMuestras[i]: arreglo}
muestras.update(datos)

def filtrar_muestras_obtenidas(self):

    # --- LLaves actuales del diccionario 'muestras' ---
    llaves = list(muestras)

    # --- FILTRO DE PROMEDIO CIRCULAR ---
    #   'num_muestras_promediar'
    #   'pc' -> Promedio circular
    #
    # -----
    # --- Numero de muestras a promediar en el filtro circular ---
    num_muestras_promediar = 4

    for k in range(len(llaves)):
        # --- Construccion de la nueva 'pc' para el dicionario 'muestras'
        #   pc = llaves[k] + '_pc'
        #
        # -----
        pc = llaves[k] + '_pc'
        # --- Arreglo para el guardado de paquetes de muestras del filtro circular ---
        arreglo = []

        for r in range(len(muestras[llaves[k]])):
            # --- Promediador de datos circular ---
            arreglo.append(self.promediador_datos_circular(muestras[llaves[k]][r],
                                                               num_muestras_promediar))
        # --- Actualizacion del diccionario 'muestras'
        muestras.update({pc:arreglo})

    # --- FILTRO PASA BAJAS ---
    #   'alpha' -> Peso asignado al promedio de muestras pasadas
    #   'Ip' -> Filtro Pasa Bajas
    #
    # -----
    # --- Peso asignado al promedio de muestras pasadas ---
    alpha = 0.5

    for k in range(len(llaves)):
        # --- Construccion de la nueva 'pc' para el dicionario 'muestras'
        #   Ip = llaves[k] + '_Ip'
        #
        # -----
        Ip = llaves[k] + '_Ip'

```

```

# --- Arreglo para el guardado de paquetes de muestras del filtro circular ---
arreglo = []

for r in range(len(muestras[llaves[k]])):
    # --- Promediado y actualizacion del diccionario 'muestras' ---
    arreglo.append(self.promediador_datos_lowpass(muestras[llaves[k]][r],
                                                    alpha))
    # --- Actualizacion del diccionario 'muestras'
    muestras.update({lp:arreglo})

def promediador_datos_circular(self, vector_muestras, numero_muestras_promediar):

    # --- Promediar "vector_muestras" mediante un buffer circular de
    #           "numero_muestras_promediar"
    # -----
    # --- Definicion del buffer on estructura 'FIFO' ---
    buffer = deque(maxlen = numero_muestras_promediar)
    # --- Inicializacion del buffer ---
    for n in range(numero_muestras_promediar):
        buffer.append(vector_muestras[0])

    prom = []

    for d in range(len(vector_muestras)):
        buffer.append(vector_muestras[d])
        prom.append(self.promediar_circular(buffer))
    return prom

def promediar_circular(self,buffer):

    suma = 0
    for b in list(buffer):
        suma += b
    return int(1000*suma/len(buffer))/1000

def promediador_datos_lowpass(self, vector_muestras, alpha):

    # --- Definicion del vector de salida ---
    prom = []
    # --- Definicion del valor promediado ---
    valor_promediado = vector_muestras[0]
    # --- Generacion del vector promediado ---
    for n in range(len(vector_muestras)):
```

```

        valor_promediado = alpha*valor_promediado+(1 - alpha)*vector_muestras[n]
        prom.append(int(1000*valor_promediado)/1000)
    return prom

@property
def muestras(self):
    return muestras # Diccionario

@property
def valores(self):
    return valores # Array

def imprimir_valores_recibidos(self):

    print(" ")
    print(" En CLASE Muestras::")
    print("*****")
    print("-- Muestras recibidas de Arduino por cada canal --")
    print("*****")
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k],": ",muestras[llaves[k]])

def imprimir_valores_procesados(self):

    print(" ")
    print(" En CLASE Muestras::")
    print("*****")
    print(" -- Muestras procesadas por cada canal --")
    print("*****")
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        grupo = 0
        print("")
        print("           ===           ",llaves[k],"")
        =====
        for m in muestras[llaves[k]]:
            print("")
            print(" --- ",llaves[k],"[",grupo,"] = ",m)
            grupo += 1

```

```
print(" ")
print(" En CLASE Muestras::")
print("*****")
print(" -- Muestras ACCESADAS COMO ARREGLOS --")
print("*****")
for k in range(len(llaves)):
    print("")
    print("           === ",llaves[k],"")
    =====
    for r in range(len(muestras[llaves[k]])):
        print("")
        print(" --- ",llaves[k],"[",r,"] = ",muestras[llaves[k]][r])
```

Figura 326. Código completo de la clase marcadores.py

11.1.2.2.4 clasificador.py (2biv)

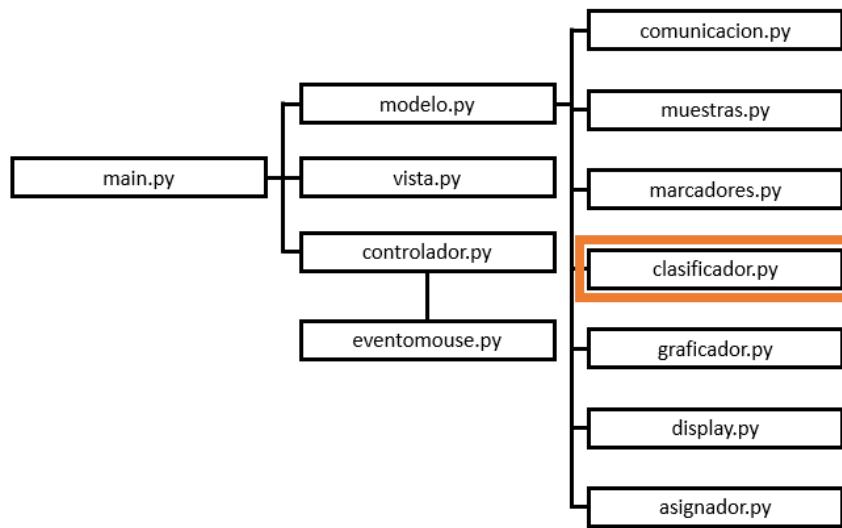


Figura 327. Ubicación de la clase clasificador.py.

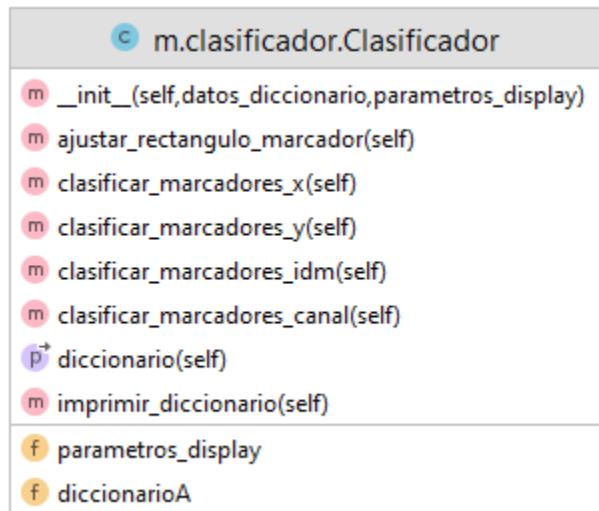


Figura 328. Métodos, atributos y propiedades de la clase clasificador.py.

```

# *****
#
# Clase: --> Clasificador
# Modulo: -> clasificador.py
#
# Descripción:
# Clasifica los marcadores de acuerdo a los siguientes criterios:
# 1) En función de su coordenada 'x' (ascendente)
# 2) En función de su coordenada 'y' (ascendente)
#
# Fecha: marzo 27/2021
#
# *****
# --- Arreglo de diccionarios asociados a cada marcador ---
diccionario = []

class Clasificador:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,datos_diccionario,parametros_display):

        global diccionario
        print("")
        print(" En CLASE Clasificador::")
        print("*****")
        print("CLASE : Clasificador: CONSTRUCTOR 1 ")

        diccionario = datos_diccionario
        self.parametros_display = parametros_display
        self.diccionarioA = {} # Información de marcadores actualizado

        self.ajustar_rectangulo_marcador()
        self.clasificar_marcadores_x()
        self.clasificar_marcadores_y()
        self.clasificar_marcadores_canal()

    def ajustar_rectangulo_marcador(self):

        *** Calculo Automático para colocar 4 DISPLAYS en linea ***
        w = self.parametros_display['ancho_imagen']
        w_4 = int(w/4)

```

```

ancho_display = w_4 - 10
self.parametros_display['ancho_display'] = ancho_display
#*****  

# --- Agregar a 'diccionario' las coordenadas (superior-izquierda)
# del 'rectangulo' que ocupará el 'display azul' sobre la imagen
for d in diccionario:
    esquina_supizq_x = int(d['centro_marcador']['cX'] -
                           self.parametros_display['ancho_display']/2)
    esquina_supizq_y = int(d['centro_marcador']['cY'] -
                           self.parametros_display['alto_display']/2)
    d['rectangulo_marcador'] = {'x': esquina_supizq_x,
                                'y': esquina_supizq_y}  

def clasificar_marcadores_x(self):
    # --- Clasificacion en función de la coordenada 'x' ascendente ---
    self.diccionarioA = sorted(diccionario, key = lambda ele: ele['x'])

    # --- Agregar a diccionario: 'posicion_x' ---
    #   'posicion_x' es la coordenada que le corresponde al marcador
    #   sobre la imagen
    # -----
    i = 0
    columna = 0
    for d in self.diccionarioA:
        # --- Calculo de las coordenadas correspondientes a los marcadores
        #   en función de si hay mas de 4 marcadores en la parte superior,
        #   entonces se colocarán en la parte inferior de la imagen
        # -----
        residuo = i%2
        if not(residuo):
            posicion_x = (self.parametros_display['ancho_display']
                          + self.parametros_display['h_gap_display'])*columna
            + self.parametros_display['offset']
            d['posicion_x'] = posicion_x
            d['posicion_renglon'] = {'x': posicion_x
                                      + self.parametros_display['h_gap_display'],
                                      'y': self.parametros_display['v_gap_display']}
        d['ventana'] = "inferior" # --- Abrir ventana hacia abajo ---
    else:
        posicion_x = (self.parametros_display['ancho_display']
                      + self.parametros_display['h_gap_display'])*columna
        + self.parametros_display['offset']

```

```

d['posicion_x'] = posicion_x
d['posicion_renglon'] = {'x': posicion_x
                         + self.parametros_display['h_gap_display'],
                         'y': self.parametros_display['alto_imagen']
                         - self.parametros_display['alto_display']
                         - self.parametros_display['v_gap_display']}
d['ventana'] = "superior" # --- Abrir ventana hacia arriba ---
if residuo:
    columna += 1
i += 1

def clasificar_marcadores_y(self):

    # --- Clasificacion en función de la coordenada 'y' ascendente ---
    self.diccionarioA = sorted(diccionario, key = lambda ele: ele['y'])

    # --- Agregar a diccionario: 'posicion_y' ---
    #   'posicion_y' es la coordenada que le corresponde al marcador
    #   sobre la imagen
    # -----
    i = 0
    for d in self.diccionarioA:
        posicion_y = (self.parametros_display['alto_display']
                      + self.parametros_display['v_gap_display'])*i
        + self.parametros_display['offset']
        d['posicion_y'] = posicion_y
        d['posicion_columna'] = {'x': self.parametros_display['offset_x'],
                                'y': posicion_y}
        i += 1

def clasificar_marcadores_idm(self):

    # --- Clasificacion en función de la coordenada 'idm' ascendente ---
    global diccionario
    diccionario = sorted(diccionario, key = lambda ele: ele['idm'])

def clasificar_marcadores_canal(self):

    # --- Clasificacion en función del canal de datos 'id' ascendente ---
    global diccionario
    diccionario = sorted(diccionario, key = lambda ele: ele['id'])

@property
def diccionario(self):

```

```
    return diccionario

def imprimir_diccionario(self):

    print(" ")
    print(" En CLASE Clasificador::")
    print("*****")
    print(" Imprimir diccionario --> ")
    print("     Información del Diccionario ")
    print("*****")
    for d in diccionario:
        print("")
        print(" --- id: ",d['id'], " ---")
        print(d)
```

Figura 329. Código completo de la clase clasificador.py

11.1.2.2.5 graficador.py (2bv)

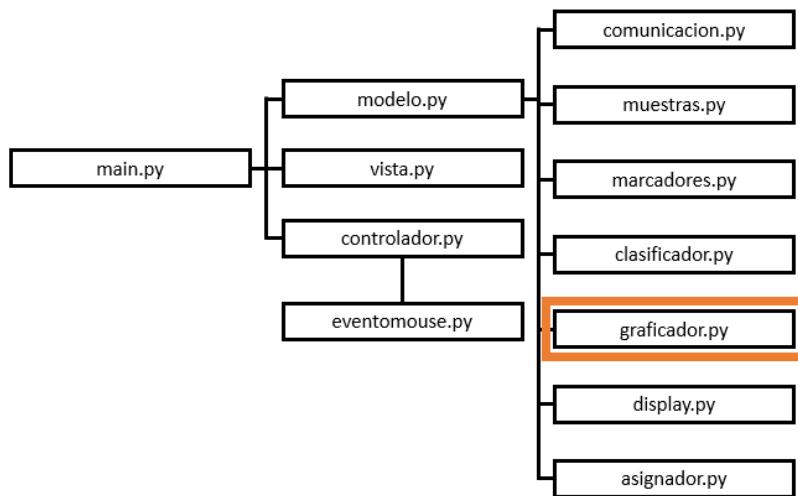


Figura 330. Ubicación de la clase graficador.py.

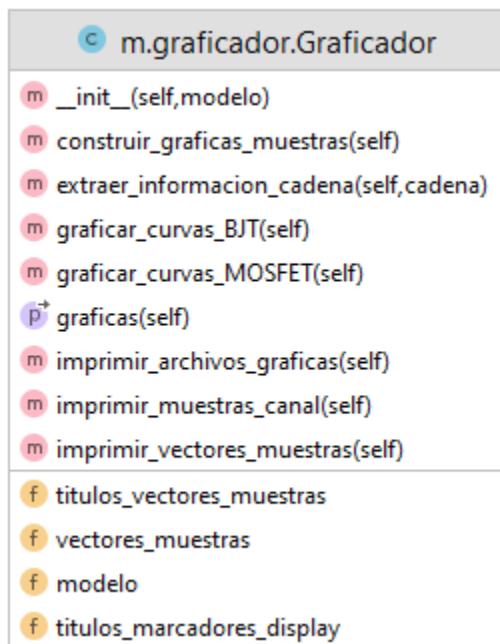


Figura 331. Métodos, atributos y propiedades de la clase graficador.py.

```

# ****
#
# Clase: --> Graficador
# Modulo: -> graficador.py
#
# Descripción:
# - Graficar los datos obtenidos por arduino
# - Dibuar, mostrar informacion
#
# Fecha: abril 13/2021
#
# ****

import numpy as np
import cv2

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

from collections import deque

graficas = []

class Graficador():

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,modelo):

        print("")
        print(" CONSTRUCTOR: Clase: Graficador")
        self.modelo = modelo
        self.vectores_muestras = modelo.vectores_muestras
        self.titulos_vectores_muestras = modelo.titulos_vectores_muestras
        self.titulos_marcadores_display = modelo.titulos_marcadores_display

    def construir_graficas_muestras(self):

        # --- Construccion AUTOMATICA de la graficas ---
        titulos_marcadores = list(self.titulos_marcadores_display)

        global graficas
        graficas = []

```

```

muestras = self.vectores_muestras

for i in range(len(titulos_marcadores)):
    titulo_grafica = self.titulos_marcadores_display[titulos_marcadores[i]]
    nombre_vectores_grafica = self.extraer_informacion_cadena(titulo_grafica)

    # --- Construcción AUTOMATICA de cada GRAFICA, a partir de la
    #   información proporcionada en la clase Modelo en las estructuras:
    #   -> titulos_vectores_muestras <- Vector
    #   -> titulos_marcadores_display <- Diccionario
    #   Estas estructuras de datos se conectan con los nombres de los
    #   vectores que se encuentran en el Diccionario muestras
    #   -> vectores_muestras <- Diccionario
    #
    # -----
    plt.figure(i)
    # --- Graficas VECTOR A vs. muestras ---
    if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 1:
        plt.plot(muestras[nombre_vectores_grafica[0][0]],
                  label = nombre_vectores_grafica[0][0] + ' vs. muestras')
    # --- Graficas VECTOR A vs VECTOR B ---
    if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 2:
        plt.plot(muestras[nombre_vectores_grafica[0][0]],
                  muestras[nombre_vectores_grafica[0][1]],
                  label = nombre_vectores_grafica[0][0] + ' vs. ' +
                  nombre_vectores_grafica[0][1])
    # --- Graficas MULTIPLES tipo: VECTOR A vs VECTOR B ---
    if len(nombre_vectores_grafica) > 2:
        for g in range(len(nombre_vectores_grafica)):
            plt.plot(muestras[nombre_vectores_grafica[g][0]],
                      muestras[nombre_vectores_grafica[g][1]],
                      label = nombre_vectores_grafica[g][0] + ' vs. ' +
                      nombre_vectores_grafica[g][1])
    plt.title = 'Grafica ' + str(i)
    plt.ylabel = 'y label'
    plt.xlabel = 'x label'
    plt.legend()

    # --- Guardado de las gráficas en:
    #   1. "Archivo en disco [*.jpg] para su uso posterior
    #       por parte de la Clase Display
    #   2. El arreglo GLOBAL: 'graficas', el cual contiene
    #       la ruta en donde se guardó la gráfica correspondiente
    #

```

```

grafica = 'g/grafica_' + str(i) + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

def extraer_informacion_cadena(self,cadena):

    #print("")
    #print(" cadena original = ",cadena)

    # --- Separa grupos de letras ---
    subcadena = ""
    subtulos = []
    ignorar_caracter = False
    num_char = 0
    for ch in cadena:
        num_char += 1
        # --- Ignorar los caracteres "espacio" [ ' ] y "coma" [ , ] ---
        if ch == ' ' or ch == ',':
            ignorar_caracter = True
        if num_char != 1:
            subtulos.append(subcadena)
            subcadena = ""
        else:
            subcadena += ch
        if num_char == len(cadena): subtulos.append(subcadena)

    # --- Genera tuplas de dos elementos ---
    tupla = []
    arreglo_tuplas = []
    for i in range(len(subtulos)):
        # --- Detecta conector [vs] ó espacio [ ' ] --
        if subtulos[i] == 'vs' or subtulos[i] == " ":
            dummy = 0 # <--- No hacer nada ---
        else:
            tupla.append(subtulos[i])
        if subtulos[i] == " " or i == (len(subtulos)-1):
            arreglo_tuplas.append(tupla)
            tupla = []

    return arreglo_tuplas

def graficar_curvas_BJT(self):

    global graficas

```

```

graficas = []

muestras = self.vectores_muestras

# --- Grafica 0 : Ic vs Vce ---
plt.figure(0)
plt.plot(muestras['Vbe'][5], muestras['Ic'][5], label='Ic vs Vbe')
#plt.ylim(0,5)
plt.xlim(0,1) # <- BJT
#plt.xlim(0,3) # <- MOSFET
plt.xlabel('Vbe (V)')
plt.ylabel('Ic (mA)')
plt.title("Ic vs Vbe")
plt.legend()
grafica = 'g/grafica_0' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 1: Ib vs Vbe ---
plt.figure(1)
plt.plot(muestras['Vbe'][5], muestras['Ib'][5], label='Ib vs Vbe')
#plt.ylim(0,20)
plt.xlim(0,1)
plt.xlabel('Vbe (V)')
plt.ylabel('Ib (uA)')
plt.title("Ib vs Vbe")
plt.legend()
grafica = 'g/grafica_1' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 2: Ic vs Vce (Cinco curvas sin promediar) ---
plt.figure(2)
for i in range(len(muestras['Ic'])-1):
    label_Vbe=str((int(np.average(muestras['Vbe'][i])*1000))/1000)
    label_Ib=str((int(np.average(muestras['Ib'][i])*1000))/1000)
    plt.plot(muestras['Vce'][i],muestas['Ic'][i],
             label='Vbe=' + label_Vbe + ', Ib =' + label_Ib + ' uA')

    plt.xlabel('Vce (V)')
    plt.ylabel('Ic (mA)')
    plt.title("Ic vs Vce")
    plt.legend()

```

```

grafica = 'g/grafica_2' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 3 : Ic vs Vbe [PROMEDIADO - CIRCULAR] ---
plt.figure(3)
plt.plot(muestras['Vbe_pc'][5], muestras['Ic_pc'][5], label='Ic vs Vbe')
plt.xlim(0,1) # <- BJT
#plt.xlim(0,3) # <- MOSFET
plt.xlabel('Vbe (V)')
plt.ylabel('Ic (mA)')
plt.title("Ic vs Vbe [pc]")
plt.legend()
grafica = 'g/grafica_3' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 4: Ib vs Vbe [PROMEDIADO - CIRCULAR] ---
plt.figure(4)
plt.plot(muestras['Vbe_pc'][5], muestras['Ib_pc'][5], label='Ib vs Vbe')
plt.xlim(0,1)
plt.xlabel('Vbe (V)')
plt.ylabel('Ib (uA)')
plt.title("Ib vs Vbe [pc]")
plt.legend()
grafica = 'g/grafica_4' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 5: Ic vs Vce (5 curvas)[PROMEDIADO - CIRCULAR] ---
plt.figure(5)
for i in range(len(muestras['Ic_pc'])-1):
    label_Vbe=str((round(np.average(muestras['Vbe_pc'][i])*1000))/1000)
    label_Ib=str((round(np.average(muestras['Ib_pc'][i])*1000))/1000)
    plt.plot(muestras['Vce_pc'][i],muestas['Ic_pc'][i],
             label='Vbe=' +label_Vbe+', Ib =' +label_Ib+' uA')
    plt.xlabel('Vce (V)')
    plt.ylabel('Ic (mA)')
    plt.title("Ic vs Vce [pc]")
    plt.legend()
    grafica = 'g/grafica_5' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

```

```

# --- Grafica 6: Ic vs Vce (5 curvas)[PROMEDIADO - LowPass - 8] ---
plt.figure(6)
for i in range(len(muestras['Ic_lp'])-1):
    label_Vbe=str((round(np.average(muestras['Vbe_lp'][i])*1000))/1000)
    label_Ib=str((round(np.average(muestras['Ib_lp'][i])*1000))/1000)
    plt.plot(muestras['Vce_lp'][i],muestras['Ic_lp'][i],
              label='Vbe=' +label_Vbe+'v, Ib =' +label_Ib+' uA')
    plt.xlabel('Vce (V)')
    plt.ylabel('Ic (mA)')
    plt.title("Ic vs Vce [lp]")
    plt.legend()
    grafica = 'g/grafica_6' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

# --- Grafica 7: Ic vs Vbe [PROMEDIADO - LowPass] ---
plt.figure(7)
for i in range(len(muestras['Ic'])-1):
    label_Vbe=str((round(np.average(muestras['Vbe'][i])*1000))/1000)
    label_Ib=str((round(np.average(muestras['Ib'][i])*1000))/1000)
    plt.plot(muestras['Vce'][i],muestras['Ic'][i],
              label='Vbe=' +label_Vbe+'v, Ib =' +label_Ib+' uA')
    label_Vbe=str((round(np.average(muestras['Vbe_pc'][i])*1000))/1000)
    label_Ib=str((round(np.average(muestras['Ib_pc'][i])*1000))/1000)
    plt.plot(muestras['Vce_pc'][i],muestras['Ic_pc'][i],
              label='pVbe=' +label_Vbe+'v, Ib =' +label_Ib+' uA')
    plt.xlabel('Vce (V)')
    plt.ylabel('Ic (mA)')
    plt.title("Ic vs Vce [m]")
    plt.legend()
    grafica = 'g/grafica_7' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

# --- Auxiliar para depuracion ---
#plt.show()

def graficar_curvas_MOSFET(self):
    global graficas
    graficas = []

    muestras = self.vectores_muestras

```

```

# --- Grafica 0 : Id vs Vds ---
plt.figure(0)
plt.plot(muestras['Vgs'][5], muestras['Id'][5], label='Id vs Vds')
#plt.ylim(0,5)
plt.xlim(0,1) # <- BJT
#plt.xlim(0,3) # <- MOSFET
plt.xlabel('Vgs (V)')
plt.ylabel('Id (mA)')
plt.title("Id vs Vds")
plt.legend()
grafica = 'g/grafica_0' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 1: Ib vs Vbe ---
plt.figure(1)
plt.plot(muestras['Vgs'][5], muestras['Ig'][5], label='Ig vs Vgs')
#plt.ylim(0,20)
plt.xlim(0,1)
plt.xlabel('Vgs (V)')
plt.ylabel('Ib (uA)')
plt.title('Ig vs Vgs')
plt.legend()
grafica = 'g/grafica_1' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 2: Ic vs Vce (Cinco curvas sin promediar) ---
plt.figure(2)
for i in range(len(muestras['Id'])-1):
    label_Vgs=str((int(np.average(muestras['Vgs'][i])*1000))/1000)
    plt.plot(muestras['Vds'][i],muestas['Id'][i],
             label='Vgs=' + label_Vgs + 'v')

    plt.xlabel('Vds (V)')
    plt.ylabel('Id (mA)')
    plt.title('Id vs Vds')
    plt.legend()
    grafica = 'g/grafica_2' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

# --- Grafica 3 : Id vs Vds [PROMEDIADO - CIRCULAR] ---

```

```

plt.figure(3)
plt.plot(muestras['Vgs_pc'][5], muestras['Id_pc'][5], label='Id vs Vds')
plt.xlim(0,1) # <- BJT
#plt.xlim(0,3) # <- MOSFET
plt.xlabel('Vgs (V)')
plt.ylabel('Id (mA)')
plt.title("Id vs Vds [pc]")
plt.legend()
grafica = 'g/grafica_3' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 4: Ib vs Vbe [PROMEDIADO - CIRCULAR] ---
plt.figure(4)
plt.plot(muestras['Vgs_pc'][5], muestras['Ig_pc'][5], label='Ig vs Vgs')
plt.xlim(0,1)
plt.xlabel('Vgs (V)')
plt.ylabel('Ig (uA)')
plt.title("Ib vs Vbe [pc]")
plt.legend()
grafica = 'g/grafica_4' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 5: Ic vs Vce (5 curvas)[PROMEDIADO - CIRCULAR] ---
plt.figure(5)
for i in range(len(muestras['Id_pc'])-1):
    label_Vgs=str((round(np.average(muestras['Vgs_pc'][i])*1000))/1000)
    plt.plot(muestras['Vds_pc'][i],muestas['Id_pc'][i],
             label='Vgs=' + label_Vgs + 'v')
    plt.xlabel('Vds (V)')
    plt.ylabel('Id (mA)')
    plt.title('Id vs Vds [pc]')
    plt.legend()
    grafica = 'g/grafica_5' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

# --- Grafica 6: Ic vs Vce (5 curvas)[PROMEDIADO - LowPass - 8] ---
plt.figure(6)
for i in range(len(muestras['Ic_lp'])-1):
    label_Vgs=str((round(np.average(muestras['Vgs_lp'][i])*1000))/1000)
    plt.plot(muestras['Vds_lp'][i],muestas['Ic_lp'][i],
             label='Vgs=' + label_Vgs + 'v,')


```

```

plt.xlabel('Vds (V)')
plt.ylabel('Id (mA)')
plt.title('Id vs Vds [Ip]')
plt.legend()
grafica = 'g/grafica_6' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 7: Id vs Vds [PROMEDIADO - LowPass] ---
plt.figure(7)
for i in range(len(muestras['Id'])-1):
    label_Vgs=str((round(np.average(muestras['Vgs'][i])*1000))/1000)
    plt.plot(muestras['Vds'][i],muestras['Id'][i],
              label='Vgs=' + label_Vgs+'v')
    label_Vgs=str((round(np.average(muestras['Vgs_pc'][i])*1000))/1000)
    plt.plot(muestras['Vce_pc'][i],muestras['Id_pc'][i],
              label='pVbe=' + label_Vgs+'v')
plt.xlabel('Vds (V)')
plt.ylabel('Id (mA)')
plt.title("Id vs Vds [m]")
plt.legend()
grafica = 'g/grafica_7' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Auxiliar para depuracion ---
#plt.show()

@property
def graficas(self):
    return graficas # --- Nombre de la grafica y ruta donde se guarda ---

def imprimir_archivos_graficas(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" --- Archivos de imagenes de graficas *.jpg ---")
    print("*****")
    for g in graficas:
        print("")
        print(g)

def imprimir_muestras_canal(self):

```

```

print(" ")
print(" En CLASE Graficador:")
print("*****")
print(" --- Vectores de muestras por canal ---")
print("*****")
# --- ESTRUCTURA del diccionario:
#   datos = {canal: i, llaveMuestras[i]: vector de datos}
# -----
muestras = self.vectores_muestras
llaves = list(muestras)
print(" llaves = ",llaves)
for k in range(len(llaves)):
    print("")
    print(" --- ",llaves[k],": ",muestras[llaves[k]])

def imprimir_vectores_muestras(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" -- Muestras leidas de la Clase Modelo --")
    print("*****")
    muestras = self.vectores_muestras
    llaves = self.titulos_vectores_muestras
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k],": ",muestras[llaves[k]])

```

Figura 332. Código completo de la clase graficador.py

11.1.2.2.6 *display.py* (2bvi)

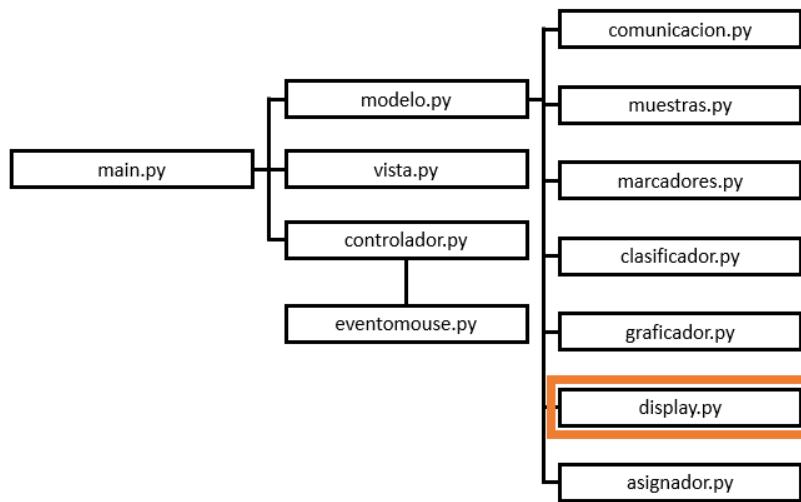


Figura 333. Ubicación de la clase *display.py*.

```
m __init__(self, modelo, image, datos_diccionario, titulos_marcadores, parametros_display)
m centrar_display_marcador(self)
m inicializar_parametros(self)
m asignar_imagen_sobrepuesta(self,img)
m posicionar_ventana_mouse(self, mouseX, mouseY)
m mouse_sobre_controlesVentana(self,mouse_x,mouse_y)
m actualizar_imagen_display(self,img)
m dibujar_circuloLinea_marcador(self,overlay)
m crear_display_principal(self,overlay)
m adjuntar_ventana_graficacion(self,overlay)
m adjuntar_ventana_graficacionMAXIMA(self,overlay)
m programar_secuencia_inicial(self)
m cambiar_aposicion_SuperiorInferior(self)
m cambiar_aposicion_marcador(self)
m move_springing_posicion(self)
m spring(self,Tx,x,Ty,y)
m check_screen_bounds(self)
m print_diccionario_display(self)
```

Figura 334. Métodos, atributos y propiedades de la clase display.py.

```

# ****
#
# Clase: --> Display
# Modulo: -> display.py
#
# Descripción:
# Genera los cuadros de display de texto y sus
# areas de graficacion respectiva
#
# Fecha: marzo 27/2021
#
# ****

import cv2
import time
import imutils

# -----
# Propiedades de la clase Display
# -----


# self.diccionario <- 'diccionario' del Arreglo de "dicionarios por cada marcador"

# --- Estructura del diccionario -----
#
# Cada MARCADOR tiene asociado el siguiente 'diccionario' (ejemplo numerico)
# {'id': 10,
# 'id': 20,
# 'centro_marcador': {'cX': 382, 'cY': 354},
# 'x': 382, 'y': 354,
# 'cuadrado_marcador': {'topRight': (369, 341), 'bottomRight': (397, 341),
#                         'bottomLeft': (396, 369), 'topLeft': (368, 368)},
# 'rectangulo_marcador': {'x': 777, 'y': 165},
# 'posicion_x': 0,
# 'posicion_renglon': {'x': 5, 'y': 528},
# 'ventana': 'superior',
# 'posicion_y': 140,
# 'posicion_columna': {'x': 65, 'y': 140}}
# -----


x_inicial = 5
y_inicial = 5

easing = 0.05 # 0.2

```

```

spring = 0.1
friction = 0.7 #0.95

x_velocity = 0
y_velocity = 0

class Display:

    def __init__(self, modelo, image, datos_diccionario,
                 titulos_marcadores, parametros_display):

        self.modelo = modelo

        self.diccionario = datos_diccionario

        self.image = image
        h, w, channels = image.shape
        self.screen_width = w
        self.screen_height = h

        self.titulos_marcadores = titulos_marcadores
        self.pd = parametros_display

        self.x = self.diccionario['x']
        self.y = self.diccionario['y']
        self.centrar_display_marcador()
        self.inicializar_parametros()

    def centrar_display_marcador(self):
        self.esquina_x = int(self.diccionario['x'] - self.pd['ancho_display']/2)
        self.esquina_y = int(self.diccionario['y'] - self.pd['alto_display']/2)

    def inicializar_parametros(self):

        self.flag_abrirVentanaDisplay = False
        self.flag_abrirVentanaDisplay_MAXIMO = False

        self.x = 5
        self.y = 5
        self.ancho_display = self.pd['ancho_display']
        self.alto_display = self.pd['alto_display']
        self.targetX = self.esquina_x # spring -> target 'x'
        self.targetY = self.esquina_y # spring -> target 'y'
        self.x_velocity = 0 # spring -> velocidad 'x'

```

```

        self.y_velocity = 0      # spring -> velocidad 'y'
        self.color = (255,0,0)   # Azul oscuro

        self.secuenciador = 0    # Secuenciador
        self.flag = 1            # Secuenciador
        self.switch_inicio = 1   # Secuenciador

        self.permitir_ocupar_arealmagen = True # Manejo del Area de Imagen para:
        self.permitir_abrir_ventana = True # Manejo de ventanas de informacion
        self.posicion_estable = True      # Manejo de ventanas de informacion

def asignar_imagen_sobrepuesta(self,img):

    self.imagen = cv2.imread(img)
    self.h0, self.w0 = self.imagen.shape[:2]
    #
    # -----#
    # img2 --> Imagen superpuesta a mostrar en tamaño DISPLAY
    # -----#
    self.img2 = imutils.resize(cv2.imread(img),self.pd['ancho_display'])
    h2, w2 = self.img2.shape[:2]
    self.himg2 = h2
    self.wimg2 = w2

    #
    # -----#
    # img3 --> Imagen superpuesta a mostrar en tamaño MAXIMO
    # -----#
    ancho_maximo = self.screen_width - 20
    self.img3 = imutils.resize(cv2.imread(img),ancho_maximo)
    h3, w3 = self.img3.shape[:2]
    self.himg3 = h3
    self.wimg3 = w3

def posicionar_ventana_mouse(self, mouseX, mouseY):
    # --- Se activa desde la clase Controlador -> clase Evento Mouse ---
    #   metodo: onMouse(self, event, x, y, flags, param)
    #

    # --- Referencia de marcadores -----
    # Se define un cuadro de 20x20 alrededor del centro del marcador
    # como un 'area' detectable para indicar que el 'mouse' se encuentra
    # sobre el 'marcador'
    #
    lat_izq = self.diccionario['x']-10
    lat_der = self.diccionario['x']+10

```

```

inf = self.diccionario['y']-10
sup = self.diccionario['y']+10

# --- Posicionar 'ventana' en RENGLON superior/inferior -----
if mouseY <= 20:
    self.permitir_abrir_ventana = True
    self.targetX = self.diccionario['posicion_renglon']['x']
    self.targetY = self.diccionario['posicion_renglon']['y']
# -----


# --- Posicionar 'ventana' en COLUMNA izquierda -----
elif mouseX <= 50:
    self.permitir_abrir_ventana = False
    self.targetX = self.diccionario['posicion_columna']['x']
    self.targetY = self.diccionario['posicion_columna']['y']


# --- Posicionar 'ventana' SOBRE marcadores: MOUSE sobre el 'marcador' ---
elif mouseX>=lat_izq and mouseX<=lat_der and mouseY>=inf and mouseY<=sup:
    self.permitir_abrir_ventana = False
    self.targetY = self.diccionario['rectangulo_marcador']['y']
    self.targetX = self.diccionario['rectangulo_marcador']['x']
    print(" ")
    print(" En CLASE Display::")
    print(" -> CURSOR sobre el MARCADOR",self.diccionario['id'])
# -----


self.move_springing_posicion()

def mouse_sobre_controlesVentana(self,mouse_x,mouse_y):
    # --- Se activa desde la clase Controlador -> clase Evento Mouse ---
    #   metodo: onMouse(self, event, x, y, flags, param)
    # -----


    # --- Referencia de controles de ventana Max/Min -----
    # Se define una zona de 25x30 (ancho=25, alto=30 en la parte derecha
    # de la 'ventana', para detectar la posicion del 'mouse' en dicha zona
    # -----


    sup_der = self.x + self.ancho_display
    sup_izq = self.x + self.ancho_display - 25
    inf_izq = self.y
    inf_der = self.y+self.alto_display


    # --- Referencia de controles de ventana Maximizar TOTAL -----
    # Se define una zona de 25x30 (ancho=25, alto=30 en la parte derecha-10

```

```

# de la 'ventana', para detectar la posicion del 'mouse' en dicha zona
# -----
max_s_d = self.x + self.ancho_display - 35 # superior_derecha
max_s_i = self.x + self.ancho_display - 60 # supeior_izquierda
max_i_i = self.y # inferior_izquierda
max_i_d = self.y+self.alto_display # inferior_derecha

# --- MOUSE sobre los controles Max/Min en la 'ventana' ---
if mouse_x>=sup_izq and mouse_x<=sup_der and mouse_y>=inf_izq and
mouse_y<=inf_der:
    # --- La ventana de graficacion se ABRE ---
    if self.flag_abrirVentanaDisplay:
        self.flag_abrirVentanaDisplay = False
    # --- La ventana de graficacion se CIERRA ---
    else:
        self.flag_abrirVentanaDisplay = True
    print(" ")
    print(" En CLASE Display::")
    print(" -> CURSOR sobre control Max/min -> MARCADOR",self.diccionario['id'])

# --- MOUSE sobre el control Maximizar TOTAL en la 'ventana' ---
if mouse_x>=max_s_i and mouse_x<=max_s_d and mouse_y>=max_i_i and
mouse_y<=max_i_d:
    # --- La ventana de graficacion se ABRE ---
    if self.flag_abrirVentanaDisplay_MAXIMO:
        self.flag_abrirVentanaDisplay_MAXIMO = False
    # --- La ventana de graficacion se CIERRA ---
    else:
        self.flag_abrirVentanaDisplay_MAXIMO = True
    print(" ")
    print(" En CLASE Display::")
    print(" -> CURSOR sobre Maximizar TOTAL -> MARCADOR",self.diccionario['id'])

def actualizar_imagen_display(self,img):

    # --- Imagen para superposicion y transparencia ---
    overlay = img.copy()

    # -----
    # Dibujar "circulo" sobre el marcador
    # Dibujar "linea" que conecta el marcador con el DISPLAY principal
    # -----
    self.dibujar_circuloLinea_marcador(overlay)

```

```

# -----
#     Creacion del DISPLAY PRINCIPAL (DISPLAY)
# -----
self.crear_display_principal(overlay)

# -----
#     Creacion de la VENTANA SECUNDARIA
#     [Graficacion o diccionario] --> (DISPLAY)
# -----


# --- ABRIR Ventana de INFORMACION (Grafica o Texto/Dibujo) ---
if      self.flag_abrirVentanaDisplay      and      self.permitir_abrir_ventana      and
self.posicion_estable:

    # --- Adjuntar Ventana de GRAFICACION ---
    self.adjuntar_ventana_graficacion(overlay)
    # --- Dibujar control de minimizar ventana graficacion ---
    cv2.rectangle(overlay, (self.x + self.ancho_display - 20,
                           self.y + 18),
                  (self.x + self.ancho_display - 8,
                   self.y + 20),
                  (255,255,255), 1)

# -----
#     INICIO DE CONTROLES DE MAXIMIZACION / MINIMIZACION MAXIMA
# -----
if self.permitir_ocupar_arealmagen and self.flag_abrirVentanaDisplay_MAXIMO:
    #
    #     Se solicita permiso a la clase Modelo para usar el area de
    #     imagen. La clase Modelo quita el permiso a todas las demás
    #     ventanas para que no usen el area de imagen
    #     metodo: solicitud_permiso_usoArealmagen(self, canal)
    #             - permitir_abrir_ventana = False
    #             - permitir_ocupar_arealmagen = False
    #     Para la ventana solicitantes:
    #             - permitir_ocupar_arealmagen = True
    #
    self.modelo.solicitud_permiso_usoArealmagen(self.diccionario['id'])
    # --- Adjuntar Ventana de GRAFICACION MAXIMA ---
    self.adjuntar_ventana_graficacionMAXIMA(overlay)
    # --- Dibujar control de Minimizar TOTAL ventana graficacion ---
    cv2.rectangle(overlay, (self.x + self.ancho_display - 35,
                           self.y + 18),
                  (self.x + self.ancho_display - 60,

```

```

        self.y + 20),
        (0,255,0), 1)

else:
# -----
#   Se solicita permiso a la clase Modelo para LIBERAR el area de
#   imagen. La clase Modelo OTORGA el permiso a todas las demás
#   ventanas para que puedan usar el area de imagen
#   metodo: solicitud_liberar_usoArealImagen(self)
#           - permitir_abrir_ventana = True
#           - permitir_ocupar_arealmagen = True
# -----
self.modelo.solicitud_liberar_usoArealImagen()
# --- Dibujar control de Maximizar TOTAL ventana graficacion ---
cv2.rectangle(overlay, (self.x + self.ancho_display - 35,
                      self.y + 8),
              (self.x + self.ancho_display - 60,
               self.y + 20),
              (0,255,0), 1)
#
#   FIN DE CONTROLES DE MAXIMIZACION / MINIMIZACION MAXIMA
# -----


# --- CERRAR Ventana de INFORMACION (Graficacion o diccionario) ---
elif self.permitir_abrir_ventana and self.posicion_estable:
    # --- Dibujar control de maximizar ventana graficacion --
    cv2.rectangle(overlay, (self.x + self.ancho_display - 20,
                           self.y + 8),
                  (self.x + self.ancho_display - 8,
                   self.y + 20),
                  (255,255,255), 1)

#
#   Superposición de las dos imagenes con transparencia
# -----


alpha = 0.8 # 0.6 el mejor
cv2.addWeighted(overlay,alpha,img,1-alpha,0,img)

def dibujar_circuloLinea_marcador(self,overlay):

# --- Linea de conexion entre "marcadores" y Ventana de diccionario [Display]
cv2.line(overlay,(self.x ,self.y),
         (self.diccionario['x'],self.diccionario['y']),
         (255, 255, 255), 1)

```

```

# --- Circunferencia sobre "marcadores"
cv2.circle(overlay, (self.diccionario['x'],self.diccionario['y']),
           10, (255,255,255), 2)
#print("- D:",self.diccionario['id']," self.x = ",self.x," self.y = ",self.y)

def crear_display_principal(self,overlay):

    # --- MARCO del DISPLAY [fondo azul] ---
    cv2.rectangle(overlay, (self.x,self.y),
                  (self.x + self.ancho_display,
                   self.y + self.alto_display),
                  self.color, -1)

    # --- TEXTO/diccionario DISPLAY [letras/digitos blancos]
    #cv2.putText(overlay, "CH{} {}".format(self.diccionario['id'],self.diccionario['id']))
    # --- IDENTIFICADOR del DISPLAY ---
    m = "m" + str(self.diccionario['id'])
    titulo = self.titulos_marcadores[m]
    cv2.putText(overlay, "M{}".format(self.diccionario['id']),
                (self.x + 15,
                 self.y + 19),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5, (255, 255, 255), 2, cv2.LINE_AA)

    # --- TITULO del DISPLAY ---
    cv2.putText(overlay, titulo,
                (self.x + 50,
                 self.y + 19),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5, (255, 255, 255), 2, cv2.LINE_AA)

def adjuntar_ventana_graficacion(self,overlay):

    # --- Pantalla GRAFICA: grafica de "matplotlib" ---
    if self.diccionario['ventana'] == "inferior":
        pip_h = self.y + self.alto_display
        pip_w = self.x
        h1, w1 = self.img2.shape[:2]
        overlay[pip_h:pip_h+h1,pip_w:pip_w+w1] = self.img2 # make it PIP
        # --- Marco de pantalla grafica ---
        cv2.rectangle(overlay, (self.x,self.y + self.alto_display),
                      (self.x + w1, self.y + self.alto_display + h1),
                      self.color, 1)

```

```

        elif self.diccionario['ventana'] == "superior":
            pip_h = self.y
            pip_w = self.x
            h1, w1 = self.img2.shape[:2]
            overlay[pip_h-h1:pip_h,pip_w:pip_w+w1] = self.img2 # make it PIP
            # --- Marco de pantalla grafica ---
            cv2.rectangle(overlay, (self.x,self.y),
                          (self.x + w1, self.y - h1),
                          self.color, 1)

    def adjuntar_ventana_graficacionMAXIMA(self,overlay):
        pip_h = int((self.screen_height - self.h0)/2)
        pip_w = int((self.screen_width - self.w0)/2)
        h0 = self.h0
        w0 = self.w0

        overlay[pip_h:pip_h+h0,pip_w:pip_w+w0] = self.imagen # make it PIP
        # --- Marco de pantalla grafica ---
        cv2.rectangle(overlay, (pip_w, pip_h), (pip_w+w0, pip_h+h0), self.color, 1)

    def programar_secuencia_inicial(self):
        # --- Se activa desde la clase Controlador -> clase Evento Mouse ---
        #   metodo: mover_objetos(self,imagen)
        #

        if self.secuenciador <= 30: # 150
            if self.flag == 1:
                self.secuenciador += 1
            else:
                self.secuenciador -= 1

        if self.secuenciador == 30: # 150
            self.flag = 0
            self.cambiar_aposicion_SuperiorInferior()
            self.switch_inicio = 0
        if self.secuenciador == 0:
            self.flag = 1
            self.cambiar_aposicion_marcador()

    def cambiar_aposicion_SuperiorInferior(self):
        self.targetX = self.diccionario['posicion_renglon']['x']
        self.targetY = self.diccionario['posicion_renglon']['y']

```

```

def cambiar_aposicion_marcador(self):
    self.targetX = self.diccionario['rectagulo_marcador']['x']
    self.targetY = self.diccionario['rectagulo_marcador']['y']

def move_springing_posicion(self):
    self.x, self.y = self.spring(self.targetX, self.x, self.targetY, self.y)

def spring(self, Tx, x, Ty, y):
    self.check_screen_bounds()

    dx = Tx - x
    dy = Ty - y

    if dx > 0 and dx <= 4: x = Tx
    if dy > 0 and dy <= 4: y = Ty

    if x == Tx and y == Ty: self.posicion_estable = True
    else: self.posicion_estable = False

    ax = dx * spring
    ay = dy * spring

    self.x_velocity += ax
    self.y_velocity += ay

    self.x_velocity *= friction
    self.y_velocity *= friction

    x += self.x_velocity
    y += self.y_velocity

    x = int(x)
    y = int(y)

    return (x, y)

def check_screen_bounds(self):

    if self.x < 5: self.x_velocity = -self.x_velocity
    if self.y < 0: self.y_velocity = -self.y_velocity
    if (self.x + self.ancho_display) > self.screen_width - 20:
        self.x_velocity = -self.x_velocity

```

```
if (self.y + self.alto_display) > self.screen_height:  
    self.y_velocity = -self.y_velocity  
  
def print_diccionario_display(self):  
    print(" ")  
    print(" En CLASE Display::")  
    print("*****")  
    print("-- Muestras recibidas de Arduino por cada canal --")  
    print("*****")  
    print("- DISPLAY",diccionario['id'],": *****")  
    print(self.diccionario)
```

Figura 335. Código completo de la clase display.py

11.1.2.2.7 *asignador.py* (2bvii)

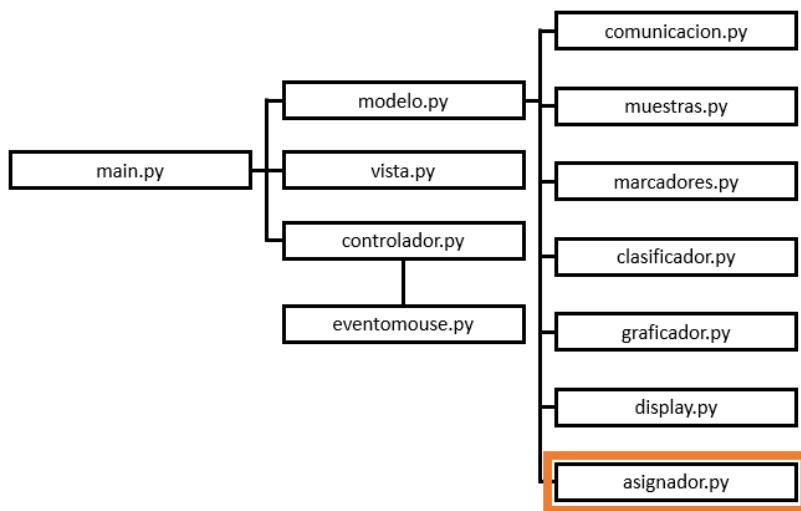


Figura 336. Ubicación de la clase `asignador.py`.

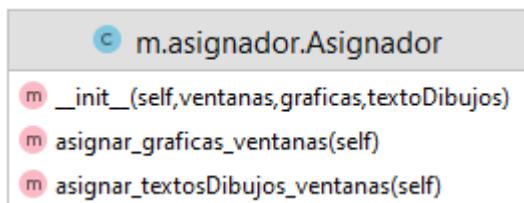


Figura 337. Métodos, atributos y propiedades de la clase `asignador.py`.

```

# ****
#
# Clase: --> Asignador
# Modulo: -> asignador.py
#
# Descripción:
# Asigna las ventanas de cada marcador a:
# 1) Graficas realizadas en "matplotlib"
# 2) Texto/Dibujos realizados por el programador
#
# Fecha: abril 10/2021
#
# ****

class Asignador:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,ventanas,graficas,textoDibujos):

        print("")
        print(" En CLASE Asignador:: CONSTRUCTOR")
        global v  # --- Arreglo "ventanas" ---
        global g  # --- Arreglo "graficas" ---
        global td  # --- Arreglo "texto-dibujos" ---
        global oa  # --- Arreglo "objetos-asignados" ---
        v = ventanas
        g = graficas
        td = textoDibujos
        oa = []

    def asignar_graficas_ventanas(self):

        print(" ")
        print(" En CLASE Asigandor::")
        print("*****")
        print(" --- Asignacion de graficas a ventanas ---")
        print("*****")

##        v[0].asignar_tipo_grafica(g[0])
##        v[1].asignar_tipo_grafica(g[1])
##        v[2].asignar_tipo_grafica(g[2])
##        v[3].asignar_tipo_grafica(g[3])
        oa.append({'grafica': g[0]})


```

```
oa.append({'grafica': g[1]})  
oa.append({'grafica': g[2]})  
oa.append({'textoDibujo': "informacion"})  
  
for i in range(len(v)):  
    v[i].asignar_objeto_ventana(oa[i])  
  
def asignar_textosDibujos_ventanas(self):  
  
    print(" ")  
    print(" En CLASE Asigandor::")  
    print("*****")  
    print(" --- Asignacion de Textos/Dibujos a ventanas ---")  
    print("*****")  
    #v[3].asignar_texto_dibujo(g[3])
```

Figura 338. Código completo de la clase asignador.py

11.1.2.3 *vista.py* (2c)

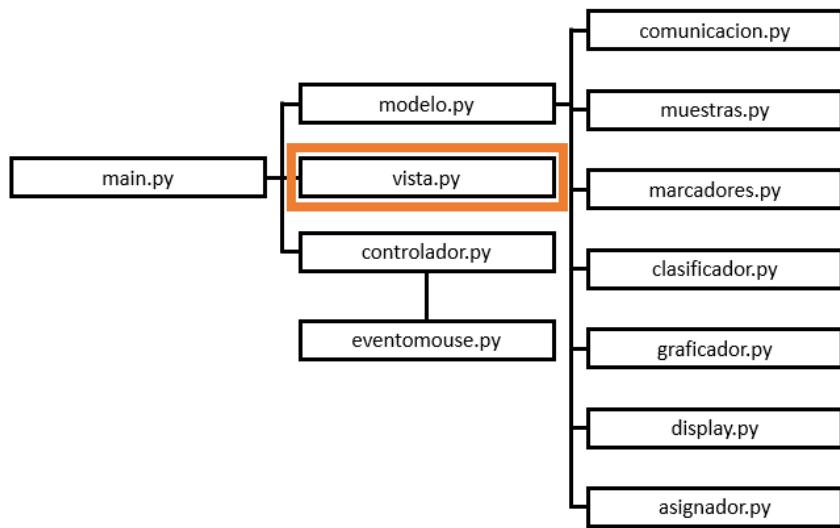


Figura 339. Ubicación de la clase *vista.py*.

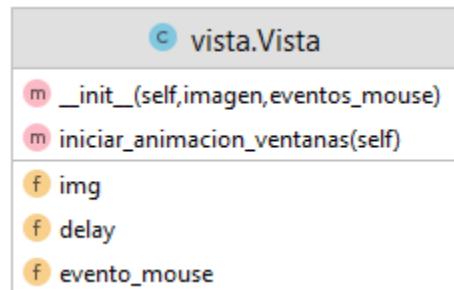


Figura 340. Métodos, atributos y propiedades de la clase *vista.py*.

```

# ****
#
# Clase: --> Vista
# Modulo: -> avista.py
#
# Descripción:
#   Crea un LOOP infinito para:
#     - Generar animación incial: movimientos de ventanas [display]
#     - Estar a la "escucha" de eventos de mouse
#
# Fecha: marzo 30/2021
#
# ****

import cv2
import time

class Vista:

    def __init__(self,imagen,eventos_mouse):

        print(" CONSTRUCTOR: Clase: Vista")
        self.img = imagen
        self.evento_mouse = eventos_mouse
        self.delay = 0.02
        self.iniciar_animacion_ventanas()

    def iniciar_animacion_ventanas(self):

        while True:
            img = self.img.copy()
            self.evento_mouse.mover_objetos(img)
            cv2.imshow("Objetos",img)
            time.sleep(self.delay)
            # --- Presionar "ESC" para salir de animacion ---
            if cv2.waitKey(1) == 27:
                break

```

Figura 341. Código completo de la clase vista.py

11.1.2.4 controlador.py (2d)

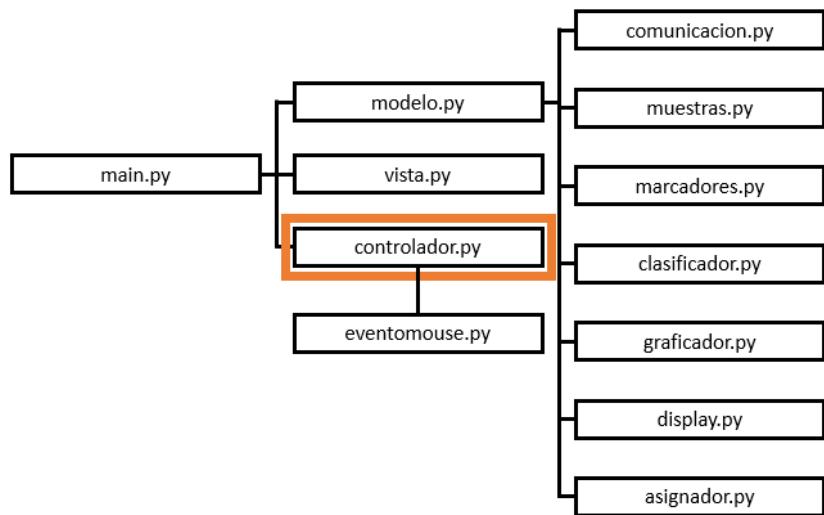


Figura 342. Ubicación del bloque controlador.py.

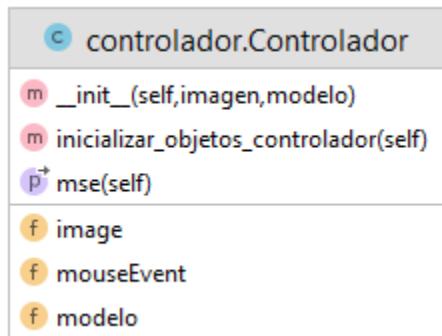


Figura 343. Métodos, atributos y propiedades de la clase controlador.py.

```

# ****
#
# Clase: --> Controlador
# Modulo: -> controlador.py
#
# Descripción:
#   Control de eventos de mouse para Realidad Aumentada
#
# Fecha: marzo 30/2021
#
# ****

import cv2
from c.eventomouse import EventoMouse

class Controlador:

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,imagen,modelo):

        print(" CONSTRUCTOR: Clase: Controlador")
        self.image = imagen
        self.modelo = modelo
        self.inicializar_objetos_controlador()

    def inicializar_objetos_controlador(self):

        # --- 8 - Manejo de Eventos de mouse ---
        mse = EventoMouse(self.image,self.modelo.ventanas)
        self.mouseEvent = mse
        cv2.namedWindow('Objetos')
        cv2.setMouseCallback('Objetos',mse.onMouse)

    @property
    def mse(self):
        return self.mouseEvent

```

Figura 344. Código completo de la clase controlador.py

11.1.2.4.1 Clase eventomouse.py (2di)

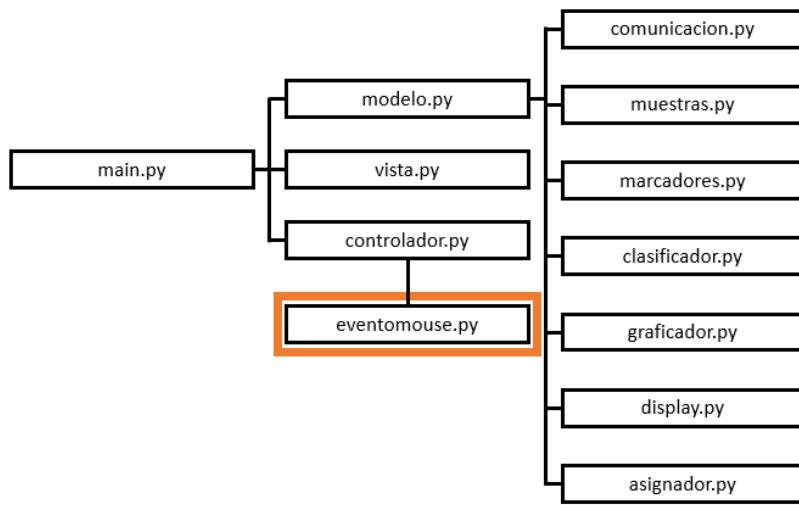


Figura 345. Ubicación de la clase eventomouse.py.

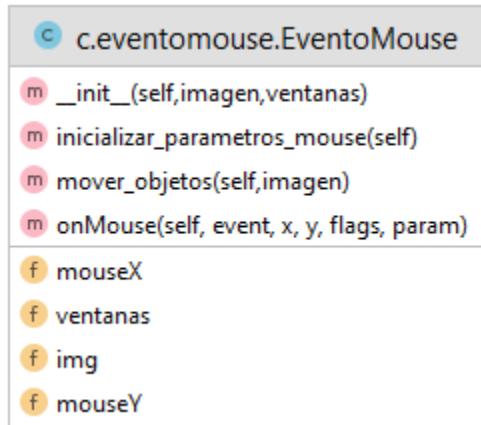


Figura 346. Métodos, atributos y propiedades de la clase eventomouse.py.

```

# ****
#
# Clase: --> EventoMouse
# Modulo: -> eventomouse.py
#
# Descripción:
# Manejo de eventos de mouse: onMouse
#     - Boton izquierdo ABAJO
#     - Mover mouse
#     - Boton izquierdo ARRIBA
#
# Fecha: marzo 30/2021
#
# ****

import cv2

class EventoMouse:

    def __init__(self,imagen,ventanas):

        print(" CONSTRUCTOR: Clase: Mouse")
        self.img = imagen
        self.ventanas = ventanas
        self.inicializar_parametros_mouse()

    def inicializar_parametros_mouse(self):

        self.mouseX = 0
        self.mouseY = 0

    def mover_objetos(self,imagen):

        for ventana in self.ventanas:
            ventana.move_springing_posicion()
            ventana.actualizar_imagen_display(imagen)
            if ventana.switch_inicio == 1:
                ventana.programar_secuencia_inicial()

    def onMouse(self, event, x, y, flags, param):

        global dist_x, dist_y
        # --- Boton izquierdo presionado ---
        if event == cv2.EVENT_LBUTTONDOWN:

```

```
print("*****")
print(" En Clase: EventoMouse")
print('x = %d, y = %d'%(x, y))
print("*****")
self.mouseX = x
self.mouseY = y
for ventana in self.ventanas:
    ventana.posicionar_ventana_mouse(self.mouseX,self.mouseY)
    if ventana.permitir_abrir_ventana:
        ventana.mouse_sobre_controlesVentana(self.mouseX,self.mouseY)
```

Figura 347. Código completo de la clase eventomouse.py

11.2 Diagramas esquemáticos y programas completos desarrollados

Se presentarán los diagramas esquemáticos y los completos desarrollados de los 4 circuitos bajo prueba propuestos:

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Toda la documentación, tanto diagramas esquemáticos como programas completos desarrollados también se encuentran en el siguiente repositorio:

<https://github.com/ferdan/CircuitosElectonicosRealidadAumentada>

11.2.1 Circuito Trazador de curvas (a)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
 - b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
 - c. **Amplificador de una etapa (c)** con un transistor MOSFET.
 - d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el **circuito Trazador de curvas (a)**, se presenta lo siguiente:

1. Diagrama esquemático del circuito.
 2. Programa de Arduino completo.
 3. De los programas en Python:
 - i. Clase calculadora_datos_graficas.py.
 - ii. Clase graficador.py.

11.2.1.1 Diagrama esquemático (1)

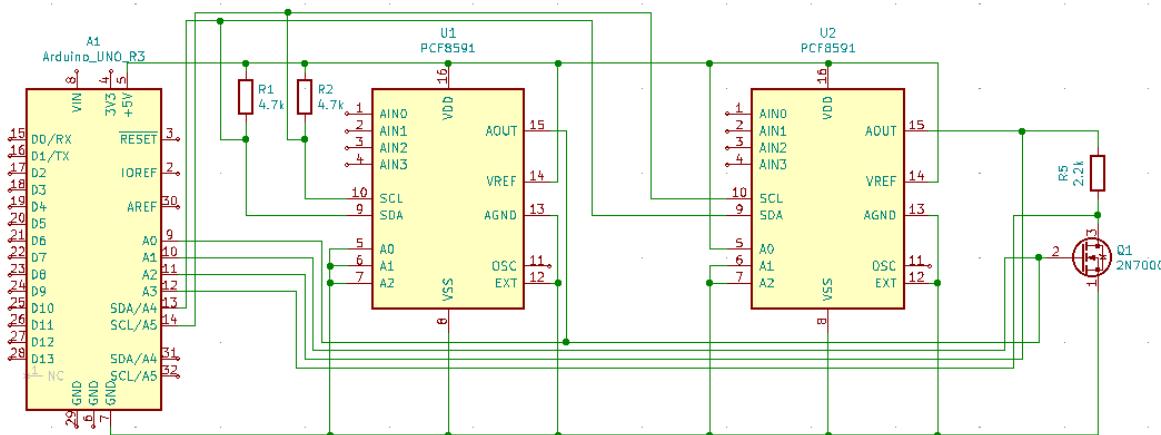


Figura 348. Diagrama esquemático completo del circuito Trazador de curvas (a).

11.2.1.2 Programa de Arduino (2)

Programa de Arduino para el Trazador de curvas
Nombre del programa: Arduino AR 65 MOSFET

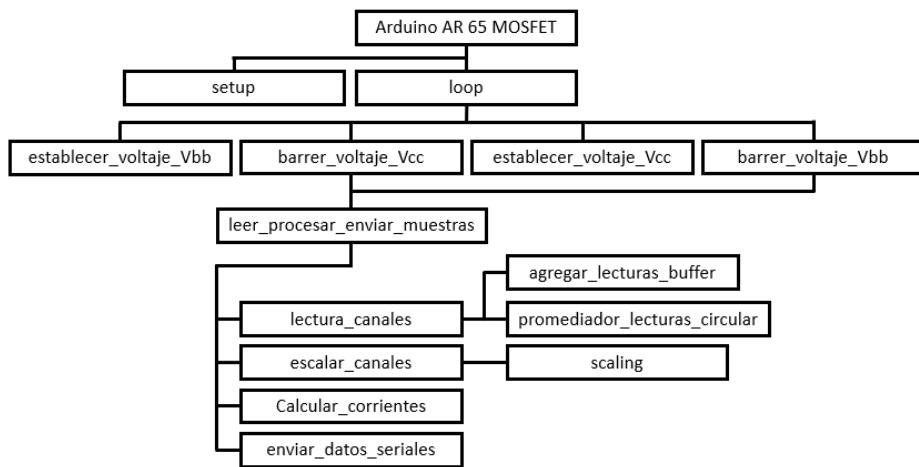


Figura 349. Diagrama a bloques del programa completo de Arduino para adquirir señales del circuito **Trazador de Curvas (a)**.

```
/*
PROGRAMA: 28072021

Trazador de curvas para un transistor MOSFET utilizando DACs

Este programa se ejecuta en conjunto con:

ARDUINO: ARDUINO_AR_65_MOSFET.ino (entradas) 08062021-A
PYTHON: Completo_AR_65_MOSFET.py (monitor) 08062021-P

TARJETA UTILIZADA: Arduino MEGA
TRANSISTOR DE PRUEBA: 2N7000
CONVERTIDORES DIGITALES ANALOGICOS UTILIZADOS: PCF8591

28 / julio / 2021

*/
#include "Wire.h"
#define PCF_Vgg 0x48 // I2C bus address
```

```

#define PCF_Vdd 0x49 // I2C bus address

// *****
// SELECCION DE MUESTRAS CON ó SIN 'Filtrado Circular'
//   'true' <- CON Filtrado || circular de 4 muestras
//   'false' <- SIN Filtrado
// *****
boolean Filtrado_Circular = true;
// *****

/** 
 * -----
----- 
 * Declaracion de arreglos:
 * canales_medidos[4] -> son los voltajes analogicos medidos directamente
del Arduino [0,1023]
 * voltajes[4] -> son los voltajes escalados del arreglo anterior [0,5]
 * corrientes[2] -> son las corrientes calculadas como la diferencia del
arreglo de voltajes anterior
 * -----
----- 

*/
const int numero_entradas_analogicas = 4;
int canales_medidos[numero_entradas_analogicas];
double voltajes[numero_entradas_analogicas];
double corrientes[2];

/** 
 * -----
----- 
 * Definicion de variables para el cálculo del FILTRO CIRCULAR
(Promediador)
 * Este filtro permite promediar cada lectura analogica de arduino
mediante un filtro circular
 * buffer_promediador[bufferSize]
 * bufferSize = tamaño del buffer
 * -----
----- 

*/
const int bufferSize = 32; // Numero de muestras a promediar
int promedios_calculados[numero_entradas_analogicas];
int buffer_promediador[numero_entradas_analogicas][bufferSize];
int index[numero_entradas_analogicas];
int canales[numero_entradas_analogicas];

/** 
 * -----
----- 
 * Definicion de pines para los 4 canales del ADC
 * PinVgg -> Es el voltaje a la salida del PCF_Vgg
 * PinVdd -> Es el voltaje a la salida del PCF_Vdd
 * PinVgs -> Es el voltaje en base-emisor del transistor MOSFET
 * PinVds -> Es el voltaje en colector-emisor del transistor MOSFET
 *
 * NOTA: Entre Vgg y Vgs no hay resistencia
 *       Entre Vdd y Vds hay una resistencia Rc de 2.2 kohm (valor
nominal)

```

```

* -----
*/
const int PinVgg = A0; // Pin de entrada analogica canal 0
const int PinVgs = A1; // Pin de entrada analogica canal 1
const int PinVdd = A2; // Pin de entrada analogica canal 2
const int PinVds = A3; // Pin de entrada analogica canal 3

/***
* -----
* Variables para el control del tiempo de muestreo
* lastime -> guarda el ultimo instante de tiempo en el que enviaron
* datos por el puerto serie
* sampleTime -> duracion entre cada muestra tomada
* numMuestras -> numero de muestras tomadas por cada curva
*
* Por lo tanto, la frecuencia de cada conjunto de muestras es de 2Hz
(numMuestras x sampleTime)/1000
* -----
*/
unsigned long lastTime = 0;
unsigned long sampleTime = 100;
const double numMuestras = 50;

/***
* -----
* Vdd -> valor medido de la fuente de voltaje que se utiliza como
alimentacion del trazador de curvas
* Vth -> valor observado experimentalmente de la curva Id vs Vgs en
donde se observa que comienza a crecer
*           el voltaje Vgs con respecto a la corriente Id (voltaje de
umbral)
* -----
*/
const double Vdd=5.0;
const double Vth=1.8;

/***
* -----
* i_muestra -> Indica el numero de la muestra que se toma. Para cada
conjunto de muestras, puede variar
*           entre [0,numMuestras]
* numero_curva -> Indica la curva de la cual se toman muestras. Las
curvas con numero_curva = 0 hasta 3,
*           se utilizan para graficar Id vs Vds. La curva con
numero_curva = 4 se utiliza para
*           graficar las demás curvas
* barrido_recta -> Realiza un barrido con un numero de muestras igual a
numMuestras para cada curva medida
* voltaje_obtencion_curva -> Valor de voltaje propuesto Vdd en formato
hexadecimal para las curvas Id vs Vds.

```

```

        *
        *                               Posteriormente, este mismo valor es Vgg
para las demas curvas
        * -----
        */
int i_muestra=0;
int numero_curva=0;
byte barrido_recta;
byte voltaje_obtencion_curva = 0x74;

/***
 * Inicializa tanto la comunicacion serial como la comunicacion I2C para
comunicarse con los dos PCF8591
*/
void setup() {
    Wire.begin();
    // --- Inicializa velocidad de transmision serial ---
    Serial.begin(9600);

    // --- Sincronizacion de comunicacion serial ---
    Serial.println("inicio");
}

/***
 * Se miden cinco curvas para graficar Id vs Vds. Posteriormente se toma
una ultima curva para graficar las demas curvas
*/
void loop() {
    // --- Toma muestras cada sampleTime ms ---
    if (millis()-lastTime > sampleTime) {

        lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
tomaron muestras

        if(numero_curva<5) // ¿Se toman muestras de las primeras 5 curvas
para Id vs Vds?
        {
            int valor;
            // --- 5 Curvas para Id vs Vds [MOSFET] ---
            valor = map(numero_curva,0,5,102,125); //probar tambien con 116 -
132

            establecer_voltaje_Vgg(valor);
            barrer_voltaje_Vdd();
        }
        else if(numero_curva==5)
        {
            // --- 1 Curva para Ig vs Vgs , Id vs Vgs , entre otras ---
            establecer_voltaje_Vdd(255);
            barrer_voltaje_Vgg();
        }
        else
        {

```

```

        numero_curva = 0;
    }
}

/***
 * Establece un valor de voltaje a Vgg en el PCF8591 correspondiente
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
*/
void establecer_voltaje_Vgg(int valor){
    Wire.beginTransmission(PCF_Vgg); // wake up PCF_Vgg
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}

/***
 * Establece un valor de voltaje a Vdd en el PCF8591 correspondiente
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
*/
void establecer_voltaje_Vdd(int valor){
    Wire.beginTransmission(PCF_Vdd); // wake up PCF_Vdd
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}

/***
 * Actualiza Vdd con diferentes valores de entre 0V y 5V
 * Despues envia por el puerto serie las muestras leidas
*/
void barrer_voltaje_Vdd(){
    // --- Barre durante 50 muestras ---
    for (int i_muestra=0; i_muestra<250; i_muestra+=5) {
        establecer_voltaje_Vdd(i_muestra);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

/***
 * Actualiza Vgg con diferentes valores de entre 0V y 5V
 * Despues envia por el puerto serie las muestras leidas
*/
void barrer_voltaje_Vgg(){
    // --- Transistor MOSFET: valores de Vgg mayores a Vth ---
    for (int i_muestra=0; i_muestra<150; i_muestra+=3) {
        establecer_voltaje_Vgg(i_muestra);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

```

```

}

/***
 * Lee los canales analogicos del circuito Vdd, Vds, Vgg y Vgs (4
voltajes)
* Escala los canales al voltaje de la fuente de alimentacion
* Calcula las corrientes a partir de la resta de voltajes
* Envia los datos de todos los canales por el puerto serie
*
*/
void leer_procesar_enviar_muestras(){
    // --- Lectura de los canales ---
    leer_canales(); // Lee los 4 voltajes del circuito

    // --- Escalamiento de canales ---
    escalar_canales(Vdd); // Escala los 4 voltajes leidos al voltaje de
alimentacion Vdd

    // -- calculo de corrientes
    calcular_corrientes(); // Calcula las corrientes con los voltajes
escalados

    // --- Transmision de cada valor al programa en Python ---
    enviar_datos_serials(); // Envia los 4 voltajes escalados y las 2
corrientes calculadas
}

/***
 * Lee los 4 voltajes del circuito. [Vgg,Vgs,Vdd,Vds]
*
*/
void leer_canales()
{
    if (Filtrado_Circular)
    {
        // --- CON FILTRADO CIRCULAR con buffer circular de 32 muestras ---
        for (int b=0; b<bufferSize; b++)
        {
            canales_medidos[0] = analogRead(PinVgg); // Vgg
            canales_medidos[1] = analogRead(PinVgs); // Vgs
            canales_medidos[2] = analogRead(PinVdd); // Vdd
            canales_medidos[3] = analogRead(PinVds); // Vds

            // Filtrado circular en cada lectura. SIN mezclar muestras en
tiempos diferentes
            agregar_lecturas_buffer(canales_medidos);
            promediador_lecturas_circular(canales_medidos);
        }
    } else {
        // --- SIN FILTRADO CIRCULAR ---
        canales_medidos[0] = analogRead(PinVgg); // Vgg
        canales_medidos[1] = analogRead(PinVgs); // Vgs
        canales_medidos[2] = analogRead(PinVdd); // Vdd
        canales_medidos[3] = analogRead(PinVds); // Vds

        for (int j=0; j<4; j++)
    }
}

```

```

        canales[j] = canales_medidos[j];
    }
}

/***
 * Escala los 4 canales de voltaje, que varian entre [0,1023] a un numero
que representa el voltaje [0,5]
*/
void escalar_canales(double Vdd_ref)
{
    for(int i=0;i<numero_entradas_analogicas;i++)
    {
        voltajes[i] = scaling(canales[i], 0, 1023, 0, Vdd_ref); //Vgg,Vbs,Vdd,Vds
    }
}

/***
 * Calcula las corrientes con los valores de voltaje escalados [Ig,Id]
 * NOTA: Las corrientes Ig e Id no estan escaladas al valor medido de las
resistencias Rg y Rd.
 *          Solo son una diferencia de voltajes. Ig = Vgg - Vgs
 *          Id = Vdd - Vds
 *
 */
void calcular_corrientes()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        corrientes[i] = (voltajes[i*2]-voltajes[i*2+1]);
    }
}

/***
 * Envia todos los datos por el puerto serial
 * En orden, se envian: [Vgg,Vgs,Ig,Vdd,Vds,Id]
 *
 */
void enviar_datos_seriales()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        Serial.println(voltajes[i*2],4); //Vgg,Vdd
        Serial.println(voltajes[i*2+1],4); //Vgs,Vds
        Serial.println(corrientes[i],4); //Ig,Id
    }
}

/***
 * Escala valores que se sabe que varian entre un intervalo definido
[in_min,in_max] a otro intervalo [out_min,out_max]
 * @param x -> es el valor a escalar
 * @param in_min -> es el valor minimo que puede tomar x antes del
escalado
 */

```

```

 * @param in_max -> es el valor maximo que puede tomar x antes del
escalado
 * @param out_min -> es el valor minimo al cual se escala x
 * @param out_max -> es el valor maximo al cual se escala x
 *
 */
float scaling(float x, float in_min, float in_max, float out_min, float
out_max)
{
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

// *****
// 
// FILTRO CIRCULAR (promediador) con buffer de 32 muestras
//
// *****
// --- Inicializacion del buffer ---
void agregar_lecturas_buffer(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        buffer_promediador[i][index[i]] = canales_medidos[i];
        index[i] += 1;
        if (index[i] >= bufferSize) index[i] = 0;
    }
}

// --- Filtro circular: promediador ---
void promediador_lecturas_circular(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        long sum = 0;
        for (int k=0; k<bufferSize; k++)
        {
            sum += buffer_promediador[i][k];
        }
        canales[i] = (int)(sum/bufferSize);
    }
}

```

Figura 350. Programa completo de Arduino

11.2.1.3 Programas en Python (3)

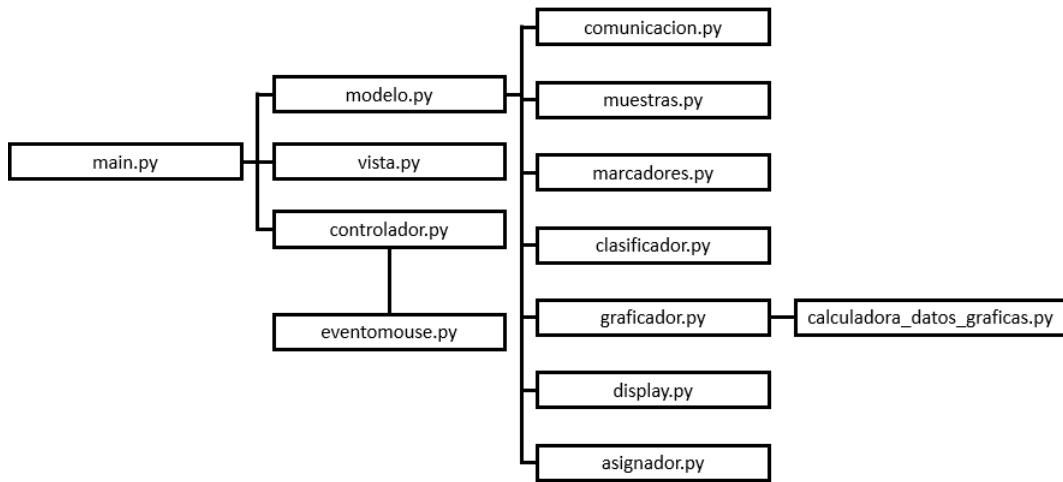


Figura 351. Diagrama de clases completo para el **círculo Trazador de curvas (a)**.

11.2.1.3.1 Clase calculadora_datos_graficas.py (3i)

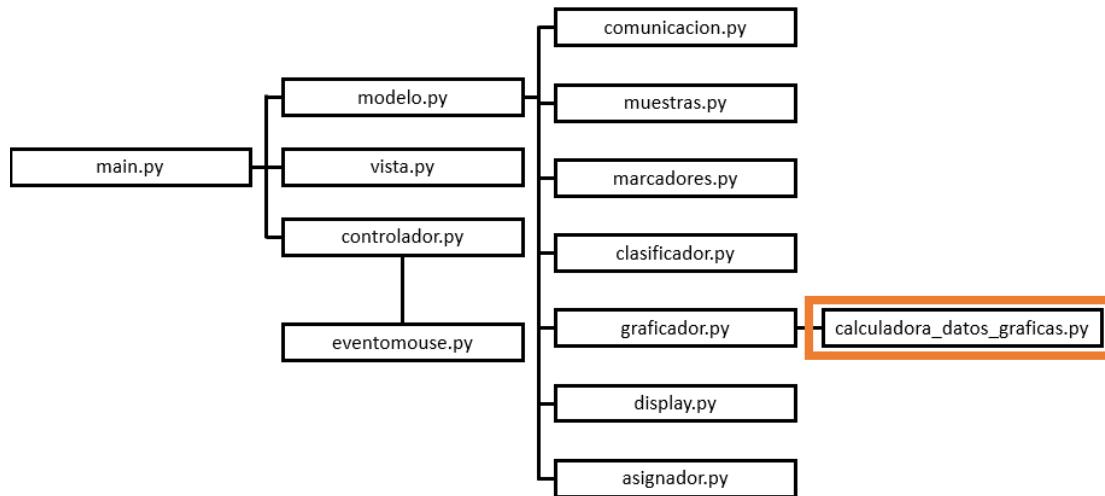


Figura 352. Ubicación de la clase calculadora_datos_graficas.py.

Esta clase contiene los siguientes métodos y atributos:

```
c m.calculadora_datos_graficas.Calculadora_datos_graficas
m __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes)
m estimate_coef(self, x, y)
m plot_regression_line(self, x, y, b)
m construir_series_datos(self)
m obtener_ind_min(self,ind_serie_datos)
m obtener_ind_max(self,ind_serie_datos,ind_min)
m regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max)
m inicializar_excel_datos(self)
m guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId, voltajeVds)
m finalizar_excel_datos(self,writer)
m leer_datos_excel(self,ind_variables, i)
m calcular_lambda(self, linea_lim_y)
m calcular_Vth_Kn_Idss(self)
m calcular_puntos_limite_grafica_ids_vds(self, Vth)

f muestras_llaves
f sel_filtro
f ind_voltajes
f ind_corrientes
f muestras
f modelo
f lim_max
```

Figura 353. Métodos, atributos y propiedades de la clase calculadora_datos_gráficas.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p[→] indica que es una **propiedad**.
- f indica que es un **atributo**.

```
#-----#
#-----#
# Clase: --> Calculadora_datos_graficas
# Modulo: -> calculadora_datos_graficas.py
#
# Descripción:
# Auxiliar de graficador.py que calcula los datos de las series de datos de las graficas a
mostrar
#
# Fecha: julio 28/2021
#
# ****
# *****

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import numpy as np
import pandas as pd
import math
import statistics

class Calculadora_datos_graficas:
    # -----
    #      CONSTRUCTOR
    # -----
    def __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes):

        print("")
        print(" CONSTRUCTOR: Clase: Calculadora_datos_graficas")
        self.modelo = modelo
        self.muestras = modelo.vectores_muestras
        self.muestras_llaves = muestras_llaves
        self.lim_max = lim_max
        self.sel_filtro = sel_filtro
        self.ind_corrientes = ind_corrientes
        self.ind_voltajes = ind_voltajes
```

```

# -----
# Calcula los coeficientes b0 y b1 del método de regresión lineal simple
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def estimate_coef(self, x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

# -----
# Grafica el resultado de la regresión lineal simple
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def plot_regression_line(self, x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

```

```

# -----
# Construye las series de datos de las diferentes curvas Id vs Vds (ej. Id1, Vd1, Id2, Vds2,
Id3, Vds3 ...)
#
# -----
def construir_series_datos(self):

    muestras = self.muestras
    ind_variables = self.ind_variables
    writer = self.inicializar_excel_datos()

    valores_id_vds = {i:[] for i in ind_variables}

    #print(ind_variables)
    #print('\n')
    #print(valores_id_vds)

    for i in range(0,int(len(ind_variables)),2):
        i_min = 50*int(i/2)
        i_max = 50*(int(i/2)+1)

        #print('valores_id_vds['+str(ind_variables[i])+']
        muestras[Id]['+str(i_min)+':'+'+str(i_max)+']')
        #print('valores_id_vds['+str(ind_variables[i+1])+']
        muestras[Vds]['+str(i_min)+':'+'+str(i_max)+']')

        valores_id_vds[ind_variables[i]] = muestras['Id'][i_min:i_max]
        valores_id_vds[ind_variables[i+1]] = muestras['Vds'][i_min:i_max]

    self.guardar_datos_excel(writer,ind_variables[i+1],ind_variables[i],valores_id_vds[ind_variables[i]],valores_id_vds[ind_variables[i+1]])
        #print(valores_id_vds[ind_variables[i]])
        #print(valores_id_vds[ind_variables[i+1]])
        #print('\n')

    self.finalizar_excel_datos(writer)

    return valores_id_vds

# -----
# Obtiene el límite mínimo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de corriente Id

```

```

# Esto es la primera muestra cuya diferencia con la anterior es menor a 0.06 y que el
índice de esa muestra sea mayor a la muestra número 2
#
def obtener_ind_min(self,ind_serie_datos):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes

    ind_min = 0;
    id_cambio = [x for x in
range(0,len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))]

    for i in range(1,30):
        id_cambio[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i]
        muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1] = -
        #print('i=' +str(i)+':'
        '+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i])+'
        '+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1])+'
        '+str(id_cambio[i]))
        #print('i: ' +str(i)+', '+str(id_cambio[i])+': '+str(id_cambio[i] < 0.06))
        if id_cambio[i] < (0.06)/1000 :
            #print('i: ' +str(i)+', '+str(id_cambio[i])+': '+str(id_cambio[i] < 0.06))
            if i > (2+int(ind_serie_datos/2)+1) :
                ind_min = i
                break

    #print('ind_min: id_cambio['+str(ind_min)+']: '+str(id_cambio[ind_min]))

    return ind_min

#
# Obtiene el límite máximo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de voltaje Vds
# Esto es la primera muestra cuya diferencia con la anterior es mayor a 0.05 y que el
índice de esa muestra sea mayor a la muestra número 20
#
def obtener_ind_max(self,ind_serie_datos,ind_min):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes

```

```

    ind_max = len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]])-2
    vds_cambio = [x for x in range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]]))]
    #print('Vds_cambio len: '+str(len(vds_cambio)))
    for i in range(20,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]])):
        vds_cambio[i] = muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][i] - muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][i-1]
        #print('i=' + str(i) + ': ' + str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][i]) + ' - ' + str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][i-1]) + ' = ' + str(vds_cambio[i]))
        #print('i: ' + str(i) + ', ' + str(vds_cambio[i]) + ': ' + str(vds_cambio[i] < 0.06))
        if vds_cambio[i] < 0.05 :
            #print('i: ' + str(i) + ', ' + str(vds_cambio[i]) + ': ' + str(vds_cambio[i] < 0.05))
            if i > ind_min :
                ind_max = i-1
                break

    #print('ind_max: vds_cambio[' + str(ind_max) + ']: ' + str(vds_cambio[ind_max]))

    return ind_max

# -----
# Funcion que obtiene lambda para un transistor mosfet
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    #print(valores_id_vds[ind_variables[ind_serie_datos+1]][ind_min:ind_max])
    #print(valores_id_vds[ind_variables[ind_serie_datos]][ind_min:ind_max])

    x = np.array(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][ind_min:ind_max]).reshape(-1)

```

```

y
np.array(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][ind_mi
n:ind_max]).reshape(-1)
=
```

```

#print(x)
#print(y)

b = self.estimate_coef(x,y)
#print("Estimated coefficients: b_0 = {} b_1 = {}".format(b[0], b[1]))
#plot_regression_line(x, y, b)

return -b[0] / b[1]
```

```

# --- Para depuracion
#
# -----
# Inicializa la escritura de datos en Excel
#
# -----
def inicializar_excel_datos(self):
    writer = pd.ExcelWriter('mediciones_promedio.xlsx', engine='xlsxwriter')
    return writer
```

```

# -----
# Guarda los valores de Id y Vds en Excel en Excel
#
# -----
def guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId,
voltajeVds):
    muestras_seriesId = pd.Series(corrienteId)
    muestras_seriesVds = pd.Series(voltajeVds)

    muestras_seriesId.to_excel(writer, sheet_name=nombre_hoja_id)
    muestras_seriesVds.to_excel(writer, sheet_name=nombre_hoja_vds)
```

```

# -----
# Finaliza la escritura de datos de Excel
#
# -----
def finalizar_excel_datos(self,writer):
    writer.save()
```

```

# -----
# Lee datos de Excel
#
```

```

# -----
def leer_datos_excel(self,ind_variables, i):
    df = pd.read_excel (r'C:\Users\ferda\Desktop\Material del profe 100421\3 Sistema Completo Realidad Aumentada\AR_multiples_graficas_Fernando_120421\Completo AR 20\mediciones_promedio_10.xlsx',
                                         sheet_name=ind_variables[i],
                                         usecols=[ind_variables[i]])
    return df

#-----
# -----
# FUNCION PRINCIPAL que calcula lambda de un transistor mos
# Obtiene primero los índices mínimos y máximos en donde se considera "recta" a la curva Id vs Vds
# Calcula lambda para esa curva
# Obtiene el promedio de todas las lambdas calculadas
# -----
def calcular_lambda(self,linea_lim_y):

    MOSFET_lambdas = []
    muestras = self.muestras
    lim_max = self.lim_max
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    #valores_id_vds = self.construir_series_datos()

    # Depuracion con datos leidos de excel
    """
    for i in range(1,6):
        ind_variables.append('Id'+str(i))
        ind_variables.append('Vds'+str(i))

    for i in range(len(ind_variables)):
        df = self.leer_datos_excel(ind_variables, i)
        valores_id_vds[ind_variables[i]] = df.values"""

    ind_min = 0
    ind_max = 0
    #print(valores)
    #print(valores['Id1'][0])

    for [ind_serie_datos,i] in zip(range(0,lim_max-1),range(1,lim_max)):


```

```

    #print('Id'+str(ind_serie_datos)+', '+Vds'+str(ind_serie_datos))
    #print('muestras[Id][ind_serie_datos]:'
    '+str(len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))+',
    +'muestras[Vds][ind_serie_datos]):'
    '+str(len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos]))+
        #print('Id'+str(ind_serie_datos)+': ')
        #print(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos])
        #print('Vds'+str(ind_serie_datos)+': ')
        #print(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos])

        ind_min =
muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos].index(linea_lim
_y[i])
        #ind_min = self.obtener_ind_min(ind_serie_datos)
        print('ind_min:           '+str(ind_min)+':           '+Id'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][ind_min]))

        ind_max = self.obtener_ind_max(ind_serie_datos,ind_min)
        print('ind_max:           '+str(ind_max)+':           '+Vds'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos][ind_max]))

        MOSFET_lambdas.append(self.regresion_lineal_lambda(ind_serie_datos,
ind_min, ind_max))
        print('lambda: '+str(MOSFET_lambdas)+'\n')

        print('lambda_promedio: '+str(statistics.mean(MOSFET_lambdas)))
        #return statistics.mean(MOSFET_lambdas)
        return MOSFET_lambdas

# -----
# Calcula los parametros Vth, Kn e Idss del transistor MOSFETT
# Ademas, calcula los indices de ids-vgs que se utilizaron para calcular estos parametros
(Vth,Kn,Idss)
# -----

def calcular_Vth_Kn_Idss(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    lim_max = self.lim_max
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    # --- Se calcula la maxima variacion de ids, entre muestras adyacentes ---
    ind_ids = []

```

```

delta_ids = [0]*50
for i in range(1,50-1):
    delta_ids[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][i]
- muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][i-1]
    #delta_vgs = muestras[muestras_llaves[ind_VOLTAJES[1]+6*sel_filtro]][lim_max-1][i] -
muestas[muestras_llaves[ind_VOLTAJES[1]+6*sel_filtro]][lim_max-1][i-1]
    #print(['+str(i)']+ 'delta_ids: '+str(delta_ids[i]))#+', delta_vgs: '+str(delta_vgs))

    # --- Se identifica el indice de la maxima variacion y el indice anterior a esta muestra -
--
ids_max = np.max(delta_ids)
ind_ids.append(delta_ids.index(ids_max)-1)
ind_ids.append(delta_ids.index(ids_max))
#print('ind_ids: '+str(ind_ids))

# --- Se calcula la diferencia de vgs y de ids, con los indices anteriores ---
vgs_1 = muestras[muestras_llaves[ind_VOLTAJES[1]+6*sel_filtro]][lim_max-1][ind_ids[0]]
vgs_2      =      muestras[muestras_llaves[ind_VOLTAJES[1]+6*sel_filtro]][lim_max-
1][ind_ids[len(ind_ids)-1]]
    ids_1      =      muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-
1][ind_ids[0]]
    ids_2      =      muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-
1][ind_ids[len(ind_ids)-1]]

#print('ids_1: '+str(ids_1))
#print('ids_2: '+str(ids_2))
#print('vgs_1: '+str(vgs_1))
#print('vgs_2: '+str(vgs_2))

# --- Se calcula vth, kn e idss ---
Vth = (vgs_1-vgs_2*math.sqrt((ids_1)/(ids_2)))/(1-math.sqrt((ids_1)/(ids_2)))
Kn = (ids_1)/((vgs_1-Vth)**2)
Idss = Kn * Vth**2

print('Vth_calculado: '+str(Vth))
print('Kn_calculada: '+str(Kn))
print('Idss_calculada: '+str(Idss))

return (Vth, Kn, Idss, ind_ids)

'''def calcular_vth(self, lim_max):
    Vth = 0
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves

```

```

sel_filtro = self.sel_filtro
ind_corrientes = self.ind_corrientes
ind_voltajes = self.ind_voltajes

for i in range(0,50):
    if muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][i] > 0.0001
and muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][i] > 1.7:
        Vth = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][i-1]
        print('Vth calculado = '+str(Vth))
        return Vth
    return 0

def calcular_kn(self, Idss, Vth):
    return Idss/(Vth**2)
# -----
# Calcula los puntos a graficar sobre las curvas vds-ids
# Se calcula la posicion de la curva que delimita las regiones ohmica y de saturacion de
un transistor MOSFET
# Regresa tambien el valor promedio vgs al cual se grafico esa curva
# -----
def calcular_puntos_limite_grafica_ids_vds(self, Vth):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_id_vds = [0,1,2,3,4]

    linea_lim_x = [0] * (len(num_curvas_id_vds)+1)
    linea_lim_y = [0] * (len(num_curvas_id_vds)+1)
    Vgs_prom = [0] * (len(num_curvas_id_vds))

    # --- Se calcula la posicion de los puntos que delimitan las regiones de operacion sobre
el conjunto de curvas ids-vds ---
    for i in num_curvas_id_vds:
        Vgs_prom[i] =
        round(np.average(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][i]),3)
        # --- Se obtiene el voltaje de saturacion vds_sat ---
        linea_lim_x[i+1] = Vgs_prom[i]-Vth
        for j in range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i])):
            if
                abs((Vgs_prom[i]-Vth) -
muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i][j]) < 0.05:
                    # --- Se obtiene la muestra mas cercana de ids correspondiente al voltaje
vds_sat ---

```

```

        linea_lim_y[i+1] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][j]
        #print('lim_curva_'+str(i)+'
                                         (temp):
'+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+'A,      dif='+str(abs((Vgs_prom-Vth) -
muestras[muestras_llaves[ind_volajes[3]+6*sel_filtro]][i][j])/220))
        print('lim_curva_'+str(i)+': '+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+'A')

return (linea_lim_x,linea_lim_y,Vgs_prom)

```

Figura 354. Código completo de la clase calculador_datos_graficas.py

11.2.1.3.1 Clase graficador.py (3ii)

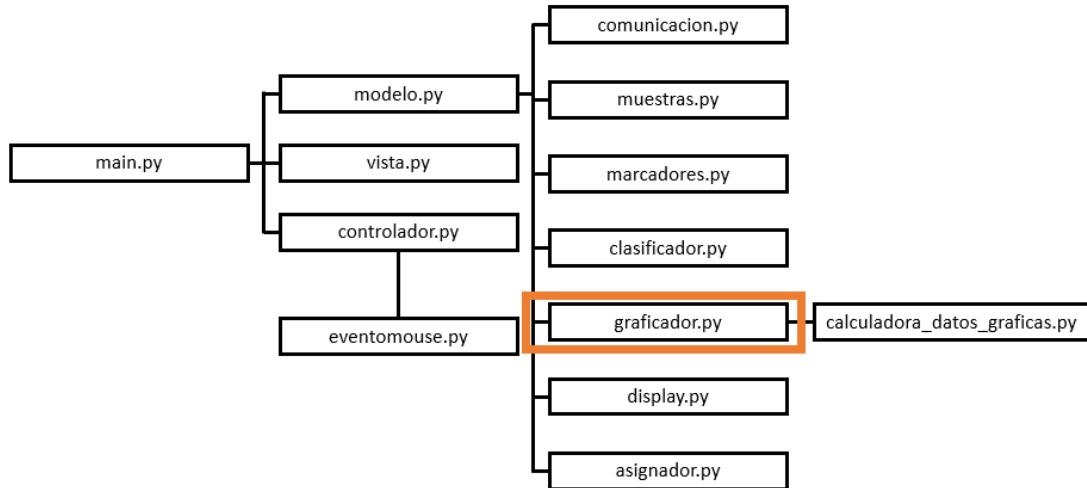


Figura 355. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

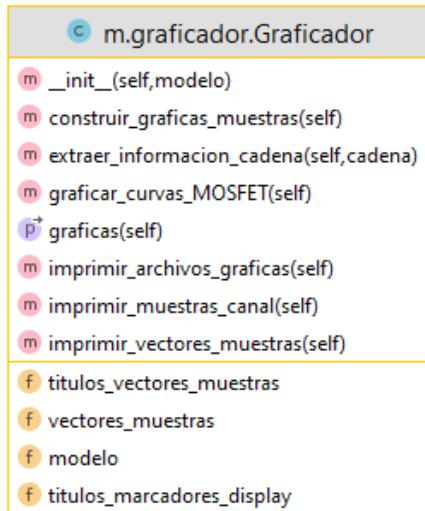


Figura 356. Métodos y atributos de la clase graficador.py.

Donde:

-  indica que es una **clase**.
-  indica que es un **método**.
-  indica que es una **propiedad**.
-  indica que es un **atributo**.

```
# ****
#
# Clase: --> Graficador
# Modulo: -> graficador.py
#
# Descripción:
# - Graficar los datos obtenidos por arduino
# - Dibuar, mostrar informacion
#
# Fecha: julio 28/2021
#
# ****
from m.calculadora_datos_graficas import Calculadora_datos_graficas

import numpy as np
import cv2

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.ticker import EngFormatter

from collections import deque

from engineering_notation import EngNumber
import math
import statistics

graficas = []

class Graficador():

    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,modelo):
```

```

print("")
print(" CONSTRUCTOR: Clase: Graficador")
self.modelo = modelo
self.vectores_muestras = modelo.vectores_muestras
self.titulos_vectores_muestras = modelo.titulos_vectores_muestras
self.titulos_marcadores_display = modelo.titulos_marcadores_display

def construir_graficas_muestras(self):

    # --- Construccion AUTOMATICA de la graficas ---
    titulos_marcadores = list(self.titulos_marcadores_display)

    global graficas
    graficas = []

    muestras = self.vectores_muestras

    for i in range(len(titulos_marcadores)):
        titulo_grafica = self.titulos_marcadores_display[titulos_marcadores[i]]
        nombre_vectores_grafica = self.extraer_informacion_cadena(titulo_grafica)

        # --- Construccion AUTOMATICA de cada GRAFICA, a partir de la
        #   informacion proporcionada en la clase Modelo en las estructuras:
        #   -> titulos_vectores_muestras <-> Vector
        #   -> titulos_marcadores_display <-> Diccionario
        #   Estas estructuras de datos se conectan con los nombres de los
        #   vectores que se encuentran en el Diccionario muestras
        #   -> vectores_muestras <-> Diccionario
        #
        # -----
        plt.figure(i)
        # --- Graficas VECTOR A vs. muestras ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 1:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      label = nombre_vectores_grafica[0][0] + ' vs. muestras')
        # --- Graficas VECTOR A vs VECTOR B ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 2:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      muestras[nombre_vectores_grafica[0][1]],
                      label = nombre_vectores_grafica[0][0] + ' vs. ' +
                      nombre_vectores_grafica[0][1])
        # --- Graficas MULTIPLES tipo: VECTOR A vs VECTOR B ---
        if len(nombre_vectores_grafica) > 2:
            for g in range(len(nombre_vectores_grafica)):
```

```

plt.plot(muestras[nombre_vectores_grafica[g][0]],
         muestras[nombre_vectores_grafica[g][1]],
         label = nombre_vectores_grafica[g][0] + ' vs. ' +
         nombre_vectores_grafica[g][1])
plt.title = 'Grafica ' + str(i)
plt.ylabel = 'y label'
plt.xlabel = 'x label'
plt.legend()

# --- Guardado de las gráficas en:
#   1. "Archivo en disco [*.jpg] para su uso posterior
#      por parte de la Clase Display
#   2. El arreglo GLOBAL: 'graficas', el cual contiene
#      la ruta en donde se guardó la gráfica correspondiente
# -----
grafica = 'g/grafica_' + str(i) + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

def extraer_informacion_cadena(self,cadena):

    #print("")
    #print(" cadena original = ",cadena)

    # --- Separa grupos de letras ---
    subcadena = ""
    subtulos = []
    ignorar_caracter = False
    num_char = 0
    for ch in cadena:
        num_char += 1
        # --- Ignorar los caracteres "espacio" [' '] y "coma" [','] ---
        if ch == ' ' or ch == ',':
            ignorar_caracter = True
            if num_char != 1:
                subtulos.append(subcadena)
            subcadena = ""
        else:
            subcadena += ch
        if num_char == len(cadena): subtulos.append(subcadena)

    # --- Genera tuplas de dos elementos ---
    tupla = []
    arreglo_tuplas = []

```

```

for i in range(len(subtitulos)):
    # --- Detecta conector ['vs'] ó espacio [ ' ] --
    if subtitulos[i] == 'vs' or subtitulos[i] == ":
        dummy = 0 # <--- No hacer nada ---
    else:
        tupla.append(subtitulos[i])
    if subtitulos[i] == " or i == (len(subtitulos)-1):
        arreglo_tuplas.append(tupla)
        tupla = []

return arreglo_tuplas

def graficar_curvas_MOSFET(self):

    global graficas
    graficas = []

    muestras = self.vectores_muestras

    muestras_llaves = list(muestras)

    # Son los indices de los canales de muestras recibidos como se difnio en la clase
    modelo
    ind_voltajes = [0,1,3,4] #Vgg, Vgs, Vdd, Vds
    ind_corrientes = [2,5] # Ig, Id

    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras
    hay 6 curvas)
    lim_max = len(muestras['Id'])
    #print('lim_max: '+str(lim_max))
    #print('muestras_llaves: '+str(len(muestras_llaves)))

    # Seleccion entre las series de muestras sin filtro, con filtro circular o con filtro pasa-
    bajas
    # 0 = sin filtro
    # 1 = con filtro circular cp
    # 2 = con filtro pasa bajas lp

    sel_filtro = 1

    # Ajuste de Id con Rd = 220 ohm y de Ig con Rg = 100 Kohm

    for i in range(ind_corrientes[1],lim_max*3,6):
        #print('Serie '+str(muestras_llaves[i])+' por ajustar')

```

```

        for j in range(0,lim_max):
            muestras[muestras_llaves[i]][j] = [x / 2200 for x in muestras[muestras_llaves[i]][j]]
            muestras[muestras_llaves[i-3]][j] = [x / 100000 for x in muestras[muestras_llaves[i-3]][j]]
        #print('Serie '+str(muestras_llaves[i])+' ajustada')

# Se obtienen los parametros lambda, Vth, Kn y gm.
calc = Calculadora_datos_graficas(self.modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes)

Vth, Kn, Idss, ind_ids = calc.calcular_Vth_Kn_Idss()
linea_lim_x,linea_lim_y,Vgs_prom = calc.calcular_puntos_límite_grafica_ids_vds(Vth)
MOS_lambdas = calc.calcular_lambda(linea_lim_y)
MOS_avg_lambda = -(1/statistics.mean(MOS_lambdas))
#print(MOS_avg_lambda)

gm = [0] * (lim_max-1)
corrientes_gm = [0] * (lim_max-1)

for i in range(0,lim_max-1):
    corrientes_gm[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][25]
    gm[i] = 2*math.sqrt(Kn*corrientes_gm[i])

#print('corriente: '+str(corrientes_gm)+', gm: '+str(gm))

#print('Parametros Híbridos')
# --- Parametros adicionales: rg, rd -----
-----
rg = float('inf')
#print(rg)

rd = [0] * (lim_max-1)

for i in range(0,lim_max-1):
    rd[i] = (1/(MOS_avg_lambda*corrientes_gm[i]))
    #print(rd[i])

# -----
#
# Algunas referencias que consulte para realizar el código, son las siguientes:
#

```

```

# Python matplotlib data labels (plt.annotate())
# https://queirozf.com/entries/add-labels-and-text-to-matplotlib-plots-annotation-examples
#
# Python matplotlib change axis limits (see first two answers)
# https://stackoverflow.com/questions/3777861/setting-y-axis-limit-in-matplotlib
#
# Python engineering format (EngNumber())
# https://pypi.org/project/engineering-notation/
#
# Python insert greek symbols
# https://pythonforundergradengineers.com/unicode-characters-in-python.html
#
# Utilizar parentesis en vez de iguales
# ej. plt.xlabel('Vds (V)') en vez de plt.xlabel = 'Vds (V)'
#
# Engineering format en axis labels (plt.gca(), EngFormatter(),
ax.xaxis.set_major_formatter() y ax.yaxis.set_major_formatter())
#
# -----
# https://matplotlib.org/stable/gallery/text_labels_and_annotations/engineering_formatter.html
# https://www.geeksforgeeks.org/formatting-axes-in-python-matplotlib/
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.gca.html
# -----
-----
```

```

print('Grafica 0')
# --- Grafica 0 : Voltajes en el tiempo -----
-----
```

```

# --- Esta grafica es para observar el comportamiento de los voltajes en el tiempo
# --- Se observan los voltajes Vgg, Vgs, Vdd, Vds
plt.figure(0)

ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)

for i in ind_voltajes:
    arr = []
    for j in range(0,lim_max):
        arr = arr+muestras[muestras_llaves[i+6*sel_filtro]][j]
    plt.plot(arr, label=muestras_llaves[i+6*sel_filtro])

plt.xlabel('Muestras')
```

```

plt.ylabel('Voltaje')
plt.title("Voltajes en el tiempo")
plt.legend()
grafica = 'g/00_Voltajes' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 1')
# --- Grafica 1: Ig vs Vgs -----
-----
plt.figure(1)
plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][[lim_max-1],muestras[muestras_llaves[ind_corrientes[0]+6*sel_filtro]][[lim_max-1], label='Ig vs Vgs']

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

x1,x2,y1,y2=plt.axis()
plt.xlim(0,4)
plt.ylim(y1-y2*10,y2*1000)
plt.xlabel('Vgs')
plt.ylabel('Ig')
plt.subplots_adjust(left=0.15)
plt.title("Ig vs Vgs")
plt.legend()
grafica = 'g/01_Ig_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 2')
# --- Grafica 2: Id vs Vgs -----
-----
plt.figure(2)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

```

```

plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][lim_max-1][0:ind_ids[len(ind_ids)-1]], muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][lim_max-1][0:ind_ids[len(ind_ids)-1]], label='Id vs Vgs')
x1,x2,y1,y2=plt.axis()
plt.axis([0,x2,y1,y2])
plt.xlabel('Vgs')
plt.ylabel('Id')
plt.title("Id vs Vgs")
plt.legend()
grafica = 'g/02_Id_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 3')
# --- Grafica 3: Id vs Vds (Cinco curvas) -----
-----
plt.figure(3)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

for i in range(0,lim_max-1):

plt.plot(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i], muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i], label='Vgs' = 'str(round(np.average(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][i]),3)))')

plt.xlabel('Vds')
plt.ylabel('Id')
plt.title("Id vs Vds")
plt.legend()
grafica = 'g/03_Id_vs_Vds' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 4')
# --- Grafica 4: gm vs Id -----
-----
```

```

plt.figure(4)

ax = plt.gca()
formatter0 = EngFormatter(unit='\u03c3')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm, gm, 'o-', label='gm vs Id')

for i in range(0, len(gm)):

    plt.annotate(str(EngNumber(gm[i])) + '\u03c3', (corrientes_gm[i], gm[i]), textcoords="offset
    points", xytext=(0, 10), ha='center')
    x1, x2, y1, y2 = plt.axis()
    plt.axis([x1 - x1 / 8, x2 + x2 / 8, y1, y2 + y2 / 16])
    plt.xlabel('Id')
    plt.ylabel('gm')
    plt.title("gm vs Id")
    plt.legend()
    grafica = 'g/04_gm_vs_Id' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

print('Grafica 5')
# --- Grafica 5: rd vs Id -----
-----
```

```

plt.figure(5)

ax = plt.gca()
formatter0 = EngFormatter(unit='\u03a9')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm, rd, 'o-', label='rd vs Id')
for i in range(0, len(rd)):

    plt.annotate(str(EngNumber(rd[i])) + '\u03a9', (corrientes_gm[i], rd[i]), textcoords="offset
    points", xytext=(0, 10), ha='left')
    x1, x2, y1, y2 = plt.axis()
```

```

plt.axis([x1,x2+x2/4,y1,y2+y2/4])
plt.xlabel('Id')
plt.ylabel('rd')
plt.title("rd vs Id")
plt.legend()
grafica = 'g/05_rd_vs_Id' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 6')
# --- Grafica 6 Resumen parametros hibridos-----
-----



# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- RESUMEN PARAMETROS HIBRIDOS ---

cv2.putText(canvas2,"Transistor      MOSFET      2N7000", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"PARAMETROS      HIBRIDOS", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Lambda : " + str(EngNumber(MOS_avg_lambda)) + 'V^-1', (30,
130),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Voltaje de umbral de encendido : " + str(EngNumber(Vth)) + 'V',
(30, 150),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Idss : " + str(EngNumber(Idss)) + 'A', (30,
170),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Kn : " + str(EngNumber(Kn)) + 'A/V^2', (30,
190),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"rg : " + str(rg), (30, 210),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)

filename = "g/06_parametros_hibridos" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Auxiliar para depuracion ---
#cv2.imshow("Canvas",canvas2)
#cv2.waitKey(0)
plt.show()

```

```

@property
def graficas(self):
    return graficas # --- Nombre de la grafica y ruta donde se guarda ---

def imprimir_archivos_graficas(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" --- Archivos de imagenes de graficas *.jpg ---")
    print("*****")
    for g in graficas:
        print("")
        print(g)

def imprimir_muestras_canal(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" --- Vectores de muestras por canal ---")
    print("*****")
    # --- ESTRUCTURA del diccionario:
    #   datos = {canal: i, llaveMuestras[i]: vector de datos}
    # -----
    muestras = self.vectores_muestras
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k]," : ",muestras[llaves[k]])

def imprimir_vectores_muestras(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" -- Muestras leidas de la Clase Modelo --")
    print("*****")
    muestras = self.vectores_muestras
    llaves = self.titulos_vectores_muestras
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
```

```
    print("")  
    print(" --- ",llaves[k],": ",muestras[llaves[k]])
```

Figura 357. Código completo de la clase graficador.py

11.2.2 Circuito generador de señal senoidal (b)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el **circuito generador de señal senoidal (b)**, se presenta lo siguiente:

1. Diagrama esquemático del circuito.
2. Programa de Arduino completo.
3. De los programas en Python:
 - i. Clase graficador.py.

11.2.2.1 Diagrama esquemático (1)

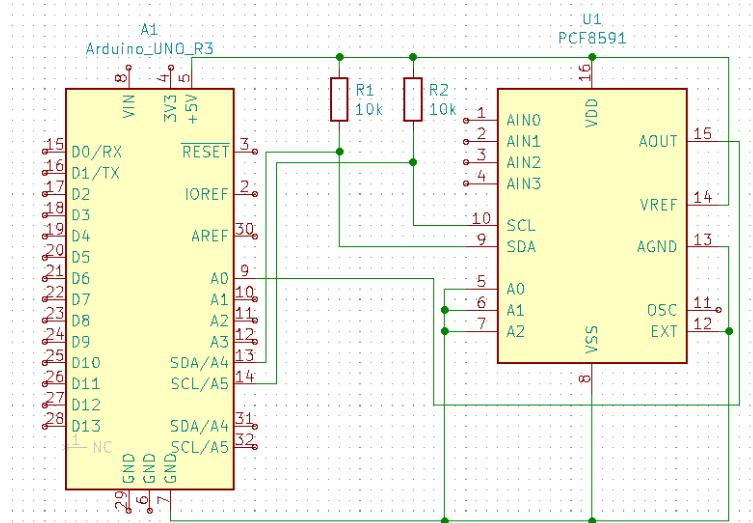


Figura 358. Diagrama esquemático completo para el **circuito generador de señal senoidal (b)**.

11.2.2.2 Programa de Arduino (2)

Programa de Arduino para el Circuito Generador de Señal Senoidal
Nombre del programa: Arduino AR 65 DDS

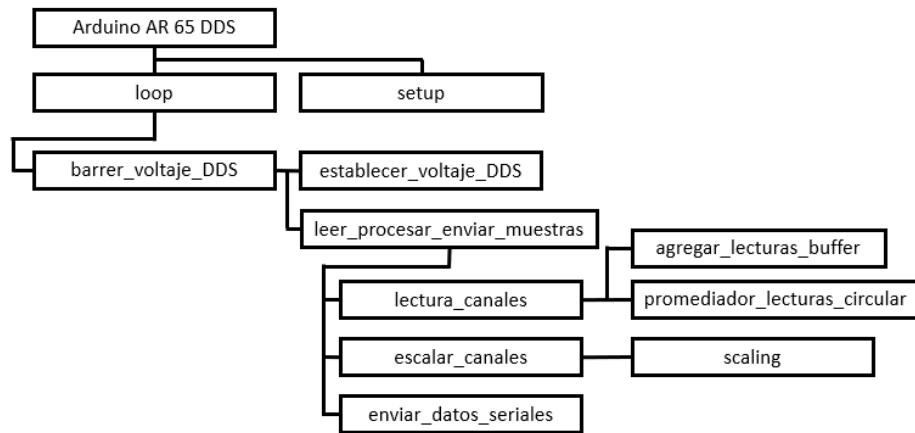


Figura 359. Diagrama a bloques del programa completo para adquirir señales del **circuito Generador de Señal Senoidal (b)**.

```
/*
PROGRAMA: 08062021

Trazador de curvas para un circuito DDS

Este programa se ejecuta en conjunto con:

ARDUINO: ARDUINO AR 65 DDS.ino (entradas) 25062021-A
PYTHON: Completo AR 65 DDS.py (monitor) 25062021-P

TARJETA UTILIZADA: Arduino Mega2560
CONVERTIDORES DIGITALES ANALOGICOS UTILIZADOS: PCF8591

8 / junio / 2021
*/
#include "Wire.h"
#define PCF_DDS 0x48 // I2C bus address
// *****
// SELECCION DE MUESTRAS CON ó SIN 'Filtrado Circular'
// 'true' <- CON Filtrado || circular de 4 muestras
// 'false' <- SIN Filtrado
// *****
boolean Filtrado_Circular = false;
```

```

// ****
/** 
* -----
* Declaracion de arreglos:
* numero_entradas_analogicas -> indica el numero de canales analogicos a
leer del circuito
* canales_medidos[2] -> son los voltajes analogicos medidos directamente
del Arduino [0,1023]
* voltajes[2] -> son los voltajes escalados del arreglo anterior [0,5]
* -----
*/
const int numero_entradas_analogicas = 2;
int canales_medidos[numero_entradas_analogicas];
double voltajes[numero_entradas_analogicas-1];
/** 
* -----
* Definicion de variables para el cálculo del FILTRO CIRCULAR
(Promediador)
* Este filtro permite promediar cada lectura analogica de arduino
mediante un filtro circular
* buffer_promediador[bufferSize]
* bufferSize = tamaño del buffer
* -----
*/
const int bufferSize = 32; // Número de muestras a promediar
int promedios_calculados[numero_entradas_analogicas];
int buffer_promediador[numero_entradas_analogicas][bufferSize];
int index[numero_entradas_analogicas];
int canales[numero_entradas_analogicas];

/** 
* -----
* Definicion del pin de la señal senoidal
* PinDDS -> Es el voltaje a la salida del PCF_DDS
* -----
*/
const int PinDDS = A0; // Pin de entrada senoidal

/** 
* -----
* Variables para el control del tiempo de muestreo
* lastime -> guarda el ultimo instante de tiempo en el que enviaron
datos por el puerto serie
* sampleTime -> duración entre cada muestra tomada
* numMuestras -> número de muestras tomadas por cada curva
*
* Por lo tanto, la frecuencia de cada conjunto de muestras es de 2Hz
(numMuestras x sampleTime)/1000

```

```

* -----
*/
unsigned long lastTime = 0;
unsigned long sampleTime = 100;
const double numMuestras = 50;

/***
* -----
* Vcc -> valor medido de la fuente de voltaje que se utiliza como
alimentacion del trazador de curvas
* -----
*/
const double Vcc=5.0;

/***
* -----
* i_muestra -> Indica el numero de la muestra que se toma. Para cada
conjunto de muestras, puede variar
*           entre [0,numMuestras]
* sine -> Realiza un barrido con un numero de muestras igual a
numMuestras para cada curva medida
* -----
*/
int i_muestra=0;
byte sine;

/***
* Inicializa tanto la comunicacion serial como la comunicacion I2C para
comunicarse con los dos PCF8591
*/
void setup() {
  Wire.begin();
  // --- Inicializa velocidad de transmision serial ---
  Serial.begin(9600);

  // --- Sincronizacion de comunicacion serial ---
  Serial.println("inicio");
}

/***
* Se mide al menos una curva para graficar la senal senoidal.
*/
void loop() {
  // --- Toma muestras cada sampleTime ms ---
  if (millis()-lastTime > sampleTime) {
    lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
tomaron muestras
    barrer_voltaje_senoidal();
  }
}

```

```

/**
 * Obtiene el valor binario entre 0x00 y 0xFF a partir de la ecuacion de
la forma de onda senoidal
 */
void barrer_voltaje_senoidal(){
    // --- Barre durante 50 muestras ---
    for (int i_muestra=0; i_muestra<250; i_muestra+=5) {
        sine =
byte(127.5*(1+sin((1000/sampleTime)*TWO_PI*i_muestra/(numMuestras-1))));
        establecer_voltaje_senoidal(sine);

        leer_procesar_enviar_muestras();
    }
}

/**
 * Establece el valor binario en el PCF8591. Este valor binario sera
convertido a un voltaje
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
 */
void establecer_voltaje_senoidal(int valor){
    Wire.beginTransmission(PCF_DDS); // wake up PCF_Vcc
    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}

/**
 * Realiza la lectura, escalado y envio de datos por el puerto serial
 */
void leer_procesar_enviar_muestras(){
    // --- Lectura de los canales ---
    lectura_canales(); // Lee el voltaje senoidal del circuito

    // --- Escalamiento de canales ---
    escalar_canales(Vcc); // Escala el voltaje senoidal leido al voltaje
de alimentacion Vcc

    // --- Transmision de cada valor al programa en Python ---
    enviar_datos_serials(); // Envia el voltaje senoidal
}

/**
 * Lee el voltaje senoidal del circuito.
 */
void lectura_canales()
{
    canales_medidos[0] = analogRead(PinDDS);
    canales_medidos[1] = sine;

    if (Filtrado_Circular)

```

```

{
    // --- CON FILTRADO CIRCULAR con buffer circular de 32 muestras ---
    for (int b=0; b<bufferSize; b++)
    {
        // Filtrado circular en cada lectura. SIN mezclar muestras en
        tiempos diferentes
        agregar_lecturas_buffer(canales_medidos);
        promediador_lecturas_circular(canales_medidos);
    }
} else {
    // --- SIN FILTRADO CIRCULAR ---
    for(int j=0; j<numero_entradas_analogicas; j++)
    {
        canales[j] = canales_medidos[j];
    }
}
}

/**
 * Escala el canal de voltaje senoidal, que varia entre [0,1023] a un
numero que representa el voltaje [0,5]
*/
void escalar_canales(double Vcc_ref)
{
    voltajes[0] = scaling(canales[0], 0, 1023, 0, Vcc_ref);
}

/**
 * Envia todos los datos por el puerto serial
 * En orden, se envia:
 *   1. Voltaje de la senal senoidal
 *   2. Numero en binario que genera ese voltaje
 */
void enviar_datos_seriales()
{
    Serial.println(voltajes[0]); // Voltaje
    Serial.println(canales[1]); // Valor binario
}

/**
 * Escala valores que se sabe que varian entre un intervalo definido
[in_min,in_max] a otro intervalo [out_min,out_max]
 * @param x -> es el valor a escalar
 * @param in_min -> es el valor minimo que puede tomar x
 * @param in_max -> es el valor maximo que puede tomar x
 * @param out_min -> es el valor minimo al cual se escala x
 * @param out_max -> es el valor maximo al cual se escala x
 */
float scaling(float x, float in_min, float in_max, float out_min, float
out_max)
{
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

```

```

// *****
// 
// FILTRO CIRCULAR (promediador) con buffer de 32 muestras
// 
// *****
// --- Inicializacion del buffer ---
void agregar_lecturas_buffer(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        buffer_promediador[i][index[i]] = canales_medidos[i];
        index[i] += 1;
        if (index[i] >= bufferSize) index[i] = 0;
    }
}

// --- Filtro circular: promediador ---
void promediador_lecturas_circular(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        long sum = 0;
        for (int k=0; k<bufferSize; k++)
        {
            sum += buffer_promediador[i][k];
        }
        canales[i] = (int)(sum/bufferSize);
    }
}

```

Figura 360. Programa completo de Arduino

11.2.2.3 Programas en Python (3)

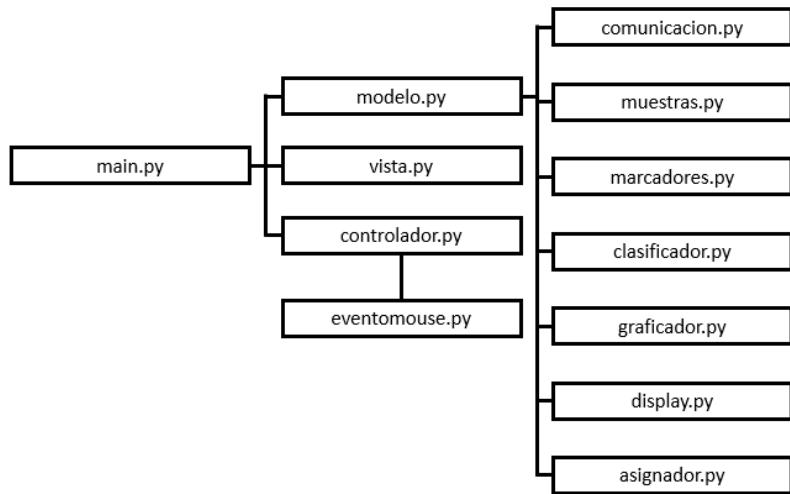


Figura 361. Diagrama de clases para el **circuito generador de señal senoidal (b)**.

11.2.2.3.1 Clase graficador.py (3i)

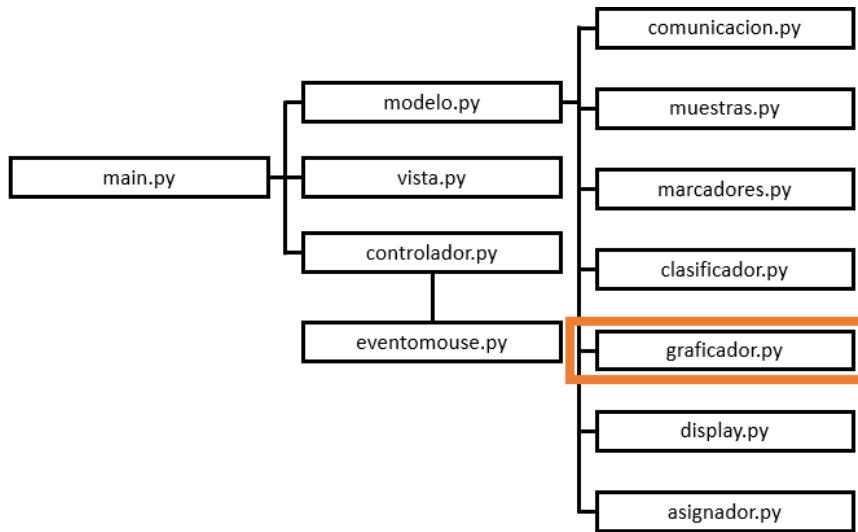


Figura 362. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

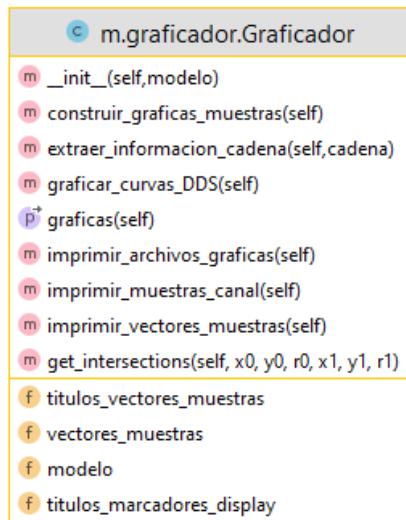


Figura 363. Métodos y atributos de la clase graficador.py.

Donde:

- (c) indica que es una **clase**.
- (m) indica que es un **método**.
- (p+) indica que es una **propiedad**.
- (f) indica que es un **atributo**.

```

# ****
#
# Clase: --> Graficador
# Modulo: -> graficador.py
#
# Descripción:
# - Graficar los datos obtenidos por arduino
# - Dibuar, mostrar informacion
# - Calcular algunos parametros
#
# Fecha: julio 28/2021
#
# ****

import numpy as np
import cv2

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.ticker import EngFormatter

from collections import deque

from engineering_notation import EngNumber
import math
import statistics

graficas = []

class Graficador():

    # -----
    #     CONSTRUCTOR
    # -----
    def __init__(self,modelo):

        print("")
        print(" CONSTRUCTOR: Clase: Graficador")
        self.modelo = modelo
        self.vectores_muestras = modelo.vectores_muestras
        self.titulos_vectores_muestras = modelo.titulos_vectores_muestras
        self.titulos_marcadores_display = modelo.titulos_marcadores_display

    def construir_graficas_muestras(self):

```

```

# --- Construccion AUTOMATICA de la graficas ---
titulos_marcadores = list(self.titulos_marcadores_display)

global graficas
graficas = []

muestras = self.vectores_muestras

for i in range(len(titulos_marcadores)):
    titulo_grafica = self.titulos_marcadores_display[titulos_marcadores[i]]
    nombre_vectores_grafica = self.extraer_informacion_cadena(titulo_grafica)

    # --- Construccion AUTOMATICA de cada GRAFICA, a partir de la
    #   informacion proporcionada en la clase Modelo en las estructuras:
    #   -> titulos_vectores_muestras <-> Vector
    #   -> titulos_marcadores_display <-> Diccionario
    #   Estas estructuras de datos se conectan con los nombres de los
    #   vectores que se encuentran en el Diccionario muestras
    #   -> vectores_muestras <-> Diccionario
    #
    plt.figure(i)
    # --- Graficas VECTOR A vs. muestras ---
    if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 1:
        plt.plot(muestras[nombre_vectores_grafica[0][0]],
                  label = nombre_vectores_grafica[0][0] + ' vs. muestras')
    # --- Graficas VECTOR A vs VECTOR B ---
    if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 2:
        plt.plot(muestras[nombre_vectores_grafica[0][0]],
                  muestras[nombre_vectores_grafica[0][1]],
                  label = nombre_vectores_grafica[0][0] + ' vs. ' +
                  nombre_vectores_grafica[0][1])
    # --- Graficas MULTIPLES tipo: VECTOR A vs VECTOR B ---
    if len(nombre_vectores_grafica) > 2:
        for g in range(len(nombre_vectores_grafica)):
            plt.plot(muestras[nombre_vectores_grafica[g][0]],
                      muestras[nombre_vectores_grafica[g][1]],
                      label = nombre_vectores_grafica[g][0] + ' vs. ' +
                      nombre_vectores_grafica[g][1])
    plt.title = 'Grafica ' + str(i)
    plt.ylabel = 'y label'
    plt.xlabel = 'x label'
    plt.legend()

```

```

# --- Guardado de las gráficas en:
#   1. "Archivo en disco [*.jpg] para su uso posterior
#      por parte de la Clase Display
#   2. El arreglo GLOBAL: 'graficas', el cual contiene
#      la ruta en donde se guardó la gráfica correspondiente
#
# -----
grafica = 'g/grafica_' + str(i) + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

def extraer_informacion_cadena(self,cadena):

    #print("")
    #print(" cadena original = ",cadena)

    # --- Separa grupos de letras ---
    subcadena = ""
    subtitulos = []
    ignorar_caracter = False
    num_char = 0
    for ch in cadena:
        num_char += 1
        # --- Ignorar los caracteres "espacio" [' '] y "coma" [,] ---
        if ch == ' ' or ch == ',':
            ignorar_caracter = True
        if num_char != 1:
            subtitulos.append(subcadena)
            subcadena = ""
        else:
            subcadena += ch
        if num_char == len(cadena): subtitulos.append(subcadena)

    # --- Genera tuplas de dos elementos ---
    tupla = []
    arreglo_tuplas = []
    for i in range(len(subtitulos)):
        # --- Detecta conector ['vs'] ó espacio [' '] --
        if subtitulos[i] == 'vs' or subtitulos[i] == " ":
            dummy = 0 # <-- No hacer nada ---
        else:
            tupla.append(subtitulos[i])
        if subtitulos[i] == " " or i == (len(subtitulos)-1):
            arreglo_tuplas.append(tupla)
            tupla = []

```

```

    return arreglo_tuplas

def graficar_curvas_DDS(self):

    global graficas
    graficas = []

    muestras = self.vectores_muestras
    titulos_vectores_muestras = self.titulos_vectores_muestras
    muestras_llaves = list(muestras)
    lim_max = len(muestras[muestras_llaves[0]])
    sel_filtro = 1

    # -----
    # Useful links:
    # Mark some points on data:
    # https://stackoverflow.com/questions/8409095/set-markers-for-individual-points-on-a-line-in-matplotlib
    # -----

    # --- Grafica 0 : Voltaje y binario 1 ---
    plt.figure(0)

    ax = plt.gca()
    formatter0 = EngFormatter(unit='V')
    ax.yaxis.set_major_formatter(formatter0)

    plt.subplot(2, 1, 1)
    plt.title("Voltaje en el tiempo")
    plt.plot(muestras['Voltaje'][0], label='Voltaje')
    plt.ylabel('Voltaje')

    plt.subplot(2, 1, 2)
    plt.plot(muestras['Binario'][0], label='Binario')
    plt.xlabel('Muestras')
    plt.ylabel('Binario')

    plt.legend()
    grafica = 'g/00_Voltaje_Binario_1' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

    # --- Grafica 1 : Voltaje y binario 2 ---

```

```

# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

cv2.putText(canvas2,"Tabla de estados binarios. Circuito DDS con PCF8591", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Numero          binario          (0-255)", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Voltaje ", (250, 100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0,
0), 1, cv2.LINE_AA)

# Generar tabla de valores voltaje-binario
for i in range(0,50,4):
    cv2.putText(canvas2,hex(int(muestras['Binario'][0][i])), (30,
100+(int(i/4)+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(muestras['Voltaje'][0][i]), (250,
100+(int(i/4)+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/01_Voltaje_Binario_2" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Grafica 2 : Voltaje y binario 3 ---
plt.figure(2)
ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)
plt.title("Voltajes en el tiempo")

# Generar arreglo con indices de muestras a marcar
markers_on = []
for i in range(0,len(muestras['Voltaje'][0]),4):
    markers_on.append( i)
#print('markers_on=' +str(markers_on))
plt.plot(muestras['Voltaje'][0],'o-', markevery=markers_on, label='Voltaje')

# Colocar la etiqueta correspondiente segun se comporte la grafica
for i in range(1,len(muestras['Voltaje'][0]),4):
    position =
    xoffset = 0

```

```

yoffset = 0
if muestras['Voltaje'][0][i] > 2.5:
    if (muestras['Voltaje'][0][i] - muestras['Voltaje'][0][i-1]) < 0:
        #print('mod1')
        position = 'left'
        xoffset = 0
        yoffset = 0
    else:
        #print('mod2')
        position = 'right'
        xoffset = -12
        yoffset = 0
else:
    if (muestras['Voltaje'][0][i-1] - muestras['Voltaje'][0][i]) < 0:
        #print('mod3')
        position = 'left'
        xoffset = 0
        yoffset = 0
    else:
        #print('mod4')
        position = 'right'
        xoffset = -10
        yoffset = -10

plt.annotate(str(muestras['Voltaje'][0][i-1])+'V,
'+hex(int(muestras['Binario'][0][i])),(i,muestras['Voltaje'][0][i-1]),textcoords="offset
points",xytext=(xoffset,yoffset),ha=position)

x1,x2,y1,y2=plt.axis()
plt.axis([-10,x2+3*x2/8,y1,5.5])
plt.ylabel('Voltaje')
plt.xlabel('Muestras')
plt.legend(['Voltaje, Binario'])
grafica = 'g/02_Voltaje_Binario_3' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 3 : Voltaje y binario 4 ---
# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

```

```

# --- Estados binarios ---

cv2.putText(canvas2,"Diagrama de numeros binarios con valores de voltaje
analogicos", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1,
cv2.LINE_AA)
#cv2.circle(canvas2, (320,260), 215,(255,0,0),2)
#cv2.circle(canvas2, (320,260), 185,(255,0,0),2)

angle = np.linspace(0,2*math.pi,16)

# Dibuja cada uno de los circulos y la informacion voltaje-binario que va dentro de ella.
Tambien dibuja las flechas entre circulos
for i in range(0,15):
    # Dibuja el circulo
    xcoor = int(320+185*math.cos(angle[i]))
    ycoor = int(260+185*math.sin(angle[i]))
    cv2.circle(canvas2, (xcoor,ycoor), 30, (0,0,0), 2)
    cv2.putText(canvas2,hex(int((muestras['Binario'][0][int((50/16)*i)]))), (xcoor-
22,ycoor-5),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(muestras['Voltaje'][0][int((50/16)*i)])+'V', (xcoor-
22,ycoor+15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    # Encuentra las coordenadas de la flecha entre dos circulos
    i1 = self.get_intersections(320,260,185,xcoor,ycoor,30)
    i2 = self.get_intersections(320,260,185,int(320+185*math.cos(angle[i+1])),int(260+185*math.si
n(angle[i+1])),30)
    dist = 100
    flecha_coords = [0]*4
    "for j in range(0,1):
        if math.dist([i1[j]*2],i1[j]*2+1],[i2[j]*2],i2[j]*2+1]) < dist:
            print('1, j=' +str(j))
            dist = math.dist([i1[j]*2],i1[j]*2+1],[i2[j]*2],i2[j]*2+1])
            flecha_coords[0] = i1[j]*2
            flecha_coords[1] = i1[j]*2+1
            flecha_coords[2] = i2[j]*2
            flecha_coords[3] = i2[j]*2+1]
        if math.dist([i1[(2-j)*2]],i1[(2-j)*2+1],[i2[j]*2],i2[j]*2+1]) < dist:
            print('2, j=' +str(j))
            dist = math.dist([i1[j]*2],i1[j]*2+1],[i2[j]*2],i2[j]*2+1])
            flecha_coords[0] = i1[(2-j)*2]
            flecha_coords[1] = i1[(2-j)*2+1]
            flecha_coords[2] = i2[j]*2
            flecha_coords[3] = i2[j]*2+1]"

```

```

# Se puede comprobar con el codigo comentado que esta es la solucion que
proporciona las coordenadas correctas
flecha_coords[0] = i1[2]
flecha_coords[1] = i1[3]
flecha_coords[2] = i2[0]
flecha_coords[3] = i2[1]

#print('coordenadas de la flecha: '+str(flecha_coords))
#print('distancia: '+str(dist))

# Dibuja la flecha
canvas2 =
cv2.arrowedLine(canvas2,(int(flecha_coords[0]),int(flecha_coords[1])),(int(flecha_coords[2])
),int(flecha_coords[3])),(0,0,0), 3, tipLength = 0.5)

filename = "g/03_Voltaje_Binario_4" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Auxiliar para depuracion ---
plt.show()

@property
def graficas(self):
    return graficas # --- Nombre de la grafica y ruta donde se guarda ---

def imprimir_archivos_graficas(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" --- Archivos de imagenes de graficas *.jpg ---")
    print("*****")
    for g in graficas:
        print("")
        print(g)

def imprimir_muestras_canal(self):

    print(" ")
    print(" En CLASE Graficador:")
    print("*****")
    print(" --- Vectores de muestras por canal ---")
    print("*****")

```

```

# --- ESTRUCTURA del diccionario:
#   datos = {canal: i, llaveMuestras[i]: vector de datos}
# -----
muestras = self.vectores_muestras
llaves = list(muestras)
print(" llaves = ",llaves)
for k in range(len(llaves)):
    print("")
    print(" --- ",llaves[k],": ",muestras[llaves[k]])

def imprimir_vectores_muestras(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" -- Muestras leidas de la Clase Modelo --")
    print("*****")
    muestras = self.vectores_muestras
    llaves = self.titulos_vectores_muestras
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k],": ",muestras[llaves[k]])

    # -----
    # Obtiene las intersecciones dadas dos circunferencias, coordenadas de su centro y
    radio.
    # Tomado de https://stackoverflow.com/questions/55816902/finding-the-intersection-of-
two-circles
    # -----
    def get_intersections(self, x0, y0, r0, x1, y1, r1):
        # circle 1: (x0, y0), radius r0
        # circle 2: (x1, y1), radius r1

        d = math.sqrt((x1-x0)**2 + (y1-y0)**2)

        # non intersecting
        if d > r0 + r1 :
            return None
        # One circle within other
        if d < abs(r0-r1):
            return None
        # coincident circles
        if d == 0 and r0 == r1:

```

```
    return None
else:
    a=(r0**2-r1**2+d**2)/(2*d)
    h=math.sqrt(r0**2-a**2)
    x2=x0+a*(x1-x0)/d
    y2=y0+a*(y1-y0)/d
    x3=x2+h*(y1-y0)/d
    y3=y2-h*(x1-x0)/d

    x4=x2-h*(y1-y0)/d
    y4=y2+h*(x1-x0)/d

return (x3, y3, x4, y4)
```

Figura 364. Código completo de la clase graficador.py

11.2.3 Circuito amplificador de una etapa (c)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d)**. Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el **circuito Amplificador de una etapa (c)**, se presenta lo siguiente:

1. Diagrama esquemático del circuito.
2. Programa de Arduino completo.
3. De los programas en Python:
 - i. Clase calculadora_datos_graficas.py.
 - ii. Clase graficador.py.

11.2.3.1 Diagrama esquemático (1)

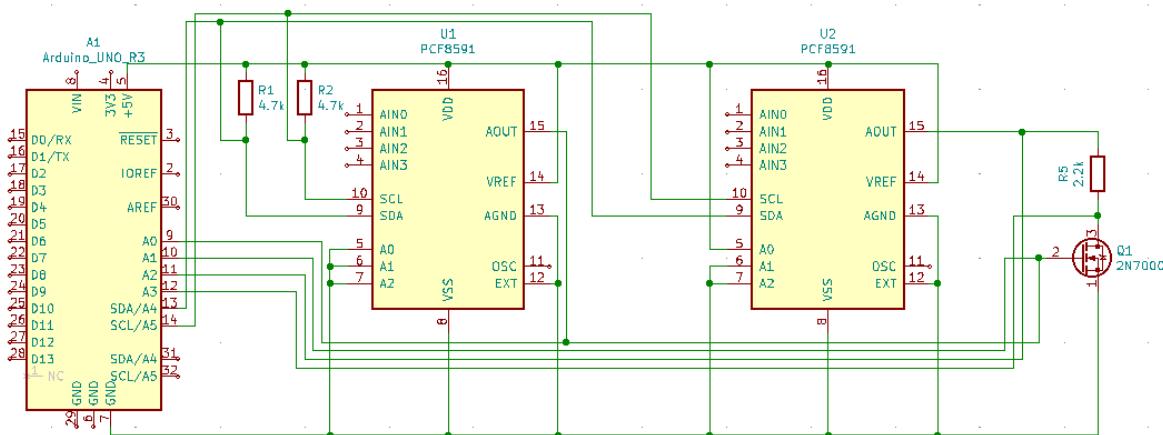


Figura 365. Diagrama esquemático completo del **circuito Amplificador de una etapa (c)**.

11.2.3.2 Programa de Arduino (2)

Programa de Arduino para el Amplificador de una etapa
Nombre del programa: Arduino AR 65 MOSFET

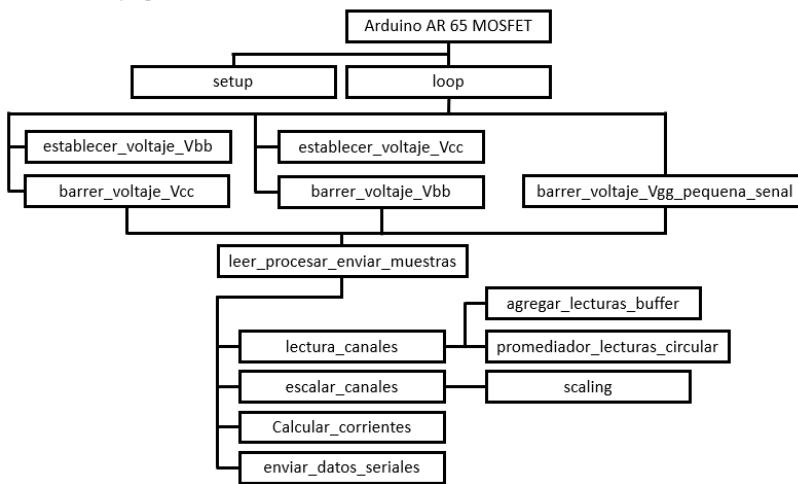


Figura 366. Diagrama a bloques del programa completo en Arduino para adquirir señales del circuito **Amplificador de una Etapa (c)**.

```
/*
PROGRAMA: 28072021

Trazador de curvas para un transistor MOSFET utilizando DACs

Este programa se ejecuta en conjunto con:

ARDUINO: ARDUINO AR 65 MOSFET Amplificador.ino (entradas) 08062021-A
PYTHON: Completo AR 65 MOSFET.py (monitor) 08062021-P

TARJETA UTILIZADA: Arduino MEGA
TRANSISTORES DE PRUEBA: 2N7000
CONVERTIDORES DIGITALES ANALOGICOS UTILIZADOS: PCF8591

28 / julio / 2021

*/
#include "Wire.h"
#define PCF_Vgg 0x48 // I2C bus address
```

```

#define PCF_Vdd 0x49 // I2C bus address

// *****
// SELECCION DE MUESTRAS CON ó SIN 'Filtrado Circular'
// 'true' <- CON Filtrado || circular de 4 muestras
// 'false' <- SIN Filtrado
// *****
boolean Filtrado_Circular = false;
// *****

/**/
* -----
* Declaracion de arreglos:
* numero_entradas_analogicas -> es el numero de entradas analogicas que
se leen del circuito
* canales_medidos[4] -> son los voltajes analogicos medidos directamente
del Arduino [0,1023]
* voltajes[4] -> son los voltajes escalados del arreglo anterior [0,5]
* corrientes[2] -> son las corrientes calculadas como la diferencia del
arreglo de voltajes anterior
* -----
*/
const int numero_entradas_analogicas = 4;
int canales_medidos[numero_entradas_analogicas];
double voltajes[numero_entradas_analogicas];
double corrientes[2];

/**/
* -----
* Definicion de variables para el cálculo del FILTRO CIRCULAR
(Promediador)
* Este filtro permite promediar cada lectura analogica de arduino
mediante un filtro circular
* buffer_promediador -> Es el buffer en el que se almacenan las
muestras leidas
* bufferSize -> Es el tamaño del buffer
* canales -> Es el numero de canales a medir
* -----
*/
const int bufferSize = 32; // Numero de muestras a promediar
int promedios_calculados[numero_entradas_analogicas];
int buffer_promediador[numero_entradas_analogicas][bufferSize];
int index[numero_entradas_analogicas];
int canales[numero_entradas_analogicas];

/**/
* -----
* Definicion de pines para los 4 canales del ADC
* PinVgg -> Es el voltaje a la salida del PCF_Vgg
* PinVdd -> Es el voltaje a la salida del PCF_Vdd

```

```

 * PinVgs -> Es el voltaje en base-emisor del transistor MOSFET
 * PinVds -> Es el voltaje en colector-emisor del transistor MOSFET
 *
 * NOTA: Entre Vgg y Vgs no hay resistencia
 *       Entre Vdd y Vds hay una resistencia Rc de 2.2 kohm (valor
nominal)
 * -----
-----
 * /
const int PinVgg = A0; // Pin de entrada analogica canal 0
const int PinVgs = A1; // Pin de entrada analogica canal 1
const int PinVdd = A2; // Pin de entrada analogica canal 2
const int PinVds = A3; // Pin de entrada analogica canal 3

/***
 * -----
-----
 * Variables para el control del tiempo de muestreo
 * lastime -> guarda el ultimo instante de tiempo en el que enviaron
datos por el puerto serie
 * sampleTime -> duracion entre cada muestra tomada
 * numMuestras -> numero de muestras tomadas por cada curva
 *
 * Por lo tanto, la frecuencia de cada conjunto de muestras es de 2Hz
(numMuestras x sampleTime)/1000
 * -----
-----
 * /
unsigned long lastTime = 0;
unsigned long sampleTime = 100;
const double numMuestras = 50;

/***
 * -----
-----
 * Vdd -> valor medido de la fuente de voltaje que se utiliza como
alimentacion del trazador de curvas
 * Vth -> valor observado experimentalmente de la curva Id vs Vgs en
donde se observa que comienza a crecer
 *           el voltaje Vgs con respecto a la corriente Id (voltaje de
umbral)
 *
-----
-----
 * /
const double Vdd=5.0;
const double Vth=1.8;

/***
 * -----
-----
 * i_muestra -> Indica el numero de la muestra que se toma. Para cada
conjunto de muestras, puede variar
 *           entre [0,numMuestras]
 * numero_curva -> Indica la curva de la cual se toman muestras. Las
curvas con numero_curva = 0 hasta 3,
 *           se utilizan para graficar Id vs Vds. La curva con
numero_curva = 4 se utiliza para

```

```

/*
 *           graficar las demas curvas
 * barrido_recta -> Realiza un barrido con un numero de muestras igual a
numMuestras para cada curva medida
 * voltaje_obtencion_curva -> Valor de voltaje propuesto Vdd en formato
hexadecimal para las curvas Id vs Vds.
 *
 *                               Posteriormente, este mismo valor es Vgg
para las demas curvas
* -----
-----
*/
int i_muestra=0;
int numero_curva=0;
byte barrido_recta;
byte voltaje_obtencion_curva = 0x74;
byte sine=0;

/***
 * -----
-----
 * num_curvas_id_vds -> Indica el numero de curvas para las que se
tomaran muestras para graficas id vs Vds
 * num_curvas_parametros -> Indica el numero de curvas para las que se
tomaran muestras para calcular parametros hibridos
 * num_curvas_pequena_señal -> Indica el numero de curvas para las que se
tomaran muestras para graficar curvas del transistor
 *
 *                               como amplificador en pequena señal
 * -----
-----
*/
const int num_curvas_id_vds = 5;
const int num_curvas_parametros = 1;
const int num_curvas_pequena_señal = 1;

/***
 * Inicializa tanto la comunicacion serial como la comunicacion I2C para
comunicarse con los dos PCF8591
 *
 */
void setup() {
  Wire.begin();
  // --- Inicializa velocidad de transmision serial ---
  Serial.begin(9600);

  // --- Sincronizacion de comunicacion serial ---
  Serial.println("inicio");
}

/***
 * Se miden cinco curvas para graficar Id vs vds. Posteriormente se toma
una curva para Id vs Vgs y una ultima curva para el amplificador en
pequeña señal
 *
 */
void loop() {
  // --- Toma muestras cada sampleTime ms ---
  if (millis()-lastTime > sampleTime) {

```

```

lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
tomaron muestras

if(numero_curva<num_curvas_id_vds) // ¿Se toman muestras de las
primeras "num_curvas_id_vds" curvas para Id vs Vds?
{
    int valor;
    // --- 5 Curvas para Id vs Vds [MOSFET] ---
    valor = map(numero_curva,0,5,102,125); //probar tambien con 116 -
132

    establecer_voltaje_Vgg(valor);
    barrer_voltaje_Vdd();
}
else if(numero_curva<num_curvas_id_vds+num_curvas_parametros)
{
    // --- 1 Curva para Ig vs Vgs , Id vs Vgs , entre otras ---
    establecer_voltaje_Vdd(255);
    barrer_voltaje_Vgg();
}
else
if(numero_curva<num_curvas_id_vds+num_curvas_parametros+num_curvas_id_vds
+num_curvas_pequena_senal)
{
    // --- 1 Curva para pequena senal ---
    establecer_voltaje_Vdd(255);
    barrer_voltaje_Vgg_pequena_senal(2.2,0.075);
}
else
{
    numero_curva = 0;
}
}

/***
 * Actualiza el valor del voltaje Vgg en el PCF8591 correspondiente
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
*/
void establecer_voltaje_Vgg(int valor){
Wire.beginTransmission(PCF_Vgg); // wake up PCF_Vgg
Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
Wire.write(valor);
Wire.endTransmission(); // end transmission
}

/***
 * Actualiza el valor del voltaje Vgg en el PCF8591 correspondiente
 * @param valor -> Es un numero en formato hexadecimal [0,255] que
representa un voltaje [0,5V]
*/
void establecer_voltaje_Vdd(int valor){
Wire.beginTransmission(PCF_Vdd); // wake up PCF_Vdd

```

```

    Wire.write(0x40); // control byte - turn on DAC (binary 1000000)
    Wire.write(valor);
    Wire.endTransmission(); // end transmission
}

/***
 * Actualiza Vdd con diferentes valores de entre 0V y 5V
 * Despues envia por el puerto serie las muestras leidas
 */
void barrer_voltaje_Vdd(){
    // --- Barre durante 50 muestras ---
    for (int i_muestra=0; i_muestra<250; i_muestra+=5){
        establecer_voltaje_Vdd(i_muestra);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

/***
 * Actualiza Vgg con diferentes valores de entre 0V y 5V
 * Despues envia por el puerto serie las muestras leidas
 */
void barrer_voltaje_Vgg(){
    // --- Transistor MOSFET: valores de Vgg mayores a Vth ---
    for (int i_muestra=0; i_muestra<150; i_muestra+=3){
        establecer_voltaje_Vgg(i_muestra);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

/***
 * Actualiza Vgg con diferentes valores de una senoidal con amplitud
 * pequena
 * @param Voffset -> Es el voltaje de offset de la senal senoidal
 * @param Amplitud_pico -> Es el valor de amplitud pico de la senoidal
 */
void barrer_voltaje_Vgg_pequena_senal(double Voffset, double
Amplitud_pico){
    for (int i_muestra=0; i_muestra<250; i_muestra+=5){
        sine =
byte(255*((Voffset/Vdd)+((Amplitud_pico/(Vdd))*sin((1000/sampleTime)*TWO_
PI*i_muestra/(numMuestras-1)))));
        establecer_voltaje_Vgg(sine);
        leer_procesar_enviar_muestras();
    }
    numero_curva++; // Indica que se tomaran muestras de la siguiente curva
}

/***
 * Lee los canales analogicos del circuito Vdd, Vds, Vgg y Vgs (4
voltajes)
 * Escala los canales al voltaje de la fuente de alimentacion
 * Calcula las corrientes a partir de la resta de voltajes

```

```

 * Envia los datos de todos los canales por el puerto serie
 *
 */
void leer_procesar_enviar_muestras(){
    // --- Lectura de los canales ---
    leer_canales(); // Lee los 4 voltajes del circuito

    // --- Escalamiento de canales ---
    escalar_canales(Vdd); // Escala los 4 voltajes leidos al voltaje de
alimentacion Vdd

    // -- calculo de corrientes
    calcular_corrientes(); // Calcula las corrientes con los voltajes
escalados

    // --- Transmision de cada valor al programa en Python ---
    enviar_datos_serials(); // Envia los 4 voltajes escalador y las 2
corrientes calculadas
}

/**
 * Lee los 4 voltajes del circuito. [Vgg,Vgs,Vdd,Vds]
 *
 */
void leer_canales()
{
    if (Filtrado_Circular)
    {
        // --- CON FILTRADO CIRCULAR con buffer circular de 32 muestras ---
        for (int b=0; b<bufferSize; b++)
        {
            canales_medidos[0] = analogRead(PinVgg); // Vgg
            canales_medidos[1] = analogRead(PinVgs); // Vgs
            canales_medidos[2] = analogRead(PinVdd); // Vdd
            canales_medidos[3] = analogRead(PinVds); // Vds
            // Filtrado circular en cada lectura. SIN mezclar muestras en
tiempos diferentes
            agregar_lecturas_buffer(canales_medidos);
            promediador_lecturas_circular(canales_medidos);
        }
    } else {
        // --- SIN FILTRADO CIRCULAR ---
        canales_medidos[0] = analogRead(PinVgg); // Vgg
        canales_medidos[1] = analogRead(PinVgs); // Vgs
        canales_medidos[2] = analogRead(PinVdd); // Vdd
        canales_medidos[3] = analogRead(PinVds); // Vds
        for(int j=0; j<numero_entradas_analogicas; j++)
        {
            canales[j] = canales_medidos[j];
        }
    }
}

/**
 * Escala los 4 canales de voltaje, que varian entre [0,1023] a un numero
que representa el voltaje [0,5]
*/

```

```

/*
void escalar_canales(double Vdd_ref)
{
    for(int i=0;i<numero_entradas_analogicas;i++)
    {
        voltajes[i] = scaling(canales[i], 0, 1023, 0, Vdd_ref); // Vgg,Vbs,Vdd,Vds
    }
}

/**
 * Calcula las corrientes con los valores de voltaje escalados [Ig,Id]
 * NOTA: Las corrientes Ig e Id no estan escaladas al valor medido de las
resistencias Rg y Rd.
*           Solo son una diferencia de voltajes. Ig = Vgg - Vgs
*                               Id = Vdd - Vds
*
*/
void calcular_corrientes()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        corrientes[i] = (voltajes[i*2]-voltajes[i*2+1]);
    }
}

/**
 * Envia todos los datos por el puerto serial
 * En orden, se envian: [Vgg,Vgs,Ig,Vdd,Vds,Id]
*
*/
void enviar_datos_seriales()
{
    for(int i=0;i<(numero_entradas_analogicas/2);i++)
    {
        Serial.println(voltajes[i*2],4); //Vgg,Vdd
        Serial.println(voltajes[i*2+1],4); //Vgs,Vds
        Serial.println(corrientes[i],4); //Ig,Id
    }
}

/**
 * Escala valores que se sabe que varian entre un intervalo definido
[in_min,in_max] a otro intervalo [out_min,out_max]
* @param x -> es el valor a escalar
* @param in_min -> es el valor minimo que puede tomar x antes del
escalado
* @param in_max -> es el valor maximo que puede tomar x antes del
escalado
* @param out_min -> es el valor minimo al cual se escala x
* @param out_max -> es el valor maximo al cual se escala x
*
*/
float scaling(float x, float in_min, float in_max, float out_min, float
out_max)
{
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

```

```

}

// *****
// FILTRO CIRCULAR (promediador) con buffer de 32 muestras
//
// *****
// --- Inicializacion del buffer ---
void agregar_lecturas_buffer(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        buffer_promediador[i][index[i]] = canales_medidos[i];
        index[i] += 1;
        if (index[i] >= bufferSize) index[i] = 0;
    }
}

// --- Filtro circular: promediador ---
void promediador_lecturas_circular(int canales_medidos[])
{
    for(int i=0; i<numero_entradas_analogicas; i++)
    {
        long sum = 0;
        for (int k=0; k<bufferSize; k++)
        {
            sum += buffer_promediador[i][k];
        }
        canales[i] = (int)(sum/bufferSize);
    }
}

```

Figura 367. Programa completo de Arduino

11.2.3.3 Programas en Python (3)

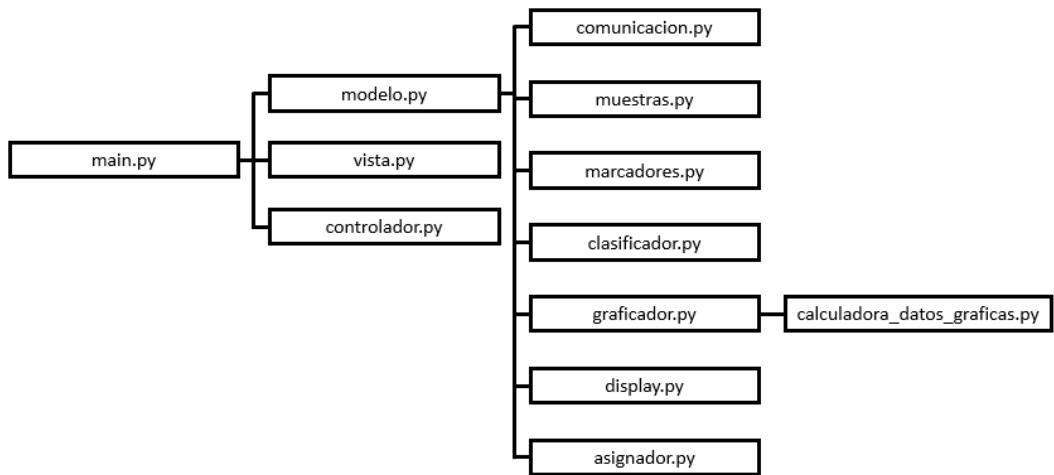


Figura 368. Diagrama de clases para el circuito **Amplificador de una etapa (c)**.

11.2.3.3.1 Clase calculadora_datos_graficas.py (3i)

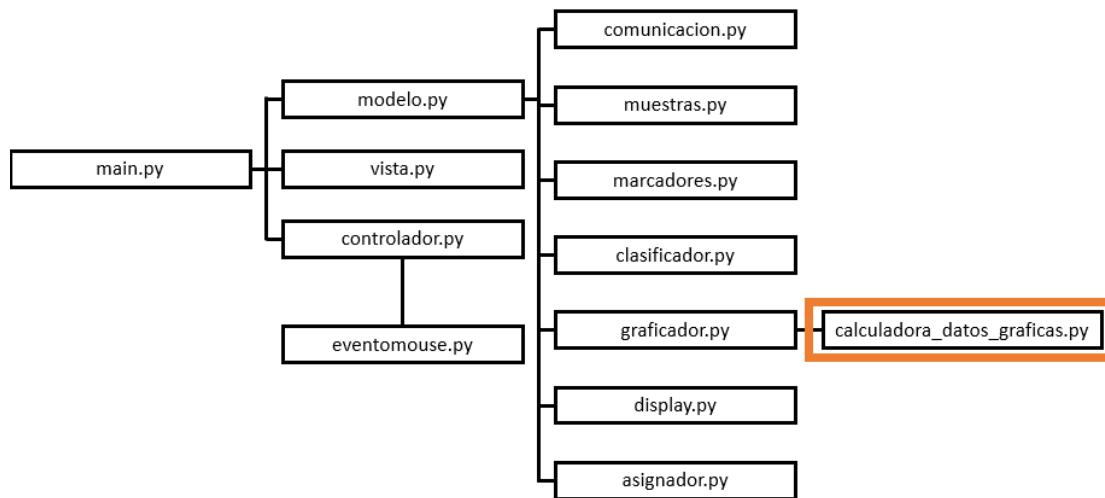


Figura 369. Ubicación de la clase calculadora_datos_gráficas.py.

Esta clase contiene los siguientes métodos y atributos:

```
 ④ m.calculadora_datos_graficas.Calculadora_datos_graficas
m __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes,num_curvas,id_vds,num_curvas_parametros,num_curvas_pequena_senal,Rd,Rg)
m estimate_coef(self, x, y)
m plot_regression_line(self, x, y, b)
m construir_series_datos(self)
m obtener_ind_min(self,ind_serie_datos)
m obtener_ind_max(self,ind_serie_datos,ind_min)
m regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max)
m inicializar_excel_datos(self)
m guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId, voltajeVds)
m finalizar_excel_datos(self,writer)
m leer_datos_excel(self,ind_variables, i)
m calcular_lambda(self, linea, lim_y)
m calcular_Vth_Kn_ids(self)
m ajustar_id_con_resistencias(self)
m calcular_gm(self, Kn)
m calcular_resistencias_hibridas(self,MOS_avg_lambda,corrientes,gm)
m puntos_sobre_grafica_ids,ygs(self, Vth, ind_ids)
m calcular_puntos_límite_grafica_ids_vds(self, Vth)
m calcular_punto_operacion_y_parametros_hibridos(self,Kn,MOS_avg_lambda)
m calcular_punto_operacion_puntos_min_max_ganancia_experimental(self)
m calcular_puntos_límite_y_operacion_sobre_grafica_ids_vds(self,x,y,v,op,i,op)
m calcular_posiciones_label(self,ind_serie,ind_muestra)
f num_curvas_id_vds
f muestras_llaves
f Rd
f Rg
f num_curvas_pequena_senal
f sel_filtro
f ind_voltajes
f ind_corrientes
f muestras
f num_curvas_parametros
f modelo
f lim_max
```

Figura 370. Métodos y atributos de la clase calculadora_datos_graficas.py.

Donde:

- ④ indica que es una **clase**.
- m indica que es un **método**.
- p+ indica que es una **propiedad**.
- f indica que es un **atributo**.

```

#-----
#
# Clase: --> Calculadora_datos_graficas
# Modulo: -> calculadora_datos_graficas.py
#
# Descripción:
# Auxiliar de graficador.py que calcula los datos de las series de datos de las graficas a
mostrar
#
# Fecha: julio 28/2021
#
# ****
# -----
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import numpy as np
import math
import statistics
from engineering_notation import EngNumber

class Calculadora_datos_graficas:
    # -----
    # CONSTRUCTOR
    # -----
    def __init__(self,modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes,ind_voltajes,num_c
urvulas_id_vds,num_curvas_parametros,num_curvas_pequena_senal,Rd,Rg):

        print("")
        print(" CONSTRUCTOR: Clase: Calculadora_datos_graficas")
        self.modelo = modelo
        self.muestras = modelo.vectores_muestras
        self.muestras_llaves = muestras_llaves
        self.lim_max = lim_max
        self.sel_filtro = sel_filtro
        self.ind_corrientes = ind_corrientes
        self.ind_voltajes = ind_voltajes
        self.num_curvas_id_vds = num_curvas_id_vds
        self.num_curvas_parametros = num_curvas_parametros
        self.num_curvas_pequena_senal = num_curvas_pequena_senal
        self.Rd = Rd
        self.Rg = Rg

    # -----

```

```

# Calcula los coeficientes b0 y b1 del método de regresión lineal simple
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def estimate_coef(self, x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

# -----
# Grafica el resultado de la regresión lineal simple
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def plot_regression_line(self, x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

#

```

```

# Construye las series de datos de las diferentes curvas Id vs Vds (ej. Id1, Vd1, Id2, Vds2,
Id3, Vds3 ...)
#
# -----
def construir_series_datos(self):

    muestras = self.muestras
    ind_variables = self.ind_variables
    writer = self.inicializar_excel_datos()

    valores_id_vds = {i:[] for i in ind_variables}

    #print(ind_variables)
    #print('\n')
    #print(valores_id_vds)

    for i in range(0,int(len(ind_variables)),2):
        i_min = 50*int(i/2)
        i_max = 50*(int(i/2)+1)

        #print('valores_id_vds['+str(ind_variables[i])+']
        muestras[Id][i_min:i_max])
        #print('valores_id_vds['+str(ind_variables[i+1])+']
        muestras[Vds][i_min:i_max])

        valores_id_vds[ind_variables[i]] = muestras['Id'][i_min:i_max]
        valores_id_vds[ind_variables[i+1]] = muestras['Vds'][i_min:i_max]

    self.guardar_datos_excel(writer,ind_variables[i+1],ind_variables[i],valores_id_vds[ind_variables[i]],valores_id_vds[ind_variables[i+1]])
        #print(valores_id_vds[ind_variables[i]])
        #print(valores_id_vds[ind_variables[i+1]])
        #print('\n')

    self.finalizar_excel_datos(writer)

    return valores_id_vds

# -----
# Obtiene el límite mínimo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de corriente Id
# Esto es la primera muestra cuya diferencia con la anterior es menor a 0.06 y que el
índice de esa muestra sea mayor a la muestra número 2

```

```

# -----
def obtener_ind_min(self,ind_serie_datos):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    Rd = self.Rd

    ind_min=0;
    id_cambio = [x for x in
range(0,len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))]

    for i in range(1,30):
        id_cambio[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i]
        muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1] = -print('i=' +str(i)+':'
'+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i])+'-
'+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][i-1])+'-
'+str(id_cambio[i]))
        #print('i: '+str(i)+', '+str(id_cambio[i])+': '+str(id_cambio[i]) < 0.06))
        if id_cambio[i] < (0.06)/Rd :
            #print('i: '+str(i)+', '+str(id_cambio[i])+': '+str(id_cambio[i]) < 0.06))
            if i > (2+int(ind_serie_datos/2)+1) :
                ind_min = i
                break

    #print('ind_min: id_cambio['+str(ind_min)+']: '+str(id_cambio[ind_min]))

    return ind_min

# -----
# Obtiene el límite máximo en donde se considera recta la parte de la curva
correspondiente Id# vs Vds# a partir de los valores de voltaje Vds
# Esto es la primera muestra cuya diferencia con la anterior es mayor a 0.05 y que el
índice de esa muestra sea mayor a la muestra número 20
# -----
def obtener_ind_max(self,ind_serie_datos,ind_min):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes

```

```

ind_max=len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos])-1
    vds_cambio = [x for x in
range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos))])
    #print('Vds_cambio len: '+str(len(vds_cambio)))
    for i in
range(20,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos])):
        vds_cambio[i]=
        muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i]]-
        muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i-1]]
        #print('i='+str(i)+':
        '+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i]])+
        '+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos][i-1]])+
        '+str(vds_cambio[i]))
        #print('i: '+str(i)+', '+str(vds_cambio[i])+': '+str(vds_cambio[i] < 0.06))
        if vds_cambio[i] <= 0.05 :
            #print('i: '+str(i)+', '+str(vds_cambio[i])+': '+str(vds_cambio[i] < 0.05))
            if i > ind_min :
                ind_max = i-1
                break

#print('ind_max: vds_cambio['+str(ind_max)+']: '+str(vds_cambio[ind_max]))

return ind_max

# -----
# Funcion que obtiene lambda para un transistor mosfet
# Tomado de https://www.geeksforgeeks.org/linear-regression-python-implementation/
# -----
def regresion_lineal_lambda(self, ind_serie_datos, ind_min, ind_max):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes

    #print(valores_id_vds[ind_variables[ind_serie_datos+1]][ind_min:ind_max])
    #print(valores_id_vds[ind_variables[ind_serie_datos]][ind_min:ind_max])

    x =
np.array(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][[ind_serie_datos]][ind_min:ind_max]).reshape(-1)

```

```

y
np.array(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][ind_mi
n:ind_max]).reshape(-1)

#print(x)
#print(y)

b = self.estimate_coef(x,y)
#print("Estimated coefficients: b_0 = {} b_1 = {}".format(b[0], b[1]))
#plot_regression_line(x, y, b)

return -b[0] / b[1]

# --- Para depuracion
#
# Inicializa la escritura de datos en Excel
#
# -----
def inicializar_excel_datos(self):
    writer = pd.ExcelWriter('mediciones_promedio.xlsx', engine='xlsxwriter')
    return writer

# -----
# Guarda los valores de Id y Vds en Excel en Excel
#
# -----
def guardar_datos_excel(self,writer, nombre_hoja_id, nombre_hoja_vds, corrienteId,
voltajeVds):
    muestras_seriesId = pd.Series(corrienteId)
    muestras_seriesVds = pd.Series(voltajeVds)

    muestras_seriesId.to_excel(writer, sheet_name=nombre_hoja_id)
    muestras_seriesVds.to_excel(writer, sheet_name=nombre_hoja_vds)

# -----
# Finaliza la escritura de datos de Excel
#
# -----
def finalizar_excel_datos(self,writer):
    writer.save()

# -----
# Lee datos de Excel
#

```

```

# -----
def leer_datos_excel(self,ind_variables, i):
    df = pd.read_excel (r'C:\Users\ferda\Desktop\Material del profe 100421\3 Sistema
Completo Realidad Aumentada\AR_multiples_graficas_Fernando_120421\Completo AR
20\mediciones_promedio_10.xlsx',
                           sheet_name=ind_variables[i],
                           usecols=[ind_variables[i]])
    return df

#-----
# -----
# FUNCION PRINCIPAL que calcula lambda de un transistor mos
# Obtiene primero los índices mínimos y máximos en donde se considera "recta" a la
curva Id vs Vds
# Calcula lambda para esa curva
# Obtiene el promedio de todas las lambdas calculadas
# -----
def calcular_lambda(self,linea_lim_y):

    MOSFET_lambdas = []
    muestras = self.muestras
    lim_max = self.lim_max
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_id_vds = self.num_curvas_id_vds

    #valores_id_vds = self.construir_series_datos()

    # Depuracion con datos leidos de excel
    """
    for i in range(1,6):
        ind_variables.append('Id'+str(i))
        ind_variables.append('Vds'+str(i))

    for i in range(len(ind_variables)):
        df = self.leer_datos_excel(ind_variables, i)
        valores_id_vds[ind_variables[i]] = df.values"""

    ind_min = []
    ind_max = []

    #print(valores)
    #print(valores['Id1'][0])

```

```

# --- Calculo de lambda para cada curva ids-vds
for [ind_serie_datos,i] in
zip(num_curvas_id_vds,range(1,len(num_curvas_id_vds)+1)):
    #print('Id'+str(ind_serie_datos)+', '+Vds'+str(ind_serie_datos))
    #print('muestras[Id][ind_serie_datos]:
    '+str(len(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]))+
    '+muestras[Vds][ind_serie_datos]):
    '+str(len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos])))
        #print('Id'+str(ind_serie_datos)+': ')
        #print(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos])
        #print('Vds'+str(ind_serie_datos)+': ')
        #print(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos])

    # --- Calculo de indice minimo y maximo para despues realizar regresion lineal y
obtener la interseccion con el eje de vds negativo. El resultado es lambda ---

ind_min.append(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos]
.index(linea_lim_y[i]))
    print('ind_min: '+str(ind_min[len(ind_min)-1])+': '+Id'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][ind_serie_datos][ind_min[le
n(ind_min)-1]]))

    ind_max.append(self.obtener_ind_max(ind_serie_datos,ind_min[len(ind_min)-1]))
    print('ind_max: '+str(ind_max[len(ind_max)-1])+': '+Vds'+str(ind_serie_datos)+',
'+str(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][ind_serie_datos][ind_max[le
n(ind_max)-1]]))

MOSFET_lambdas.append(self.regresion_lineal_lambda(ind_serie_datos,
ind_min[len(ind_min)-1], ind_max[len(ind_max)-1]))
    print('lambda: '+str(MOSFET_lambdas[ind_serie_datos])+'\n')

print('lambda_promedio: '+str(statistics.mean(MOSFET_lambdas)))
#return statistics.mean(MOSFET_lambdas)
return ind_min,ind_max,MOSFET_lambdas

#
# -----
# Calcula los parametros Vth, Kn e Idss del transistor MOSFETT
# Ademas, calcula los indices de ids-vgs que se utilizaron para calcular estos parametros
(Vth,Kn,Idss)
#
def calcular_Vth_Kn_Idss(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves

```

```

sel_filtro = self.sel_filtro
ind_corrientes = self.ind_corrientes
ind_voltajes = self.ind_voltajes
num_curvas_parametros = self.num_curvas_parametros

# --- Se calcula la maxima variacion de ids, entre muestras adyacentes ---
ind_ids = []
delta_ids = [0]*50
for i in range(1,50-1):
    delta_ids[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][i] - muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][i-1]
    #delta_vgs = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][i] - muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][i-1]
    #print(['+'+str(i)+']' 'delta_ids: '+str(delta_ids[i])#+', delta_vgs: '+str(delta_vgs))

# --- Se identifica el indice de la maxima variacion y el indice anterior a esta muestra -
--=
ids_max = np.max(delta_ids)
ind_ids.append(delta_ids.index(ids_max)-1)
ind_ids.append(delta_ids.index(ids_max))
#print('ind_ids: '+str(ind_ids))

# --- Se calcula la diferencia de vgs y de ids, con los indices anteriores ---
vgs_1 = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][ind_ids[0]]
vgs_2 = muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][ind_ids[len(ind_ids)-1]]
ids_1 = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][ind_ids[0]]
ids_2 = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][[num_curvas_parametros[0]]][ind_ids[len(ind_ids)-1]]

#print('ids_1: '+str(ids_1))
#print('ids_2: '+str(ids_2))
#print('vgs_1: '+str(vgs_1))
#print('vgs_2: '+str(vgs_2))

# --- Se calcula vth, kn e idss ---

```

```

Vth = (vgs_1-vgs_2*math.sqrt((ids_1)/(ids_2)))/(1-math.sqrt((ids_1)/(ids_2)))
Kn = (ids_1)/((vgs_1-Vth)**2)
Idss = Kn * Vth**2

print('Vth_calculado: '+str(Vth))
print('Kn_calculada: '+str(Kn))
print('Idss_calculada: '+str(Idss))

return (Vth, Kn, Idss, ind_ids)

# -----
# Ajusta el valor de todas las corrientes, puesto que solo son una diferencia de voltajes
(ej. Id=Vdd-Vds ; Ig=Vgg-Vgs)
#
# -----
def ajustar_id_con_resistencias(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    ind_corrientes = self.ind_corrientes
    lim_max = self.lim_max
    Rd = self.Rd
    Rg = self.Rg

    for i in range(ind_corrientes[1],lim_max*3,6):
        #print('Serie '+str(muestras_llaves[i])+' por ajustar')
        for j in range(0,lim_max):
            muestras[muestras_llaves[i]][j] = [x / Rd for x in muestras[muestras_llaves[i]][j]]
            muestras[muestras_llaves[i-3]][j] = [x / Rg for x in muestras[muestras_llaves[i-3]][j]]
    #print('Serie '+str(muestras_llaves[i])+' ajustada')

# -----
# Calcula gm a partir de Kn para cada curva ids-vds
# Ademas, regresa los valores de corrientes utilizados para cada calculo de gm
#
# -----
def calcular_gm(self, Kn):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    num_curvas_id_vds = self.num_curvas_id_vds

    gm = [0] * len(num_curvas_id_vds)
    corrientes_gm = [0] * len(num_curvas_id_vds)

```

```

# --- Se calcula gm para cada corriente elegida del conjunto de curvas ids-vds ---
for i in num_curvas_id_vds:
    corrientes_gm[i] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][25]
    gm[i] = 2*math.sqrt(Kn*corrientes_gm[i])

return (gm,corrientes_gm)

# -----
# Calcula las resistencias rg y rd del modelo hibrido pi del transistor MOSFET
#
# -----
def calcular_resistencias_hibridas(self,MOS_avg_lambda,corrientes_gm):
    num_curvas_id_vds = self.num_curvas_id_vds

    rg = float('inf')
    print('rg: '+str(rg))

    rd = [0] * len(num_curvas_id_vds)

    # --- Se calcula rd para cada conjunto de curvas ids-vds ---
    for i in num_curvas_id_vds:
        rd[i] = (1/(MOS_avg_lambda*corrientes_gm[i]))
        #print(rd[i])
    print('rd: '+str(rd))
    return (rd,rg)

# -----
# Calcula los puntos a graficar sobre las curvas vgs-ids
# Calcula la posicion de tres puntos: vth y dos puntos utilizados para calcular Kn,Vth y
Idss
# -----
def puntos_sobre_grafica_ids_vgs(self, Vth, ind_ids):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_parametros = self.num_curvas_parametros

    markers_on = []

    # --- Se obtiene el indice mas cercano al valor Vth previamente calculado ---
    for i in range(0,50):

```

```

        if                                     Vth
muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][num_curvas_parametros[0]][i] <= 0:
    ind_vth = i
    break
# --- Se arma el arreglo con los tres puntos a graficar sobre la curva ids-vgs ---
markers_on.append(ind_vth)
markers_on.append(ind_ids[0])
markers_on.append(ind_ids[len(ind_ids)-1])

vgs = []*len(markers_on)
ids = []*len(markers_on)

print('markers_on: '+str(markers_on))#, ' 1: '+str(markers_on[0]))
#print('len(markers_on): '+str(len(markers_on)))

# --- Se obtienen las coordenadas de los puntos anteriores (vgs,ids)---
for (i,j) in zip(markers_on,range(0,len(markers_on))):

vgs.append(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][num_curvas_parametros[0]][i])

ids.append(muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_parametros[0]][i])
#print('vgs: '+str(vgs[int(j)]))
#print('ids: '+str(ids[int(j)]))

return (vgs,ids,markers_on)

# -----
# Calcula los puntos a graficar sobre las curvas vds-ids
# Se calcula la posicion de la curva que delimita las regiones ohmica y de saturacion de
un transistor MOSFET
# Regresa tambien el valor promedio vgs al cual se grafico esa curva
# -----
def calcular_puntos_limite_grafica_ids_vds(self,Vth):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_id_vds = self.num_curvas_id_vds

    linea_lim_x = [0] * (len(num_curvas_id_vds)+1)
    linea_lim_y = [0] * (len(num_curvas_id_vds)+1)

```

```

Vgs_prom = [0] * (len(num_curvas_id_vds))

# --- Se calcula la posicion de los puntos que delimitan las regiones de operacion sobre
el conjunto de curvas ids-vds ---
for i in num_curvas_id_vds:
    Vgs_prom[i]
    round(np.average(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][i]),3)
        # --- Se obtiene el voltaje de saturacion vds_sat ---
        linea_lim_x[i+1] = Vgs_prom[i]-Vth
        for j in range(0,len(muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i])):
            if abs((Vgs_prom[i]-Vth) - muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i][j]) < 0.05:
                # --- Se obtiene la muestra mas cercana de ids correspondiente al voltaje
vds_sat ---
                linea_lim_y[i+1] = muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][i][j]
                #print('lim_curva_'+str(i)+'' (temp):
'+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+''A, dif='+str(abs((Vgs_prom-Vth)
muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][i][j])/220))
                print('lim_curva_'+str(i)+': '+str(linea_lim_x[i+1])+'V,'+str(linea_lim_y[i+1])+''A')

return (linea_lim_x,linea_lim_y,Vgs_prom)

# -----
# Calcula todos los parametros del punto de operacion del transistor MOSFET cuando
funciona como amplificador de pequena senal
# Calcula Vrd Id gm rd rlac=(Rd||rd) en el punto de operacion (v_op,i_op)
# Calcula tambien la ganancia teorica esperada
# -----
def calcular_punto_operacion_y_parametros_hibridos(self,Kn,MOS_avg_lambda):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_corrientes = self.ind_corrientes
    ind_voltajes = self.ind_voltajes
    num_curvas_pequena_senal = self.num_curvas_pequena_senal
    Rd = self.Rd

    # --- Se obtiene el punto de operacion del amplificador. Es la primera muestra del
conjunto de curvas para pequena senal ---
    v_op =
    muestras[muestras_llaves[ind_voltajes[3]+6*sel_filtro]][num_curvas_pequena_senal[0]][0]
    i_op =
    muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_pequena_senal[0]][
0]

```

```

print('v_op: '+str(v_op))
print('i_op: '+str(i_op))

# --- Se calculan los parametros hibridos en el punto de operacion ---
Vrdq
muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_señal[0]][0]
- v_op
    Idq = Vrdq/Rd
    gmq = 2*math.sqrt(Kn*Idq)
    rdq = 1/(MOS_avg_lambda*Idq)
    Rlac = 1/((1/Rd)+(1/rdq))
    ganancia_teorica = -gmq*Rlac
    return (v_op,i_op,Vrdq,Idq,gmq,rdq,Rlac,ganancia_teorica)

# -----
# Calcula la ganancia experimental (vout/vin) de acuerdo a la curva de voltajes en el
tiempo cuando el transistor MOSFET funciona como amplificador en pequena señal
# Regresa tambien los indices correspondientes a los minimos y maximos de las senales
de entrada y de salida
# -----
def calcular_punto_operacion_puntos_min_max_ganancia_experimental(self):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes
    num_curvas_pequena_señal = self.num_curvas_pequena_señal

    v_max_ind = [0]*2
    v_min_ind = [0]*2
    v_max = [0]*2
    v_min = [0]*2
    v_pp = [0]*2
    marker_v = []

    # --- Se calculan los voltajes maximos y minimos de las senales de entrada (vgs) y
salida (vds) ---
    for i in [0,1]:
        v_max[i] = np.max(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_señal[0]])
        v_min[i] = np.min(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_señal[0]])

```

```

        v_max_ind[i] = muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_señal[0]].index(v_max[i])
        v_min_ind[i] = muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][num_curvas_pequena_señal[0]].index(v_min[i])

    v_pp[i] = v_max[i] - v_min[i]

    marker_v.append(v_min_ind[i])
    marker_v.append(v_max_ind[i])

# --- Se calcula la ganancia experimental como vout/vin ---
ganancia_experimental = -v_pp[1]/v_pp[0]
print('ganancia_experimental: '+str(ganancia_experimental))

return (ganancia_experimental,marker_v)

# -----
# Calcula la posicion de los puntos que delimitan las regiones de operacion sobre las curvas ids-vds
# Calcula las ecuaciones de la recta de carga estatica y de la parabola que delimita las regiones de operacion del transistor MOSFET
# Calcula tambien la posicion del punto de operacion sobre la recta de carga estatica
# Calcula tambien la posicion del punto de maxima variacion simetrica sobre la recta de carga y otras intersecciones
# -----
def calcular_puntos_límite_y_operacion_sobre_grafica_ids_vds(self,x,y,v_op,i_op):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    sel_filtro = self.sel_filtro
    ind_voltajes = self.ind_voltajes
    num_curvas_pequena_señal = self.num_curvas_pequena_señal
    Rd = self.Rd

    # --- Se hace regresion polinomial al conjunto de puntos que indican el limite entre regiones de operacion ---
    # --- Se observo que una curva de 3er grado predice acertadamente el comportamiento de este conjunto de puntos ---
    parabola_vsat = np.poly1d(np.polyfit(x, y, 3))
    print('parabola inicial: \n'+str(np.poly1d(parabola_vsat)))

    # --- Se resta la ecuacion de la recta de carga estatica a la parabola, esto es restar los coeficientes de los terminos independiente y x (x^2 y x^3 permanecen intactos) ---

```

```

    vdd_prom
    = sum(muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_senal[0]])
    / len(muestras[muestras_llaves[ind_voltajes[2]+6*sel_filtro]][num_curvas_pequena_senal[0]])
)
print('vdd_prom: '+str(vdd_prom))
parabola_vsat[0] = parabola_vsat[0]-(vdd_prom/Rd)
parabola_vsat[1] = parabola_vsat[1]-(-1/Rd)
# --- Se obtienen las raices o soluciones de esta ecuacion de 3er grado. Estas
soluciones son las intersecciones entre la curva que delimita regiones y la recta de carga
estatica ---
print('intersecciones: \n'+str(parabola_vsat.r))

interseccion = []

# --- Las raices validas para este problema (calcular el punto de la maxima variacion
simetrica) son las no complejas y positivas ---
for i in parabola_vsat.r:
    if not np.iscomplex(i) and i>=0:
        interseccion.append(i)
print('intersecciones reales: '+str(interseccion))

# --- Se restaura la ecuacion de la curva, puesto que fue modificada anteriormente ---
parabola_vsat = np.poly1d(np.polyfit(x, y, 3))
#print('parabola sin modificar: '+str(np.poly1d(parabola_vsat)))

# --- Se calcula la ecuacion de la recta estatica ---
puntos_grafica = np.linspace(0,vdd_prom, 50)
recta_carga_estatica = np.polynomial.polynomial.Polynomial([vdd_prom/Rd,-1/Rd])

# --- Se convierten las intersecciones de numeros numpy complejos a numeros python
flotantes ---
inters = np.real(interseccion[0].item())

# --- Se calcula el punto de la maxima variacion simetrica ---
v_sim_max = ((vdd_prom - inters) / 2) + inters

print('Punto de maxima variacion simetrica: '+str(EngNumber(v_sim_max))+'V,
'+str(EngNumber(recta_carga_estatica(v_sim_max)))))

# --- Se arma el arreglo de estos tres puntos: punto de operacion, interseccion entre la
curva que delimita regiones y la recta estatica y el punto de la maxima variacion simetrica -
--
puntos_x = [v_op,inters,v_sim_max]

```

```

puntos_y = [i_op,recta_carga_estatica(inters),recta_carga_estatica(v_sim_max)]

return (recta_carga_estatica,parabola_vsat,puntos_x,puntos_y,puntos_grafica)

# -----
# Calcula la posicion de las etiquetas cuando la forma de onda es una senoidal
#
# -----
def calcular_posiciones_label(self,ind_serie,ind_muestra):
    muestras = self.muestras
    muestras_llaves = self.muestras_llaves
    num_curvas_pequena_señal = self.num_curvas_pequena_señal

    position = ""
    xoffset = 0
    yoffset = 0

    # --- Determina la mejor posicion de la etiqueta de datos con respecto al cuadrante de
    la señal senoidal ---
    if muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][ind_muestra] > muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][0]:
        if (muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][ind_muestra] - muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][ind_muestra-1]) < 0:
            #print('mod1')
            position = 'left'
            xoffset = 0
            yoffset = 0
        else:
            #print('mod2')
            position = 'right'
            xoffset = -5
            yoffset = 3
    else:
        if (muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][ind_muestra-1] - muestras[muestras_llaves[ind_serie]][num_curvas_pequena_señal[0]][ind_muestra]) < 0:
            #print('mod3')
            position = 'left'
            xoffset = 0
            yoffset = 0
        else:
            #print('mod4')
            position = 'right'

```

```
xoffset = -5  
yoffset = -10  
  
return (xoffset,yoffset)
```

Figura 371. Código completo de la clase calculadora_datos_graficas.py

11.2.3.3.2 Clase graficador.py (3ii)

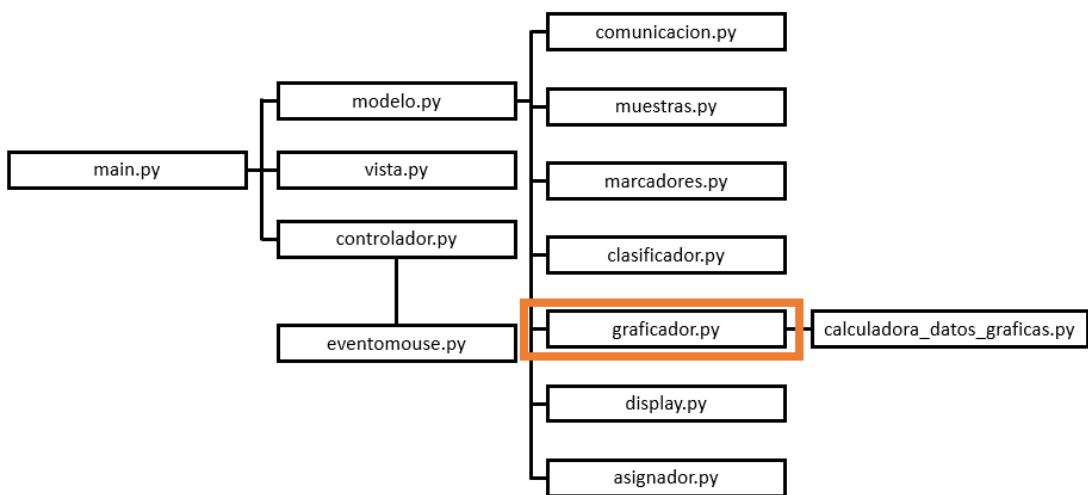


Figura 372. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

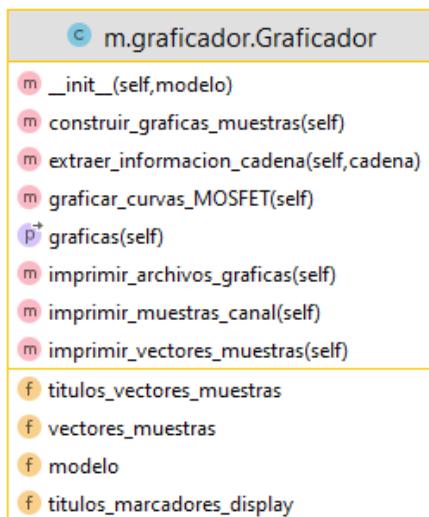


Figura 373. Métodos y atributos de la clase graficador.py.

Donde:

- (c) indica que es una **clase**.
- (m) indica que es un **método**.
- (p+) indica que es una **propiedad**.
- (f) indica que es un **atributo**.

```

# ****
#
# Clase: --> Graficador
# Modulo: -> graficador.py
#
# Descripción:
# - Graficar los datos obtenidos por arduino
# - Dibuar, mostrar informacion
#
# Fecha: julio 28/2021
#
# ****

from m.calculadora_datos_graficas import Calculadora_datos_graficas

import numpy as np
import cv2

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.ticker import EngFormatter

from collections import deque

from engineering_notation import EngNumber
import math
import statistics

graficas = []

class Graficador():

    # -----
    #      CONSTRUCTOR
    # -----
    def __init__(self,modelo):

        print("")
        print(" CONSTRUCTOR: Clase: Graficador")
        self.modelo = modelo
        self.vectores_muestras = modelo.vectores_muestras
        self.titulos_vectores_muestras = modelo.titulos_vectores_muestras
        self.titulos_marcadores_display = modelo.titulos_marcadores_display

```

```

def construir_graficas_muestras(self):

    # --- Construccion AUTOMATICA de la graficas ---
    titulos_marcadores = list(self.titulos_marcadores_display)

    global graficas
    graficas = []

    muestras = self.vectores_muestras

    for i in range(len(titulos_marcadores)):
        titulo_grafica = self.titulos_marcadores_display[titulos_marcadores[i]]
        nombre_vectores_grafica = self.extraer_informacion_cadena(titulo_grafica)

        # --- Construccion AUTOMATICA de cada GRAFICA, a partir de la
        #   informacion proporcionada en la clase Modelo en las estructuras:
        #   -> titulos_vectores_muestras <-> Vector
        #   -> titulos_marcadores_display <-> Diccionario
        #   Estas estructuras de datos se conectan con los nombres de los
        #   vectores que se encuentran en el Diccionario muestras
        #   -> vectores_muestras <-> Diccionario
        #
        # -----
        plt.figure(i)
        # --- Graficas VECTOR A vs. muestras ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 1:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      label = nombre_vectores_grafica[0][0] + ' vs. muestras')
        # --- Graficas VECTOR A vs VECTOR B ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 2:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      muestras[nombre_vectores_grafica[0][1]],
                      label = nombre_vectores_grafica[0][0] + ' vs. ' +
                      nombre_vectores_grafica[0][1])
        # --- Graficas MULTIPLES tipo: VECTOR A vs VECTOR B ---
        if len(nombre_vectores_grafica) > 2:
            for g in range(len(nombre_vectores_grafica)):
                plt.plot(muestras[nombre_vectores_grafica[g][0]],
                          muestras[nombre_vectores_grafica[g][1]],
                          label = nombre_vectores_grafica[g][0] + ' vs. ' +
                          nombre_vectores_grafica[g][1])
        plt.title = 'Grafica ' + str(i)
        plt.ylabel = 'y label'
        plt.xlabel = 'x label'
        plt.legend()

```

```

# --- Guardado de las gráficas en:
#   1. "Archivo en disco [*.jpg] para su uso posterior
#   por parte de la Clase Display
#   2. El arreglo GLOBAL: 'graficas', el cual contiene
#       la ruta en donde se guardó la gráfica correspondiente
# -----
grafica = 'g/grafica_' + str(i) + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

def extraer_informacion_cadena(self,cadena):

    #print("")
    #print(" cadena original = ",cadena)

    # --- Separa grupos de letras ---
    subcadena = ""
    subtulos = []
    ignorar_caracter = False
    num_char = 0
    for ch in cadena:
        num_char += 1
        # --- Ignorar los caracteres "espacio" [' '] y "coma" [','] ---
        if ch == ' ' or ch == ',':
            ignorar_caracter = True
            if num_char != 1:
                subtulos.append(subcadena)
            subcadena = ""
        else:
            subcadena += ch
        if num_char == len(cadena): subtulos.append(subcadena)

    # --- Genera tuplas de dos elementos ---
    tupla = []
    arreglo_tuplas = []
    for i in range(len(subtulos)):
        # --- Detecta conector [vs] ó espacio [' '] --
        if subtulos[i] == 'vs' or subtulos[i] == " ":
            dummy = 0 # <-- No hacer nada ---
        else:
            tupla.append(subtulos[i])
        if subtulos[i] == " " or i == (len(subtulos)-1):
            arreglo_tuplas.append(tupla)

```

```

        tupla = []

    return arreglo_tuplas

def graficar_curvas_MOSFET(self):

    global graficas
    graficas = []

    muestras = self.vectores_muestras

    muestras_llaves = list(muestras)

    # Son los indices de los canales de muestras recibidos como se difnio en la clase
    modelo
    ind_voltajes = [0,1,3,4] #Vgg, Vgs, Vdd, Vds
    ind_corrientes = [2,5] # Ig, Id

    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras
    hay 6 curvas)
    lim_max = len(muestras['Id'])
    #print('lim_max: '+str(lim_max))
    #print('muestras_llaves: '+str(len(muestras_llaves)))

    # Establecer el numero de curvas para cada conjunto de parametros
    num_curvas_id_vds = [0,1,2,3,4]
    num_curvas_parametros = [5]
    num_curvas_pequena_senal = [6]

    Rd = 2200
    Rg = 100000

    # Seleccion entre las series de muestras sin filtro, con filtro circular o con filtro pasa-
    bajas
    # 0 = sin filtro
    # 1 = con filtro circular cp
    # 2 = con filtro pasa bajas lp

    sel_filtro = 1

    calc
    Calculadora_datos_graficas(self.modelo,muestras_llaves,lim_max,sel_filtro,ind_corrientes
    ,ind_voltajes,num_curvas_id_vds,num_curvas_parametros,num_curvas_pequena_senal,
    Rd,Rg)

```

```

# Ajuste de Id con Rd = 220 ohm y de Ig con Rg = 100 Kohm
calc.ajustar_id_con_resistencias()

# Se obtienen los parametros Vth, Kn y gm.
Vth, Kn, Idss, ind_ids = calc.calcular_Vth_Kn_Idss()

gm, corrientes_gm = calc.calcular_gm(Kn)
#print('corriente: '+str(corrientes_gm)+', gm: '+str(gm))

#print('Parametros Híbridos')

# --- Calculos para la grafica 2 -----
vgs, ids, markers_on = calc.puntos_sobre_grafica_ids_vgs(Vth,ind_ids)

# --- Calculos para la grafica 3 -----
# --- Limites de las regiones de operacion -----
linea_lim_x,linea_lim_y,Vgs_prom = calc.calcular_puntos_lmite_grafica_ids_vds(Vth)

ids_min,ids_max,MOS_lambda = calc.calcular_lambda(linea_lim_y)
MOS_avg_lambda = -(1/statistics.mean(MOS_lambda))
#print('MOS_avg_lambda: '+str(MOS_avg_lambda))
# --- Parametros adicionales: rg, rd -----
rd, rg = calc.calcular_resistencias_hibridas(MOS_avg_lambda,corrientes_gm)

# --- Punto de operacion -----
v_op,i_op,Vrdq,Idq,gmq,rdq,Rlac,ganancia_teorica =
calc.calcular_punto_operacion_y_parametros_hibridos(Kn,MOS_avg_lambda)

# --- Ganancia experimental -----
ganancia_experimental,marker_v =
calc.calcular_punto_operacion_puntos_min_max_ganancia_experimental()

# --- Coordenadas de los puntos de los limites de las regiones de operacion, recta de
carga estatica, punto de operacion -----

```

```

    recta_carga_estatica,parabola_vsat,puntos_x,puntos_y,puntos_grafica      =
calc.calcular_puntos_limite_y_operacion_sobre_grafica_ids_vds(linea_lim_x,linea_lim_y,v
_op,i_op)

# -----
-----
```

Algunas referencias que consulte para realizar el código, son las siguientes:

```

# Python matplotlib data labels (plt.annotate())
#     https://queirozf.com/entries/add-labels-and-text-to-matplotlib-plots-annotation-examples
#
# Python matplotlib change axis limits (see first two answers)
# https://stackoverflow.com/questions/3777861/setting-y-axis-limit-in-matplotlib
#
# Python engineering format (EngNumber())
# https://pypi.org/project/engineering-notation/
#
# Python insert greek symbols
# https://pythonforundergradengineers.com/unicode-characters-in-python.html
#
# Utilizar parentesis en vez de iguales
# ej. plt.xlabel('Vds (V)') en vez de plt.xlabel = 'Vds (V)'
#
# Engineering format en axis labels (plt.gca(), EngFormatter(),
ax.xaxis.set_major_formatter() y ax.yaxis.set_major_formatter())
#
https://matplotlib.org/stable/gallery/text_labels_and_annotations/engineering_formatter.html

# https://www.geeksforgeeks.org/formatting-axes-in-python-matplotlib/
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.gca.html
#
# Converting from numpy float complex number to python float number
# https://www.geeksforgeeks.org/numpy-iscomplex-python/
# https://numpy.org/doc/stable/reference/generated/numpy.ndarray.item.html
#
# -----
-----
```

```

print('Grafica 0')
# --- Grafica 0 : Voltajes en el tiempo
#
# --- Esta grafica es para observar el comportamiento de los voltajes en el tiempo
# --- Se observan los voltajes Vgg, Vgs, Vdd, Vds
```

```

plt.figure(0)

ax = plt.gca()
formatter0 = EngFormatter(unit='V')
ax.yaxis.set_major_formatter(formatter0)

for i in ind_voltajes:
    arr = []
    for j in range(0,lim_max):
        arr = arr+muestras[muestras_llaves[i+6*sel_filtro]][j]
    plt.plot(arr, label=muestras_llaves[i+6*sel_filtro])

plt.xlabel('Muestras')
plt.ylabel('Voltaje')
plt.title("Voltajes en el tiempo")
plt.legend()
grafica = 'g/00_Voltajes' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 1')
# --- Grafica 1: Ig vs Vgs -----
-----
plt.figure(1)

plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][num_curvas_parametros[0]],
,muestras[muestras_llaves[ind_corrientes[0]+6*sel_filtro]][num_curvas_parametros[0]],
label='Ig vs Vgs')

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

x1,x2,y1,y2=plt.axis()

plt.xlim(0,4)
plt.ylim(y1-y2*10,y2*1000)
plt.xlabel('Vgs')
plt.ylabel('Ig')
plt.subplots_adjust(left=0.15)
plt.title("Ig vs Vgs")
plt.legend()

```

```

grafica = 'g/01_lg_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 2')
# --- Grafica 2: Id vs Vgs -----
-----
plt.figure(2)

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(muestras[muestras_llaves[ind_voltajes[1]+6*sel_filtro]][num_curvas_parametros[0]]
[0:(markers_on[len(markers_on)-
1]+2)],muestras[muestras_llaves[ind_corrientes[1]+6*sel_filtro]][num_curvas_parametros[0]]
][0:(markers_on[len(markers_on)-1]+2)],'o-', markevery=markers_on, label='Id vs Vgs')

for i in range(0,len(markers_on)):
    vth_str = ""
    if i == 0:
        vth_str = 'Vth: '
    plt.annotate(vth_str+str(EngNumber(vgs[i]))+'V,
    '+str(EngNumber(ids[i]))+'A',(vgs[i],ids[i]),textcoords="offset points",xytext=(-5,3),ha='right')

x1,x2,y1,y2=plt.axis()
plt.axis([0,x2,y1,y2])

plt.xlabel('Vgs')
plt.ylabel('Id')
plt.title("Id vs Vgs")
plt.legend()
grafica = 'g/02_Id_vs_Vgs' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 3')
# --- Grafica 3: Id vs Vds (Cinco curvas) -----
-----
plt.figure(3)

```

```

ax = plt.gca()
formatter0 = EngFormatter(unit='A')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='V')
ax.xaxis.set_major_formatter(formatter0)

# --- Cuevas caracteristicas ids-vds ---
for [i,j] in zip(num_curvas_id_vds,range(0,len(num_curvas_id_vds))):


plt.plot(muestras[muestras_llaves[ind_VOLTAJES[3]+6*sel_filtro]][i],muestras[muestras_llaves
[ind_CORRIENTES[1]+6*sel_filtro]][i], label='Vgs = '+str(Vgs_prom[i])#, 'o-',
markevery=[ids_min[j],ids_max[j]], label='Vgs = '+str(Vgs_prom[i]))


# --- Puntos que delimitan las regiones de operacion ---
plt.plot(linea_lim_x,linea_lim_y,linestyle = 'none',marker='o')
for i in range(1,len(linea_lim_x)):
    plt.annotate(str(EngNumber(linea_lim_x[i]))+'V',
    '+str(EngNumber(linea_lim_y[i]))+'A',(linea_lim_x[i],linea_lim_y[i]),textcoords="offset
    points",xytext=(5,-7),ha='left')

# --- Recta de carga estatica ---
plt.plot(puntos_grafica,recta_carga_estatica(puntos_grafica), label='Recta de carga
estatica')


# --- Punto de operacion,interseccion entre curva que delimita regiones y recta de
carga estatica, punto de maxima variacion simetrica ---
plt.plot(puntos_x[0],puntos_y[0],'o',label='Punto de operacion')
#plt.plot([puntos_x[1],puntos_x[2]],[puntos_y[1],puntos_y[2]],'o')
plt.plot(puntos_x[1:(len(puntos_x))],puntos_y[1:(len(puntos_x))],'o')

for i in range(0,len(puntos_x)):
    v_sim_str =
    alineacion = 'left'
    desplazamiento = (5,3)
    if i == 2:
        v_sim_str = 'Maxima variacion simetrica: \n'
        desplazamiento = (5,0)
    elif i == 0 and puntos_x[0] < puntos_x[2]:
        alineacion = 'right'
        desplazamiento = (-5,-3)
    elif i == 0 and puntos_x[0] > puntos_x[2]:
        alineacion = 'left'
        desplazamiento = (5,-10)

```

```

    plt.annotate(v_sim_str+str(EngNumber(puntos_x[i]))+'V',
'+str(EngNumber(puntos_y[i]))+'A',(puntos_x[i],puntos_y[i]),textcoords="offset
points",xytext=desplazamiento,ha=alineacion)

# --- Curva que delimita regiones de operacion ---
puntos_grafica = np.linspace(0, puntos_x[1]*9/8, 100)
plt.plot(puntos_grafica,parabola_vsat(puntos_grafica),'--',label='Limite entre regiones')

x1,x2,y1,y2=plt.axis()
plt.axis([x1,5,y1,y2])

plt.xlabel('Vds')
plt.ylabel('Id')
plt.title("Id vs Vds")
plt.legend()
grafica = 'g/03_Id_vs_Vds' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 4')
# --- Grafica 4: gm vs Id -----
-----
```

```

plt.figure(4)

ax = plt.gca()
formatter0 = EngFormatter(unit='\u03c3')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm,gm,'o-', label='gm vs Id')

for i in range(0,len(gm)):

plt.annotate(str(EngNumber(gm[i]))+'\u03c3',(corrientes_gm[i],gm[i]),textcoords="offset
points",xytext=(0,10),ha='center')

x1,x2,y1,y2=plt.axis()
plt.axis([x1-x1/8,x2+x2/8,y1,y2+y2/16])

plt.xlabel('Id')
plt.ylabel('gm')
plt.title("gm vs Id")
```

```

plt.legend()
grafica = 'g/04_gm_vs_Id' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

print('Grafica 5')
# --- Grafica 5: rd vs Id -----
-----
```

```

plt.figure(5)

ax = plt.gca()
formatter0 = EngFormatter(unit='\u03a9')
ax.yaxis.set_major_formatter(formatter0)
formatter0 = EngFormatter(unit='A')
ax.xaxis.set_major_formatter(formatter0)

plt.plot(corrientes_gm,rd,'o-', label='rd vs Id')
for i in range(0,len(rd)):

    plt.annotate(str(EngNumber(rd[i]))+'\u03a9',(corrientes_gm[i],rd[i]),textcoords="offset
points",xytext=(0,10),ha='left')

    x1,x2,y1,y2 = plt.axis()
    plt.axis([x1,x2+x2/4,y1,y2+y2/4])

    plt.xlabel('Id')
    plt.ylabel('rd')
    plt.title("rd vs Id")
    plt.legend()
    grafica = 'g/05_rd_vs_Id' + ".jpg"
    plt.savefig(grafica)
    graficas.append(grafica)

print('Grafica 6')
# --- Grafica 6 Voltajes en el tiempo. Amplificador en fuente comun -----
-----
```

```

plt.figure(6)

for i in [0,1]:
    mark_v = [marker_v[2*i],marker_v[2*i+1]]
```

```

plt.plot(muestras[muestras_llaves[ind_voltajes[i*2]+6*sel_filtro]][[num_curvas_pequena_señal[0]], label=muestras_llaves[ind_voltajes[2*i]+6*sel_filtro])

plt.plot(muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][[num_curvas_pequena_señal[0]],'o-', markevery=mark_v, label=muestras_llaves[ind_voltajes[2*i+1]+6*sel_filtro])
    for j in [0,1]:
        voltaje =
        muestras[muestras_llaves[ind_voltajes[1+i*2]+6*sel_filtro]][[num_curvas_pequena_señal[0]
        ][marker_v[2*i+j]]]
        xoffset,yoffset =
        calc.calcular_posiciones_label(ind_voltajes[1+i*2]+6*sel_filtro,marker_v[2*i+j])

        plt.annotate(str(EngNumber(voltaje))+'V',(marker_v[2*i+j],voltaje),textcoords="offset
        points",xytext=(xoffset,yoffset),ha='right')

        x1,x2,y1,y2=plt.axis()
        plt.axis([x1,x2,0,y2])

        plt.xlabel('Muestras')
        plt.ylabel('Voltaje')
        plt.title("Voltajes en el tiempo. Amplificador en pequena señal")
        plt.legend()
        grafica = 'g/06_Voltajes_Amplificador_Fuente_Comun' + ".jpg"
        plt.savefig(grafica)
        graficas.append(grafica)

'''print('Punto de operacion')
print('Idq: '+str(Idq))
print('gm: '+str(gm))
print('rd: '+str(rd))
print('Rlac: '+str(Rlac))
print('Ganancia experimental: '+str(ganancia_experimental))
print('Ganancia teorica: '+str(ganancia_teorica))'''

# --- Grafica 7 Resumen parametros hibridos-----
-----


# --- Generacion de un canvas de w = 240 x h = 180 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

```

```
# --- RESUMEN PARAMETROS HIBRIDOS ---
```

```
    cv2.putText(canvas2,"Transistor      MOSFET      2N7000", (30,
70),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"PARAMETROS      HIBRIDOS", (30,
100),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Lambda : " + str(EngNumber(MOS_avg_lambda)) + 'V^-1', (30,
130),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Voltaje de umbral de encendido : " + str(EngNumber(Vth)) + 'V', (30,
150),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Idss : " + str(EngNumber(Idss)) + 'A', (30,
170),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Kn : " + str(EngNumber(Kn)) + 'A/V^2', (30,
190),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
```

```
# --- PUNTO DE OPERACION (AMPLIFICADOR EN PEQUENA SENAL) ---
```

```
    cv2.putText(canvas2,"PUNTO      DE      OPERACION", (30,
240),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Vdsq : " + str(EngNumber(v_op))+ 'V', (30,
270),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Idq : " + str(EngNumber(Idq))+ 'A', (30,
290),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"gm : " + str(EngNumber(gmq))+ 'S', (30,
310),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"rd : " + str(EngNumber(rdq))+ 'ohm', (30,
330),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"rg : " + str(rg), (30, 350),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)

    cv2.putText(canvas2,"Rlac (Rd||rd) : " + str(EngNumber(Rlac))+ 'ohm', (30,
375),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
```

```
    cv2.putText(canvas2,"Ganancia experimental (Vout/Vin): " +
str(EngNumber(ganancia_experimental)), (30, 395),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)
```

```
    cv2.putText(canvas2,"Ganancia teorica (-gm*Rlac): " +
str(EngNumber(ganancia_teorica)), (30, 425),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0,
0), 1, cv2.LINE_AA)
```

```
filename = "g/07_parametros_hibridos" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)
```

```

# --- Auxiliar para depuracion ---
#cv2.imshow("Canvas",canvas2)
#cv2.waitKey(0)
plt.show()

@property
def graficas(self):
    return graficas # --- Nombre de la grafica y ruta donde se guarda ---

def imprimir_archivos_graficas(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" --- Archivos de imagenes de graficas *.jpg ---")
    print("*****")
    for g in graficas:
        print("")
        print(g)

def imprimir_muestras_canal(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" --- Vectores de muestras por canal ---")
    print("*****")
    # --- ESTRUCTURA del diccionario:
    #   datos = {canal: i, llaveMuestras[i]: vector de datos}
    # -----
    muestras = self.vectores_muestras
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k],": ",muestras[llaves[k]])

def imprimir_vectores_muestras(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")

```

```
print(" -- Muestras leidas de la Clase Modelo --")
print("*****")
muestras = self.vectores_muestras
llaves = self.titulos_vectores_muestras
print(" llaves = ",llaves)
for k in range(len(llaves)):
    print("")
    print(" --- ",llaves[k],": ",muestras[llaves[k]])
```

Figura 374. Código completo de la clase graficador.py

11.2.4 Circuito secuencial (d)

Ubicación de los circuitos bajo prueba propuestos

- a. **Trazador de curvas (a)** con un transistor MOSFET de propósito general.
- b. **Circuito generador de señal senoidal digital (b)** de muy baja frecuencia (DDS - Síntesis Digital Directa) construido con convertidores digitales analógicos.
- c. **Amplificador de una etapa (c)** con un transistor MOSFET.
- d. **Circuito secuencial (d).** Se proponen dos versiones: un sistema secuencial discreto armado con compuertas y flip-flops (**implementación por hardware (i)**) y el mismo sistema secuencial programado en un microcontrolador (**implementación por software (ii)**).

Para el **circuito secuencial ()**, se presenta lo siguiente:

1. Diagrama esquemático del circuito.
2. Programas de Arduino completos:
 - i. Programa que implementa la máquina de estados.
 - ii. Programa que prueba y lee la máquina de estados.
3. De los programas en Python:
 - i. Clase fsm.py.
 - ii. Clase calculadora_datos_graficas.py.
 - iii. Clase graficador.py.

11.2.4.1 Diagrama esquemático (1)

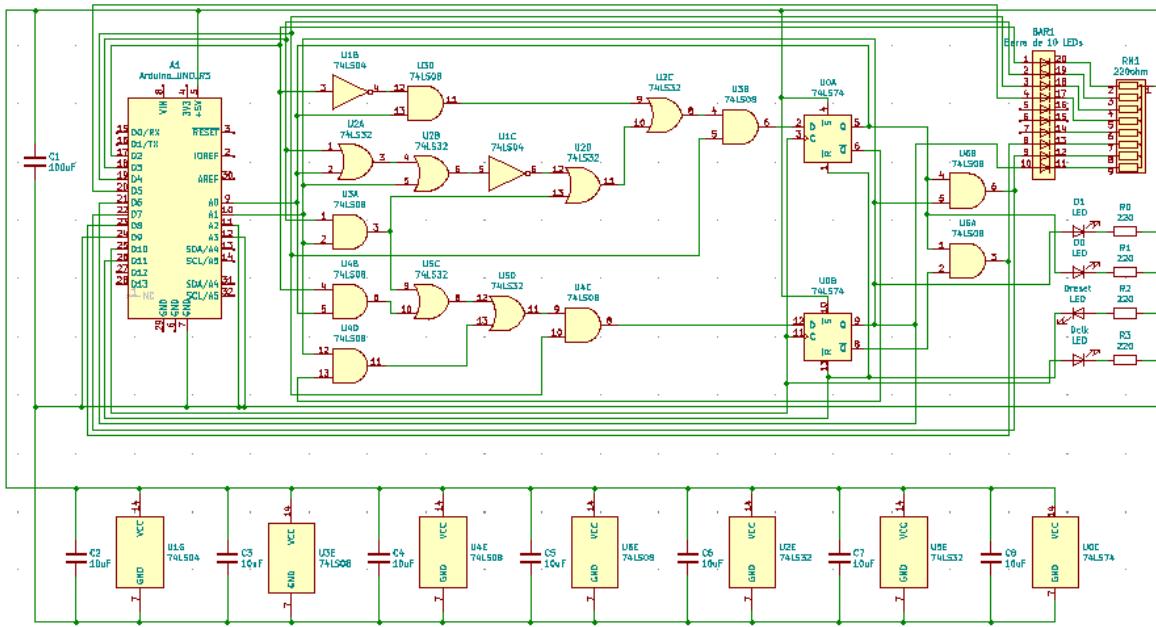


Figura 375. Diagrama esquemático completo del circuito secuencial (d) implementado por hardware (i).

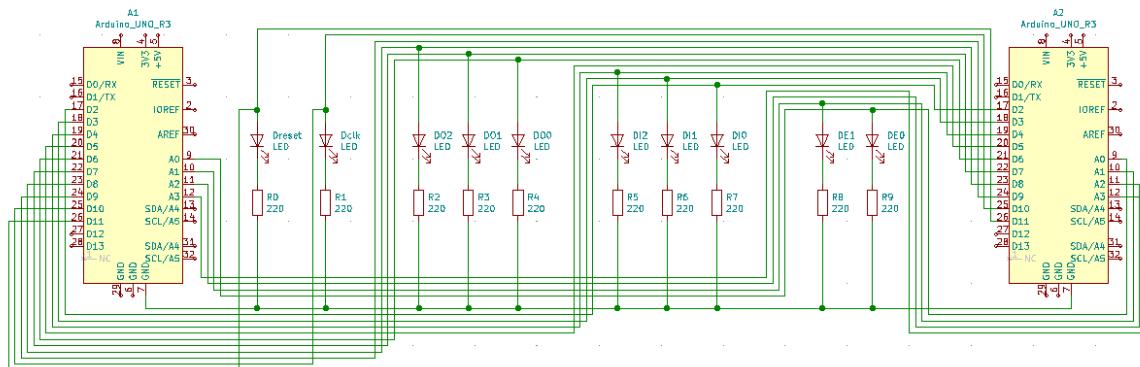


Figura 376. Diagrama esquemático completo del circuito secuencial (d) implementado por software (ii).

11.2.4.2 Programa de Arduino (2)

11.2.4.2.1 Programa de Arduino. Implementación de la máquina de estados (2i)

Programa de Arduino para la implementación del Circuito Secuencial
Nombre del programa: Arduino AR 65 Implementación FSM

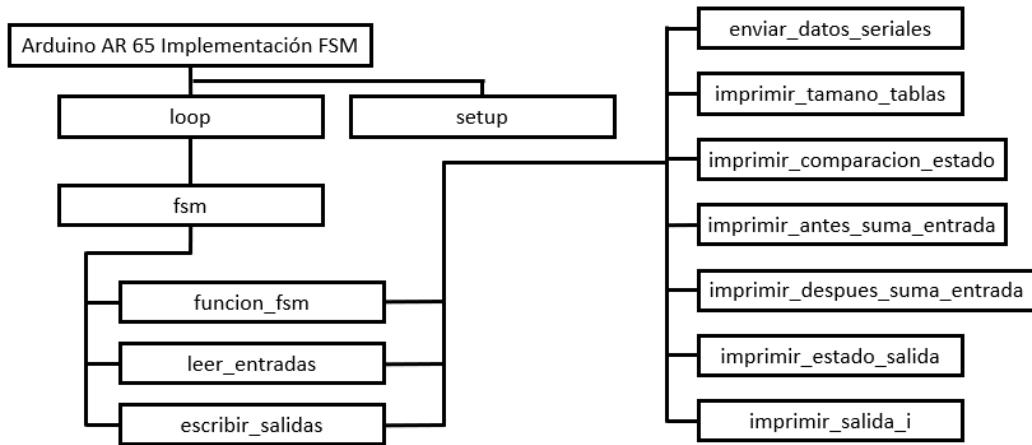


Figura 377. Diagrama a bloques del programa completo de Arduino para la implementación del **Círcuito Secuencial (4)**.

```
/*
PROGRAMA: 26072021

Implementacion de una maquina de estados finitos

Este programa se ejecuta en conjunto con:

ARDUINO (1): ARDUINO AR 65 Implemenracion FSM.ino
ARDUINO (2): ARDUINO AR 65 Prueba y Lectura FSM.ino
PYTHON: Completo AR 65 FSM.py (monitor) 25062021-P

TARJETA UTILIZADA: Arduino UNO, expandible a Arduino MEGA
Opcional: Utilizar LEDs conresistencias para observar el funcionamiento
del circuito

28 / julio / 2021

*/
/***
* -----
* Declaracion de arreglos de pines digitales
*/
```

```

 * pin_entradas[] -> son los pines en donde se leeran las entradas que
llegan la maquina de estados finitos
 * pin_lecturas[] -> son los pines de donde se escribiran las salidas de
la maquina de estados finitos
 * pin_estados[] -> son los pines que escribira el estado en el que se
encuentra la maquina de estados finitos
 *
 * Como se observa, se pueden implementar maquinas de estados con las
siguientes caracteristicas:
 * - Hasta cuatro entradas
 * - Hasta cuatro salidas
 * - Hasta 16 estados diferentes
 * Este programa se probó sobre un Arduino UNO, pero es facilmente
expandible si se implementara, por ejemplo,
 * en un Arduino MEGA
 * -----
-----
 */

const int pin_entradas[] = {2,3,4,5};
const int pin_salidas[] = {6,7,8,9};
const byte pin_estados[] = {A0,A1,A2,A3};
const int estado_inicial = 0;

/**
 * -----
-----
 * Declaracion de pines de control
 * pinCLK -> Define el pin del reloj que se utilizara para enviar señales
de reloj a la maquina de estados
 * pinReset -> Define el pin de reset que inicializara la maquina de
estados
 *
 * clk -> Es el estado del reloj
 * reset -> Indica si hay que reiniciar la maquina de estados
 * -----
-----

int pinCLK = 10;
int pinReset = 11;
int clk = 0;
int reset = 0;

/**
 * -----
-----
 * Variables para el control del tiempo de muestreo
 * lastime -> guarda el ultimo instante de tiempo en el que enviaron
datos por el puerto serie
 * sampleTime -> duración entre cada muestra tomada
 * numMuestras -> numero de muestras tomadas por cada curva
 *
 * Por lo tanto, la frecuencia de cada conjunto de muestras es de 2Hz
(numMuestras * sampleTime)/1000
 * -----
-----

unsigned long lastTime = 0;

```

```

unsigned long sampleTime = 1000;
double firstTime = 0;

/**
 * -----
-----
 * Variables de la maquina de estados
 * entrada -> Es el valor de entrada a la maquina
 * salida -> Es el valor de salida que genera la maquina
 *
 *
-----
 */
int entrada = 0x00;
int salida = 0x00;

// Ejemplo de un control para proteccion contra corto circuito para
fuente de alimentacion
/**
 * -----
-----
 * estado_inicial -> Indica el estado en el cual empieza la maquina
 * estado_actual -> Indica el estado actual de la maquina
 * estado_futuro -> Indica el estado al cual va a cambiar la maquina en
el siguiente pulso de reloj
 *
 *
-----
 */
byte estado_actual = 0;
byte estado_futuro = 0;

/**
 * -----
-----
 * num_estados -> Indica cuantos estados tiene la maquina
 * bits_transicion -> Indica cuantos bits tiene las numero de las
entradas de transiciones que permiten cambiar entre estados
 * bits_salida -> Indica la cantidad de bits de salida de la maquina de
estados
 *
 *
-----
 */
const int num_estados = 4;
const int bits_transicion = 3;
const int bits_salida = 3;

/**
 * -----
-----
 * Las siguientes tablas especifican el funcionamiento de la maquina de
estados finitos
 * El prefijo 0B indica que el numero que sigue esta en binario
 * El tamano de estas tablas esta determinador por las constantes
num_estados, bits_salida y bits_transicion
 * Cada estado de la maquina corresponde a una entrada de la tabla, como
se indica en cada una de las tablas.
 *

```

```

 * tabla_transiciones_estados -> Indica cuales son las transiciones con
las que la maquina cambia de estado
 *
Las transiciones que no aparecen, se
asume que la maquina no cambia de estado
 *
IMPORTANTE: Se completan la tabla con -1
para las transiciones en las que no hay cambio de estado
 *
 * tabla_transiciones_estado_futuro -> Indica el estado al cual va a
cambiar la maquina en el siguiente pulso
 *
de reloj. Cada estado especificado
en esta tabla corresponde a la
 *
transicion de la tabla
tabla_transiciones_estados
 *
 * tabla_output_por_estado -> Indica la salida del estado actual de la
maquina
 * -----
-----
 */
const int
tabla_transiciones_estados[num_estados][round(pow(2,bits_transicion))] =
{
    {0B100,0B101,-1,-1,-1,-1,-1,-1,-1}, //estado 0: Apagado
1}, //estado 1: Encendido funcionando
{0B000,0B001,0B010,0B011,0B101,0B11
1,-1,-1}, //estado 2: Corto circuito
{0B000,0B001,0B010,0B011,0B110,0B11
1,-1,-1}, //estado 3: Reset
{0B000,0B001,0B010,0B011,0B100,0B10
1,-1,-1}};

};

const int
tabla_transiciones_estado_futuro[num_estados][round(pow(2,bits_transicion
))] = {
Apagado {1,1}, //estado 0:
Encendido funcionando {0,0,0,0,2,2}, //estado 1:
Corto circuito {0,0,0,0,3,3}, //estado 2:
Reset {0,0,0,0,1,2} //estado 3:
};

const int tabla_output_por_estado[num_estados] = {
    (0B000), //estado 0: Apagado
    (0B100), //estado 1: Encendido
funcionando (0B001), //estado 2: Corto
circuito (0B011) //estado 3: Reset
};

/**

```

```

 * Inicializa todos los pines a utilizar, inicializa el reloj y la
maquina de estados, la comunicacion serial y la comunicacion I2C para
comunicarse con los dos PCF8591
*
*/
void setup() {
// Inicializa los pines en donde se escribira el estado de la maquina
for(int i=0;i<sizeof(pin_estados)/sizeof(pin_estados[0]);i++)
{
    pinMode(pin_estados[i],OUTPUT);
}

// Inicializa los pines en donde se leera la entrada a la maquina
for(int i=0;i<sizeof(pin_entradas)/sizeof(pin_entradas[0]);i++)
{
    pinMode(pin_entradas[i],INPUT);
}

// Inicializa los pines en donde se escribira la salida de la maquina
for(int i=0;i<sizeof(pin_salidas)/sizeof(pin_salidas[0]);i++)
{
    pinMode(pin_salidas[i],OUTPUT);
}

// Se inicializan los pines en donde se leera el reloj y el reset
pinMode(pinCLK,INPUT);
pinMode(pinReset,INPUT);

// --- Inicializa velocidad de transmision serial ---
Serial.begin(9600);

// --- Sincronizacion de comunicacion serial ---
Serial.println("inicio");
estado_actual = estado_inicial;
fsm();
firstTime = millis();
}

void loop() {
// Lee e identifica un cambio a 1 en el pin del reloj
// La maquina realiza cambios en los flancos positivos del pin del
reloj
clk = digitalRead(pinCLK);
if(clk == 1)
{
    // Lee el pin de reset. Si es 1 se inicializa la maquina de estados
finitos a su estado inicial
    // De lo contrario continua con el funcionamiento de la maquina
reset = digitalRead(pinReset);
    if(reset == 0)
    {
        estado_actual = estado_inicial;
    }
fsm();

// Espera a que el pin de reloj cambie a 0 para continuar
while(clk == 1){clk = digitalRead(pinCLK);};
}

```

```

    }

    // (Opcional) Cada sampleTime milisegundos envia informacion al puerto
    serie para observar su comportamiento
    /*if ((millis()-lastTime) >= sampleTime)
    {
        //Serial.println(millis()-firstTime);
        //Serial.println(millis()-lastTime);
        lastTime = millis(); // Actualiza el tiempo de la ultima vez que se
        tomaron muestras
        //imprimir_estado_maquina();

    }*/
}

/***
 * Implementa el funcionamiento de la maquina de estados finitos, de
 acuerdo a los parametros establecidos al principio del programa
 */
void fsm()
{
    // Lee la entrada
    leer_entradas();

    // Determina si cambia de estado
    funcion_fsm();
    //escribir_salidas(pin_salidas,salida);

    // Escribe la salida correspondiente al estado actual de la maquina de
    estados
    escribir_salidas();

    // Escribe el estado actual de la maquina de estados
    escribir_estados();

    // (Opcional) Envia datos seriales sobre la entrada, estado y salida de
    la maquina de estados
    //enviar_datos_seriales();

    // Actualiza el estado para el siguiente pulso de reloj
    estado_actual = estado_futuro;
}

/***
 * Envia al puerto serial la entrada, estado y salida de la maquina de
 estados
 */
void enviar_datos_seriales(){
    if(estado_actual != estado_futuro)
    {
        //Serial.print("Entrada: ");
        Serial.println(entrada);
        //Serial.print(", Estado actual: ");
        Serial.println(estado_actual);
    }
}

```

```

        //Serial.print(", salida: ");
        Serial.println(salida);
    }

    /**
     * Implementa el funcionamiento de la maquina de estados, de acuerdo a
     los parametros establecidos al inicio del programa
     */
    void funcion_fsm()
{
    //imprimir_tamano_tablas();
    // Actualiza la salida de la maquina
    salida = tabla_output_por_estado[estado_actual];

    // Verifica si hay que hacer algun cambio de estado, comparando con las
    posibles entradas que indican un cambio de estado
    for(int
i=0;i<(sizeof(tabla_transiciones_estados[estado_actual])/2);i++)
    {
        //imprimir_comparacion_estado(i);
        if(entrada == tabla_transiciones_estados[estado_actual][i])
        {
            estado_futuro = tabla_transiciones_estado_futuro[estado_actual][i];
            //imprimir_cambio_estado();
            return;
        }
    }
}

/**
 * Lee las entradas en binario que le llegan a la maquina de estados
 * Convierte este valor leido en un valor en decimal
 */
void leer_entradas()
{
    entrada = 0x00;
    for(int i=0;i<bits_transicion;i++){
        //imprimir_anteriores_suma_entrada();
        entrada += (byte)round(digitalRead(pin_entradas[i])*pow(2,i));
        //imprimir_despues_suma_entrada(i);
    }
    //Serial.print("entrada(final): ");
    //Serial.println(entrada);
}

/**
 * Escribe la salida de la maquina de estados
 * De un valor en decimal lo convierte en binario
 */
void escribir_salidas()
{
    int temp = salida;
    for(int i=(sizeof(pin_salidas)/2)-1;i>=0;i--)

```

```

{
    //imprimir_estado_salida(i,salida);
    if(salida>=(pow(2,i)))
    {
        salida -= round(pow(2,i));
        digitalWrite(pin_salidas[i],HIGH);
        //imprimir_salida_i(i,1);
    }
    else
    {
        digitalWrite(pin_salidas[i],LOW);
        //imprimir_salida_i(i,0);
    }
}
salida = temp;
}

/**
 * Escribe el estado actual de la maquina de estados
 * De un valor en decimal lo convierte en binario
 */
void escribir_estados()
{
    int temp = estado_actual;
    //Serial.print("estado actual: ");
    //Serial.println(estado_actual);
    for(int i=(sizeof(pin_estados)/2)-1;i>=0;i--)
    {
        //imprimir_estado_salida(i,estado_actual);
        if(estado_actual>=(pow(2,i)))
        {
            estado_actual -= round(pow(2,i));
            digitalWrite(pin_estados[i],HIGH);
            //imprimir_salida_i(i,1);
        }
        else
        {
            digitalWrite(pin_estados[i],LOW);
            //imprimir_salida_i(i,0);
        }
    }
    //delay(500);
    estado_actual = temp;
}

// Las siguientes subrutinas despliegan informacion extra sobre lo que
// ocurre en el programa

/**
 * Imprime en el monitor serial el tamano de las tablas de la maquina de
estados
*/
void imprimir_tamano_tablas(){
    Serial.print("sizeof(tabla_transiciones_estados): ");
    Serial.println(sizeof(tabla_transiciones_estados[estado_actual])/2);
}

```

```

    Serial.print("sizeof(tabla_transiciones_estado_futuro): ");
    Serial.println(sizeof(tabla_transiciones_estado_futuro[0])/2);
    Serial.print("sizeof(tabla_output_por_estado): ");
    Serial.println(sizeof(tabla_output_por_estado[0])/2);
}

/**
 * Imprime en el monitor serial si con la entrada leida es necesario
realizar un cambio de estado
*/
void imprimir_comparacion_estado(int i){
    Serial.print("iteracion ");
    Serial.print(i);
    Serial.print(": ");
    Serial.print(entrada);
    Serial.print(" == ");
    Serial.print(tabla_transiciones_estados[estado_actual][i]);
    Serial.print(" : ");
    Serial.println(entrada ==
tabla_transiciones_estados[estado_actual][i]);
}

/**
 * Imprime en el monitor serial el cambio de estado
*/
void imprimir_cambio_estado(){
    Serial.print("Cambio de estado! ");
    Serial.print("Estado actual: ");
    Serial.print(estado_actual);

    Serial.print(", salida: ");
    Serial.println(salida);
    Serial.print("Nuevo estado: ");
    Serial.println(estado_futuro);
}

/**
 * Imprime en el monitor serial el valor leido en decimal antes de
modificarlo
*/
void imprimir_anter_suma_entrada(){
    Serial.print("entrada(anter): ");
    Serial.print(entrada);
    Serial.print(" :: ");
}
 

/**
 * Imprime en el monitor serial el valor leido en decimal despues de
modificarlo
*/
void imprimir_despues_suma_entrada(int i){
    Serial.print(i);
    Serial.print(": ");
}

```

```

    Serial.print(digitalRead(pin_entradas[i]));
    Serial.print("*");
    Serial.print(pow(2,i));
    Serial.print(" = ");
    Serial.print(digitalRead(pin_entradas[i])*pow(2,i));
    Serial.print(", ");
    Serial.print("entrada(despues): ");
    Serial.println(entrada);
}

/**
 * Imprime en el monitor serial la salida de la maquina, si un bit fue 0
o 1
 *
 */
void imprimir_estado_salida(int i,int salida){
    Serial.print("salida >= (pow(2,i)) : ");
    Serial.print(salida);
    Serial.print(" >= ");
    Serial.print((pow(2,i)));
    Serial.print(" : ");
    Serial.println(salida >= round(pow(2,i)) ? "HIGH" : "LOW");
}

/**
 * Imprime en el monitor serial tanto la salida como el estado actual de
la maquina
 *
 */
void imprimir_salida_i(int i,int estado){
    Serial.print("Salida ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println(estado);
}

```

Figura 378. Programa completo de Arduino

11.2.4.2.2 Programa de Arduino. Prueba y lectura de la máquina de estados (2ii)

Programa de Arduino para la lectura y prueba del Circuito Secuencial
 Nombre del programa: Arduino AR 65 Prueba y Lectura FSM

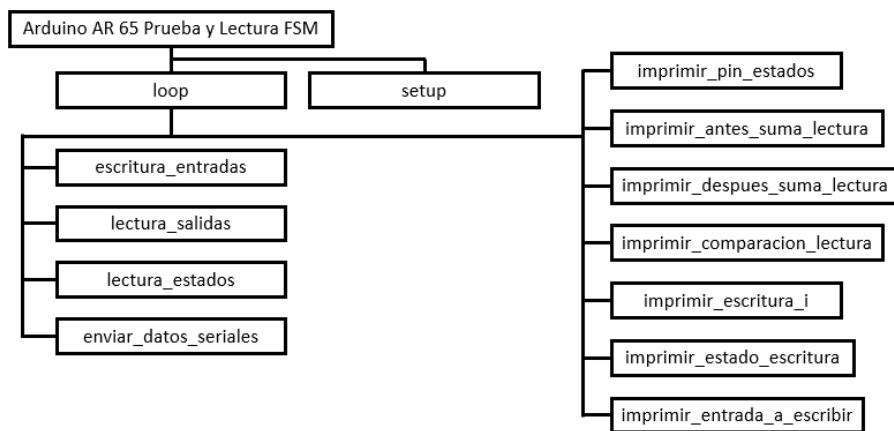


Figura 379. Diagrama a bloques del programa completo en Arduino para adquirir señales del **Circuito secuencial (d)**.

```

/*****
PROGRAMA: 26072021

Prueba y lectura de una maquina de estados finitos

Este programa se ejecuta en conjunto con:

ARDUINO (1): ARDUINO AR 65 Implementacion FSM.ino
ARDUINO (2): ARDUINO AR 65 Prueba y Lectura FSM.ino
PYTHON: Completo AR 65 FSM.py (monitor) 25062021-P

TARJETA UTILIZADA: Arduino Mega
Opcional: Utilizar LEDs con resistencias para observar el funcionamiento
del circuito

28 / julio / 2021

***** */

/***
* -----
* * Declaracion de arreglos de pines digitales
* * pin_escrituras[] -> son los pines en donde se escribirán las entradas
* a la maquina de estados finitos
  
```

```

 * pin_lecturas[] -> son los pines de donde se leeran las salidas de la
maquina de estados finitos
 * pin_estados[] -> son los pines que leera el estado en el que se
encuentra la maquina de estados finitos
 *
 * Como se observa, se pueden implementar maquinas de estados con las
siguientes caracteristicas:
 * - Hasta cuatro entradas
 * - Hasta cuatro salidas
 * - Hasta 16 estados diferentes
 *
 * Este programa se puede expandir a pines, modificando los arreglos.
* -----
-----*/
// Declare multiple analog pins:
// https://forum.arduino.cc/t/analog-pin-numbers-dont-match-standard-pinout-mega2560/610772/6
// See answer no.6 from david_2008

const int pin_escrituras[] = {2,3,12,5};
const int pin_lecturas[] = {6,7,8,9};
const byte pin_estados[] = {A0,A1,A2,A3};

/***
 * -----
-----*/
 * Declaracion de pines de control
 * pinCLK -> Define el pin del reloj que se utilizara para enviar pulsos
de reloj a la maquina de estados
 * pinReset -> Define el pin de reset que inicializara la maquina de
estados
 * -----
-----*/
int pinCLK = 10;
int pinReset = 11;

/***
 * -----
-----*/
 * Variables para el control del tiempo de muestreo
 * lastime -> guarda el ultimo instante de tiempo en el que enviaron
datos por el puerto serie
 * sampleTime -> duracion entre cada muestra tomada
 * numMuestras -> numero de muestras tomadas por cada curva
 *
 * -----
-----*/
unsigned long lastTime = 0;
unsigned long sampleTime = 100;
double firstTime = 0;

/***
 * -----
-----*/

```

```

 * Variables para la prueba de la maquina de estados
 * lectura -> Es el valor leido de la salida de la maquina
 * escritura -> Es el valor que se le escribe a la maquina
 * estado -> Es el valor leido del estado en el que se encuentra de la
maquina
 *
 *
-----
```

```

 */
int lectura = 0;
int escritura = 0;
int estado = 0;

// Ejemplo de un control para proteccion contra corto circuito para
fuente de alimentacion
/**
 * tabla_vectores_escritura_prueba[] -> contiene las entradas a escribir
a la maquina de estados finitos
*
*/
const int tabla_vectores_escritura_prueba[] =
{0B000,0B100,0B101,0B111,0B101,0B100,0B110,0B100,0B100,0B000};

/**
 * Inicializa todos los pines a utilizar, inicializa el reloj, la
frecuencia de operacion, la comunicacion serial y la comunicacion I2C
para comunicarse con los dos PCF8591
 * Para cambiar la frecuencia del PWM de pines en Arduino Mega:
 * https://forum.arduino.cc/t/mega-2560-pwm-frequency/71434/2
*
*/
void setup() {

    // Se limpian los registros que definen la frecuencia de los pines PWM
    int myEraser = 7;
    TCCR0B &= ~myEraser;
    TCCR1B &= ~myEraser;
    TCCR2B &= ~myEraser;
    TCCR3B &= ~myEraser;
    TCCR4B &= ~myEraser;

    // Se configuran los registros para la frecuencia de los pines PWM
    int myPrescaler = 3;
    TCCR0B |= myPrescaler;
    TCCR1B |= myPrescaler;
    TCCR2B |= myPrescaler;
    TCCR3B |= myPrescaler;
    TCCR4B |= myPrescaler;

    // Inicializa los pines en donde se leera el estado de la maquina
    for(int i=0;i<sizeof(pin_estados)/sizeof(pin_estados[0]);i++)
    {
        pinMode(pin_estados[i],INPUT);
    }
}
```

```

// Inicializa los pines en donde se escribiran las entradas de la
maquina
for(int i=0;i<sizeof(pin_escrituras)/sizeof(pin_escrituras[0]);i++)
{
    pinMode(pin_escrituras[i],OUTPUT);
}

// Inicializa los pines en donde se leeran las salidas de la maquina
for(int i=0;i<sizeof(pin_lecturas)/sizeof(pin_lecturas[0]);i++)
{
    pinMode(pin_lecturas[i],INPUT);
}

// Se inicializan los pines del reloj y de reset
pinMode(pinCLK,OUTPUT);
pinMode(pinReset,OUTPUT);

// Se inicializa la señal de reloj, escribiendo un voltaje PWM
analogWrite(pinCLK,127);

// --- Inicializa velocidad de transmision serial ---
Serial.begin(9600);

// Manda un pulso de reset para inicializar la maquina de estados
finitos
digitalWrite(pinReset,~HIGH);
delay(sampleTime/50);
digitalWrite(pinReset,~LOW);
delay(sampleTime-sampleTime/50);

firstTime = millis();

// --- Sincronizacion de comunicacion serial ---
Serial.println("inicio");
leer_salidas();
leer_estados();
enviar_datos_seriales();

}

int i=0;

void loop() {
    // Obtiene el siguiente valor binario a escribir a la entrada de la
    maquina de estados finitos
    if(i<(sizeof(tabla_vectores_escritura_prueba)/2)-1) { i++; } else {
        i=0; }

    // Escribe a la entrada de la maquina uno de los valores de la tabla de
    vectores de prueba
    escritura = tabla_vectores_escritura_prueba[i];

    //imprimir_entrada_a_escribir();

    // Escribe un valor a la entrada de la maquina de estados finitos
    escribir_entradas();
}

```

```

    delay(20);
    // Lee la salida y el estado en el que se encuentra la maquina de
    estados finitos
    leer_salidas();
    leer_estados();

    // Envia datos por el puerto serial
    enviar_datos_seriales();

    delay(sampleTime);
}

/**
 * Envia el valor que se le envio a la maquina, la salida y el estado de
 misma
 *
 */
void enviar_datos_seriales(){
    //Serial.print("Escritura: ");
    Serial.println(escritura);
    //Serial.print(" , Estado: ");
    Serial.println(estado);
    //Serial.print(" , Lectura: ");
    Serial.println(lectura);
}

/**
 * Escribe un valor en binario a la entrada de la maquina de estados
 finitos
 * Convierte el valor en decimal a escribir a binario
 *
 */
void escribir_entradas()
{
    int temp = escritura;
    for(int i=(sizeof(pin_escrituras)/2)-1;i>=0;i--) {
        //imprimir_estado_escritura(i,escritura);
        if(escritura>=(pow(2,i)))
        {
            escritura -= round(pow(2,i));
            digitalWrite(pin_escrituras[i],HIGH);
            //imprimir_escritura_i(i,1);
        }
        else
        {
            digitalWrite(pin_escrituras[i],LOW);
            //imprimir_escritura_i(i,0);
        }
    }
    escritura = temp;
}

/**
 * Lee un valor binario a la salida de la maquina de estados finitos
 * Convierte este valor en binario leido a un valor decimal
 *
 */

```

```

void leer_salidas()
{
    lectura = 0x00;

    for(int i=0;i<sizeof(pin_lecturas)/2;i++) {
        //imprimir_anteriores_suma_lectura();
        lectura += (byte)(digitalRead(pin_lecturas[i])*pow(2,i)+0.5);
        //imprimir_comparacion_estado(i);
    }
    //lectura = byte(lectura);
    //imprimir_despues_suma_lectura();
}

/**
 * Lee un valor binario correspondiente al estado de la maquina de
estados finitos
 * Convierte este valor en binario leido a un valor decimal
 *
 */
void leer_estados()
{
    estado = 0x00;
    for(int i=0;i<sizeof(pin_estados)/2;i++) {

        //imprimir_pin_estados(i);
        estado += (byte)(digitalRead(pin_estados[i])*pow(2,i)+0.5);
    }
}

// Las siguientes subrutinas despliegan informacion extra sobre lo que
ocurre en el programa

/**
 * Imprime en el monitor serial el estdao de los pines que leen el estado
en el que se encuentra la maquina
 *
 */
void imprimir_pin_estados(int i){
    Serial.print("pin_estados[");
    Serial.print(i);
    Serial.print("]: ");
    Serial.print(analogRead(pin_estados[1]));
    Serial.print(" : ");
    Serial.println(digitalRead(pin_estados[i]));
}

/**
 * Imprime en el monitor serial el valor de conversion a decimal antes de
modificarlo
 *
 */
void imprimir_anteriores_suma_lectura(){
    Serial.print("lectura(anteriores): ");
    Serial.print(lectura);
    Serial.print(" :: ");
}

```

```

/**
 * Imprime en el monitor serial el valor de conversion a decimal despues
de modificarlo
 *
 */
void imprimir_despues_suma_lectura(){
    Serial.print("lectura(despues): ");
    Serial.println(lectura);
}

/**
 * Imprime en el monitor serial si algun bit de la lectua de salida fue 1
o 0
 *
 */
void imprimir_comparacion_lectura(int i){
    Serial.print(i);
    Serial.print(": ");
    Serial.print(digitalRead(pin_lecturas[i]));
    Serial.print("*");
    Serial.print(pow(2,i));
    Serial.print(" = ");
    Serial.print(digitalRead(pin_lecturas[i])*pow(2,i));
    Serial.print(", ");
    Serial.print("lectura(despues): ");
    Serial.println(lectura);
}

/**
 * Imprime en el monitor serial la salida y el estado de la maquina
despues de escribirle una entrada
 *
 */
void imprimir_escritura_i(int i,int estado){
    Serial.print("Salida ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println(estado);
}

/**
 * Imprime en el monitor serial si algun bit en la escritura de la
entrada a la maquina fue 0 o 1
 *
 */
void imprimir_estado_escritura(int i,int salida){
    Serial.print("escritura >= (pow(2,i)): ");
    Serial.print(escritura);
    Serial.print(" >= ");
    Serial.print((pow(2,i)));
    Serial.print(" : ");
    Serial.println(escritura >= round(pow(2,i)));
}

/**
 * Imprime en el monitor serial la entrada a escribir a la maquina
*

```

```
 */
void imprimir_entrada_a_escribir(int i){
    Serial.print("Entrada a escribir ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println(tabla_vectores_escritura_prueba[i]);
}
```

Figura 380. Programa completo de Arduino

11.2.4.3 Programas en Python (3)

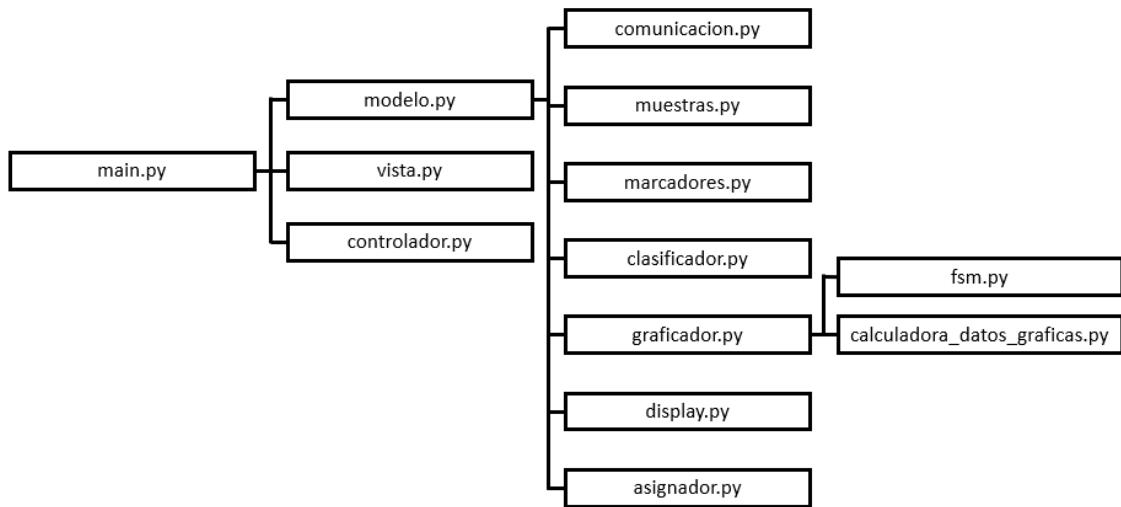


Figura 381. Diagrama de clases para el **círcuito secuencial (d)**.

11.2.4.3.1 Clase fsm.py (3i)

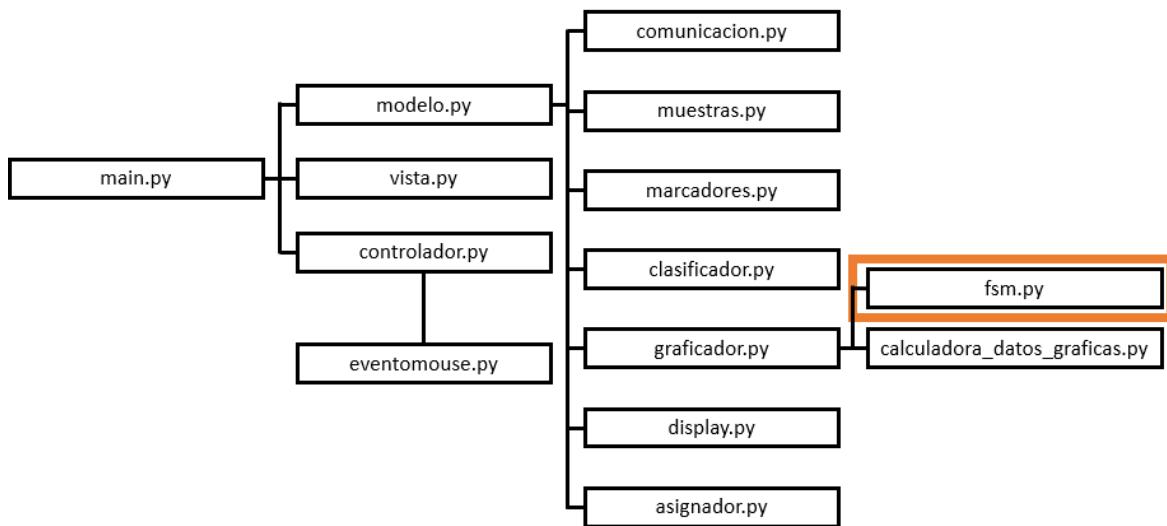


Figura 382. Ubicación de la clase fsm.py.

Esta clase contiene todos los datos que especifican el comportamiento del circuito secuencial o máquina de estados finitos.



Figura 383. Métodos, atributos y propiedades de la clase fsm.py.

Donde:

indica que es una **clase**.

 indica que es un **método**.

 indica que es una **propiedad**.

 indica que es un **atributo**.

```
# ****
#
# Clase: --> FSM
# Modulo: -> FSM.py
#
# Descripción:
# - Describir la maquina de estados finitos
#
#
# Fecha: julio 26/2021
#
# ****

class fsm():
    def __init__(self):
        print("")
        print(" CONSTRUCTOR: Clase: fsm")

        # Copiar del programa de ARDUINO IMPLEMENTACION FSM todos los datos de la
        # maquina de estados y adaptarlo a la sintaxis de python

        # -----
        # num_estados -> indica cuantos estados tiene la maquina
        # bits_transicion -> Indica cuantos bits tiene las transiciones que permiten cambiar
        # entre estados
        # bits_salida -> Indica la cantidad de bits de salida de la maquina de estados
        # -----


        self.num_estados = 4
        self.bits_transicion = 3
        self.bits_salida = 3
        self.estado_inicial = 0
        self.estado_final = 1

        # -----
        # Las siguientes tablas especifican el funcionamiento de la maquina de estados finitos
        # El tamano de estas tablas esta determinado por las constantes num_estados,
        # bits_salida y bits_transicion
```

```

# Cada estado de la maquina corresponde a una entrada de la tabla, como se indica
en cada una de las tablas.
#
# tabla_transiciones_estados -> Indica cuales son las transiciones con las que la
maquina cambia de estado
# Las transiciones que no aparecen, se asume que la maquina no
cambia de estado
# IMPORTANTE: Se completan la tabla con -1 para las transiciones
en las que no hay cambio de estado
#
# tabla_transiciones_estado_futuro -> Indica el estado al cual va a cambiar la maquina
en el siguiente pulso
# de reloj. Cada estado especificado en esta tabla corresponde
a la
# transicion de la tabla tabla_transiciones_estados
#
# tabla_output_por_estado -> Indica la salida del estado actual de la maquina
# -----

```

self.tabla_transiciones_estados = [

- [0B100,0B101], # estado 0: Apagado
- [0B000,0B001,0B010,0B011,0B101,0B111], # estado 1:

Encendido funcionando

- [0B000,0B001,0B010,0B011,0B110,0B111], # estado 2:

Corto circuito

- [0B000,0B001,0B010,0B011,0B100,0B101] # estado 3:

Reset

-]

self.tabla_transiciones_estado_futuro = [

- [1,1], # estado 0
- [0,0,0,0,2,2], # estado 1
- [0,0,0,0,3,3], # estado 2
- [0,0,0,0,1,2] # estado 3

-]

self.tabla_output_por_estado = [

- 0B000, # estado 0
- 0B100, # estado 1
- 0B001, # estado 2
- 0B011 # estado 3

-]

self.tabla_nombres_estados = [

```
'q0 Apagado',
'q1 Encendido funcionando',
'q2 Corto circuito',
'q3 Reset'
]

self.tabla_nombres_salidas = [
    'bit 2: Relevador ON/OFF',
    'bit 1: LED de Reset',
    'bit 0: LED de Corto circuito'
]

self.tabla_nombres_entradas = [
    'bit 2: Boton de Alimentacion',
    'bit 1: Boton de Reset',
    'bit 0: Sensor de Corto Circuito'
]
```

Figura 384. Código completo de la clase fsm.py

11.2.4.3.2 Clase calculadora_datos_graficas.py (3ii)

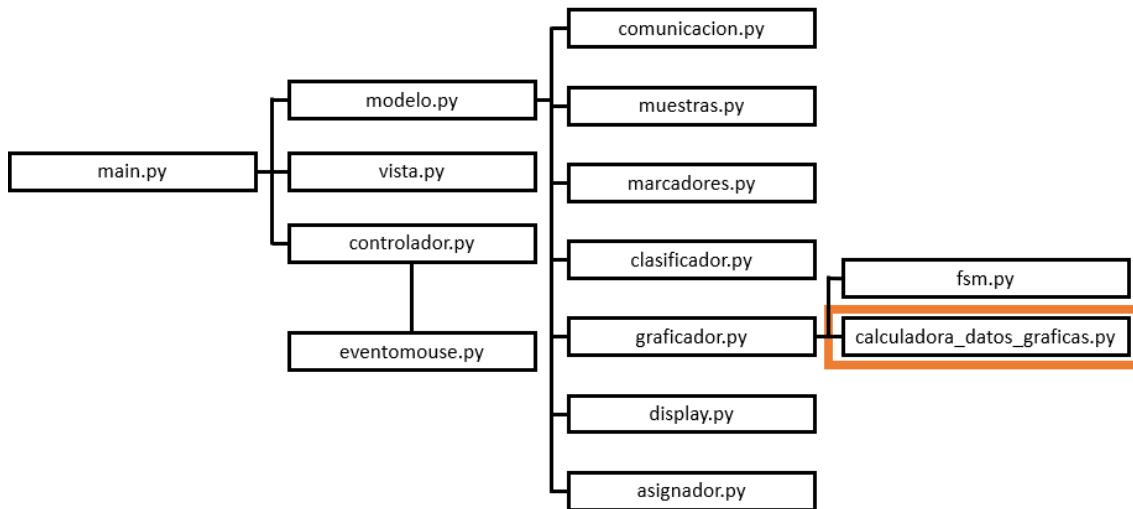


Figura 385. Ubicación de la clase calculadora_datos_gráficas.py.

Esta clase contiene los siguientes métodos y atributos:

- m.calculadora_datos_gráficas.calculadora_datos_gráficas
- __init__(self)
- ec_recta_dos_puntos(self,p1,p2)
- ec_recta_punto_pendiente_perpendicular(self,p1,m)
- puntos_equidistantes_sobre_recta_desde_punto(self,p1,m,d)
- list_duplicates_of(self,seq,item)
- decimal_a_binario(self,decimal,num_bits)
- obtener_bits_cambiantes(self,fsm1,transiciones,estado)
- obtener_str_de_binario_transicion(self,bits,binario,diff)
- circle_line_segment_intersection(self,circle_center,circle_radius, pt1, pt2, full_line=True, tangent_tol=1e-9)
- obtener_puntos_lineas_transicion(self,canvas2,xcoords,ycoords,c_radio,length_lineas,c_actual,c_transicion)
- obtener_coordenadas_labels(self,p1,p2)
- get_intersections(self, x0, y0, r0, x1, y1, r1)

Figura 386. Métodos y atributos de la clase calculadora_datos_gráficas.py.

Donde:

- indica que es una **clase**.
- indica que es un **método**.
- **p** indica que es una **propiedad**.
- **f** indica que es un **atributo**.

```

# ****
#
# Clase: --> Calculadora_datos_graficas
# Modulo: -> calculadora_datos_graficas.py
#
# Descripción:
#   Auxiliar de graficador.py que calcula, entre otras cosas, coordenadas de etiquetas,
#   flechas y circulos.
#
# Fecha: julio 28/2021
#
# ****

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import numpy as np
import math
import statistics
from engineering_notation import EngNumber
import cv2

class calculadora_datos_graficas:
    # -----
    #     CONSTRUCTOR
    # -----
    def __init__(self):

        print("")
        print(" CONSTRUCTOR: Clase: Calculadora_datos_graficas")

    # -----
    # Calcula la ecuacion de una recta (pendiente m y ordenada al origen b) dados dos
    # puntos
    #
    # -----
    def ec_recta_dos_puntos(self,p1,p2):
        xp1,yp1 = p1
        xp2,yp2 = p2
        m = float('inf')
        b = float('inf')
        if xp2 - xp1 != 0:
            m = (yp2 - yp1) / (xp2 - xp1)

```

```

b = (-xp1*yp2 - yp1*xp2) / (xp2 - xp1)

#print("m: "+str(m)+" , b: "+str(b))
return m,b

#
# Calcula la ecuacion de una recta perpendicular (pendiente m y ordenada al origen b)
# dado un punto y una pendiente
#
# -----
def ec_recta_punto_pendiente_perpendicular(self,p1,m):
    xp1,yp1 = p1

    if m == 0:
        mp = float('inf')
    else:
        mp = -1 / m

    bp = (-mp*xp1 + yp1)
    #print("m: "+str(mp)+" , b: "+str(bp))
    return mp,bp

#
# Calcula el valor absoluto de la distancia dados dos puntos
#
# -----
def distancia_entre_puntos(self,p1,p2):
    xp1,yp1 = p1
    xp2,yp2 = p2

    d = math.sqrt((xp2-xp1)**2+(yp2-yp1)**2)
    return d

#
# Calcula las coordenadas de dos puntos equidistantes dado un punto, la pendiente de
# una recta y la distancia entre ellos.
#
# -----
def puntos_equidistantes_sobre_recta_desde_punto(self,p1,m,d):
    xp1,yp1 = p1
    x3a = xp1
    x3b = xp1
    y3a = yp1 + d
    y3b = yp1 - d

```

```

if m < 9223372036854775807 and m != 0: # la recta no es vertical ni horizontal
    ec_puntos = np.poly1d([1+m**2,-2*xp1*(1+m**2),(1+m**2)*xp1-d**2])
    print(ec_puntos)
    x3a,x3b = ec_puntos.r
    print(ec_puntos.r)
    y3a = m * x3a - m * xp1 + yp1
    y3b = m * x3b - m * xp1 + yp1
if m == 0: # la recta es horizontal
    #print('aqui')
    x3a = xp1 - d
    x3b = xp1 + d
    y3a = yp1
    y3b = yp1
    ""print(x3a)
    print(y3a)
    print(x3b)
    print(y3b)"""
return int(x3a),int(y3a),int(x3b),int(y3b)

# -----
# Obtiene todos los indices de un elemento repetido en una lista.
# Tomado de https://stackoverflow.com/questions/5419204/index-of-duplicates-items-in-a-python-list
# -----
def list_duplicates_of(self,seq,item):
    start_at = -1
    locs = []
    while True:
        try:
            loc = seq.index(item,start_at+1)
        except ValueError:
            break
        else:
            locs.append(loc)
            start_at = loc
    return locs

# -----
# Obtiene una representacion en binario de un numero en decimal
#
# -----
def decimal_a_binario(self,decimal,num_bits):
    #str_bin = '0b'

```

```

str_bin = ""
for i in range(num_bits-1,-1,-1):
    bstr = decimal & int(pow(2,i))
    bstr >>= i
    str_bin +=str(bstr)
return str_bin

# -----
# Obtiene los bits que son diferentes entre dos transiciones de estados
#
# -----
def obtener_bits_cambiantes(self,fsm1,transiciones,estado):
    diff = 0
    if len(transiciones) > 1:
        for k in range(1,len(transiciones)):
            diff_orx      = fsm1.tabla_transiciones_estados[estado][transiciones[0]]
            fsm1.tabla_transiciones_estados[estado][transiciones[k]] ^ 
            diff |= diff_orx
    return diff

# -----
# Obtiene una representacion en binario de un numero en decimal, que ademas incluye
condiciones de no importa (representado por una "X")
#
# -----
def obtener_str_de_binario_transicion(self,bits,binario,diff):
    str_transicion =""
    for k in range(bits,-1,-1):
        if diff >= math.pow(2,k):
            diff -= math.pow(2,k)
            str_transicion += 'X'
        else:
            binstr = binario & int(math.pow(2,k))
            binstr >>= k
            str_transicion += str(binstr)
    return str_transicion

# -----
# Obtiene las coordenadas de las intersecciones entre un circulo y una recta.
# Tomado de https://stackoverflow.com/questions/30844482/what-is-most-efficient-way-
to-find-the-intersection-of-a-line-and-a-circle-in-py
#
# -----
def circle_line_segment_intersection(self, circle_center, circle_radius, pt1, pt2,
full_line=True, tangent_tol=1e-9):

```

```
""" Find the points at which a circle intersects a line-segment. This can happen at 0, 1, or 2 points.
```

```
:param circle_center: The (x, y) location of the circle center
:param circle_radius: The radius of the circle
:param pt1: The (x, y) location of the first point of the segment
:param pt2: The (x, y) location of the second point of the segment
:param full_line: True to find intersections along full line - not just in the segment. False will just return intersections within the segment.
:param tangent_tol: Numerical tolerance at which we decide the intersections are close enough to consider it a tangent
:return Sequence[Tuple[float, float]]: A list of length 0, 1, or 2, where each element is a point at which the circle intercepts a line segment.
```

Note: We follow: <http://mathworld.wolfram.com/Circle-LineIntersection.html>

```
"""
(p1x, p1y), (p2x, p2y), (cx, cy) = pt1, pt2, circle_center
(x1, y1), (x2, y2) = (p1x - cx, p1y - cy), (p2x - cx, p2y - cy)
dx, dy = (x2 - x1), (y2 - y1)
dr = (dx ** 2 + dy ** 2)**.5
big_d = x1 * y2 - x2 * y1
discriminant = circle_radius ** 2 * dr ** 2 - big_d ** 2

if discriminant < 0: # No intersection between circle and line
    return []
else: # There may be 0, 1, or 2 intersections with the segment
    intersections = [
        [int(cx + (big_d * dy + sign * (-1 if dy < 0 else 1) * dx * discriminant**.5) / dr ** 2),
         int(cy + (-big_d * dx + sign * abs(dy) * discriminant**.5) / dr ** 2)]
        for sign in ((1, -1) if dy < 0 else (-1, 1))] # This makes sure the order along the
    segment is correct
    if not full_line: # If only considering the segment, filter out intersections that do not fall within the segment
        fraction_along_segment = [(xi - p1x) / dx if abs(dx) > abs(dy) else (yi - p1y) / dy
for xi, yi in intersections]
        intersections = [pt for pt, frac in zip(intersections, fraction_along_segment) if 0 <= frac <= 1]
    if len(intersections) == 2 and abs(discriminant) <= tangent_tol: # If line is tangent to circle, return just one point (as both intersections have same location)
        return [intersections[0]]
    else:
        return intersections
```

```

# -----
# Obtiene las coordenadas utilizadas para dibujar lineas entre estados para el diagrama
de estados
# Se siguió la siguiente secuencia de pasos:
# 1. Encontrar las intersecciones entre la circunferencia del estado actual y la linea que
une los centros de la circunferencia del estado actual y la del centro que es de transicion.
# 2. Calcular la recta perpendicular a la línea generada en el paso anterior sobre el punto
de intersección.
# 3. Calcular la intersección de la recta perpendicular anterior con la circunferencia del
estado actual. Se deben obtener dos soluciones.
# Hay dos casos especiales que son que la pendiente de la recta del paso 1 sea
horizontal o vertical:
# a. Cuando la pendiente m tiende a infinito, se calcula las coordenadas de los puntos
que equidistan de la intersección del paso 1 y se ubican sobre la recta perpendicular
# b. Si la pendiente m no tiende a infinito o m es igual a cero, se utiliza un método más
sencillo. Se toma la intersección del paso 1 como centro de una nueva circunferencia
# y se buscan las intersecciones con la circunferencia del estado actual.
# 4. Calcular las intersecciones con la circunferencia de transición
# Hay dos casos especiales que igual son dependiendo de la pendiente de la recta del
paso 1 si es horizontal o vertical:
# a. Cuando la pendiente m tiende a infinito, se realiza el mismo procedimiento descrito
en los pasos 1, 2 y 3.a, esta vez con la circunferencia de transición.
# b. Si la pendiente m no tiende a infinito, se calcula la ecuación que une las
circunferencias del estado actual y la de transición en los puntos calculados en el paso 3
# para encontrar las intersecciones con la circunferencia de transición.
# -----
def
obtener_puntos_lineas_transicion(self, canvas2, xcoords, ycoords, c_radio, length_lineas, c_a
ctual, c_transicion):
    lineas_transiciones = []

    # Se obtienen las intersecciones de la recta que une los centros de las circunferencias
    # del estado actual y de transición y la circunferencia del estado actual
    inters_origen =
        self.circle_line_segment_intersection((xcoords[c_actual], ycoords[c_actual]), c_radio[1], (xco
ords[c_actual], ycoords[c_actual]), (xcoords[c_transicion], ycoords[c_transicion]), False, 1e-9)
    #print("inters_origen: "+str(inters_origen), "len(inters_origen): "+str(len(inters_origen)))

    # Calcula la ecuación de la recta que une los centros de las circunferencias del estado
    # actual y de transición
    m, b =
        self.ec_recta_dos_puntos(inters_origen[0], [xcoords[c_transicion], ycoords[c_transicion]])
    #recta = np.poly1d([m,b])

```

```

# Calcula la ecuacion de la recta perpendicular a la anterior y sobre el punto de
interseccion
mp, bp = self.ec_recta_punto_pendiente_perpendicular(inters_origen[0],m)
recta_p = np.poly1d([mp,bp])
#print('recta perpendicular calculada')
#print("m: "+str(m)+" , mp: "+str(mp))

if mp > 9223372036854775807:# la recta recta_p es perpendicular
    #print('calcular puntos con recta perpendicular')
    xp1,yp1,xp2,yp2 =
self.puntos_equidistantes_sobre_recta_desde_punto((inters_origen[0][0],inters_origen[0][1]
]),mp,length_lineas)
    lineas_transiciones.append([xp1,yp1])
    lineas_transiciones.append([xp2,yp2])
    #print(lineas_transiciones)

else:# la recta recta_p no es perpendicular
    if m > 9223372036854775807:# la recta recta_p es horizontal (la recta que une los
centros de las circunferencias del estado actual y de transicion es vertical)
        #print('calcular puntos con circunferencia especial')
        # Se realiza un ajuste para obtener las intersecciones con la circunferencia del
estadio actual utilizando el punto de interseccion obtenido anteriormente como centro de
una circunferencia
        lineas_transiciones =
self.circle_line_segment_intersection(inters_origen[0],length_lineas,(xcoords[c_actual]-
50,int(recta_p(xcoords[c_actual]))),(xcoords[c_transicion]+50,int(recta_p(xcoords[c_transic
ion]))),True,1e-9)

    else:
        #print('calcular puntos con circunferencia')
        # Se obtienen las intersecciones con la circunferencia del estadio actual utilizando
el punto de interseccion obtenido anteriormente como centro de una circunferencia
        lineas_transiciones =
self.circle_line_segment_intersection(inters_origen[0],length_lineas,(xcoords[c_actual],int(r
ecta_p(xcoords[c_actual]))),(xcoords[c_transicion],int(recta_p(xcoords[c_transicion]))),Tru
e,1e-9)

#print('primeras dos transiciones calculadas')
#print("lineas_transiciones: "+str(lineas_transiciones))

# Se calculan las intersecciones con la circunferencia de transicion
if m < 9223372036854775807:# la recta que une los centros de las circunferencias del
estadio actual y de transicion no es vertical

```

```

for i in [0,1]:
    # Se calcula una recta que une alguna de las intersecciones anteriores con la
    #circunferencia de transicion
    recta_transicion = np.poly1d([m,-
m*lineas_transiciones[i][0]+lineas_transiciones[i][1]])

    #print(str(lineas_transiciones[i][0])+
"+str(recta_transicion(lineas_transiciones[i][0])))"
    #print(str(xcoords[c_transicion])+
"+str(recta_transicion(xcoords[c_transicion])))"

    # Puntos de inicio y final de esta recta
    ptr0 = (lineas_transiciones[i][0],recta_transicion(lineas_transiciones[i][0]))
    ptr1 = (xcoords[c_transicion],recta_transicion(xcoords[c_transicion]))

    #print("ptr0: "+str(ptr0))
    #print("ptr1: "+str(ptr1))

    # Se obtienen las coordenadas de las intersecciones con la circunferencia de
    #transicion con la recta recta_transicion
    p_transicion =
self.circle_line_segment_intersection((xcoords[c_transicion],ycoords[c_transicion]),c_radio
[1],ptr0,ptr1,False,1e-9)

    p_transicion = np.reshape(p_transicion,-1)
    p_transicion = p_transicion.tolist()
    lineas_transiciones.append(p_transicion)

else:# la recta que une los centros de las circunferencias del estado actual y de
transicion es vertical
    # Obtiene las coordenadas de la intersección entre la recta que une los centros de
    #las circunferencias de estado actual y de transicion y la circunferencia de transicion
    inters_fin =
self.circle_line_segment_intersection((xcoords[c_transicion],ycoords[c_transicion]),c_radio
[1],(xcoords[c_transicion],ycoords[c_transicion]),(xcoords[c_actual],ycoords[c_actual]),Fals
e,1e-9)
    #print("inters_fin: "+str(inters_fin), "len(inters_fin): "+str(len(inters_fin)))

    #cv2.circle(canvas2, tuple(inters_fin[0]), 5, (0,255,0), 3)

    # Calcula la recta perpendicular a la anterior en el punto de la intersección
    mp,bp = self.ec_recta_punto_pendiente_perpendicular(inters_fin[0],m)
    #recta_p_aux = np.poly1d([mp,bp])
    #print(": "+str(recta_p_aux))

```

```

# Se obtienen los puntos equidistantes al punto de intersección sobre la recta
perpendicular.
xp1,yp1,xp2,yp2 =
self.puntos_equidistantes_sobre_recta_desde_punto((inters_fin[0][0],inters_fin[0][1]),mp,le
ngth_lineas)

lineas_transiciones.append([xp1,yp1])
lineas_transiciones.append([xp2,yp2])
#print(lineas_transiciones)

#print("lineas_transiciones: "+str(lineas_transiciones))
#for i in [0,1,2,3]:
#cv2.circle(canvas2, tuple(lineas_transiciones[i]), 5, (0,0,0), 3)

return lineas_transiciones

# -----
# Obtiene las coordenadas de las etiquetas para las líneas de transiciones del diagrama
de estados.
#
# -----
def obtener_coordenadas_labels(self,p1,p2):
    xp1,yp1 = p1
    xp2,yp2 = p2

    x_lb = min(xp1,xp2) + int(abs(xp2 - xp1)/2)
    y_lb = min(yp1,yp2) + int(abs(yp2 - yp1)/2)

    if xp1 < xp2:
        if yp1 < yp2:
            x_lb += 0
            y_lb -= 0
        elif yp1 > yp2:
            x_lb -= 30
            y_lb -= 8
        else:
            x_lb -= 17
            y_lb -= 8
    elif xp1 > xp2:
        if yp1 < yp2:
            x_lb += 5
            y_lb += 10
        elif yp1 > yp2:

```

```

        x_lb += 5
        y_lb += 0
    else:
        x_lb -= 17
        y_lb += 18
    else:
        if yp1 < yp2:
            x_lb += 2
        elif yp1 > yp2:
            x_lb -= 33
    return x_lb,y_lb

# -----
# Obtiene las intersecciones dadas dos circunferencias, dadas las coordenadas de su
centro y radio.
# Tomado de https://stackoverflow.com/questions/55816902/finding-the-intersection-of-
two-circles
# -----
def get_intersections(self, x0, y0, r0, x1, y1, r1):
    # circle 1: (x0, y0), radius r0
    # circle 2: (x1, y1), radius r1

    d = math.sqrt((x1-x0)**2 + (y1-y0)**2)

    # non intersecting
    if d > r0 + r1 :
        return None
    # One circle within other
    if d < abs(r0-r1):
        return None
    # coincident circles
    if d == 0 and r0 == r1:
        return None
    else:
        a=(r0**2-r1**2+d**2)/(2*d)
        h=math.sqrt(r0**2-a**2)
        x2=x0+a*(x1-x0)/d
        y2=y0+a*(y1-y0)/d
        x3=x2+h*(y1-y0)/d
        y3=y2-h*(x1-x0)/d

        x4=x2-h*(y1-y0)/d
        y4=y2+h*(x1-x0)/d

```

```
    return (x3, y3, x4, y4)
```

Figura 387. Código completo de la clase calculadora_datos_graficas.py

11.2.4.3.3 Clase graficador.py (3iii)

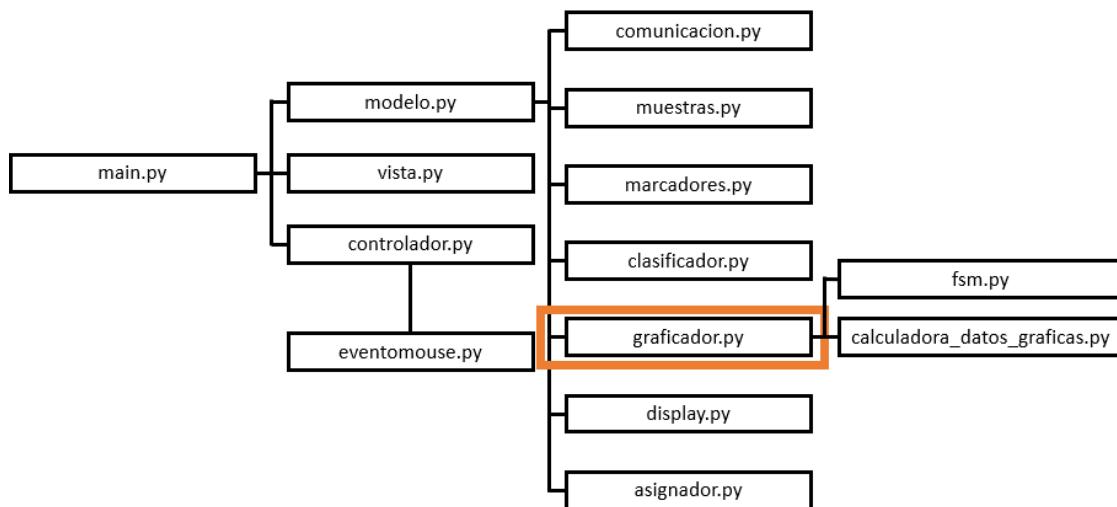


Figura 388. Ubicación de la clase graficador.py.

Esta clase contiene los siguientes métodos y atributos:

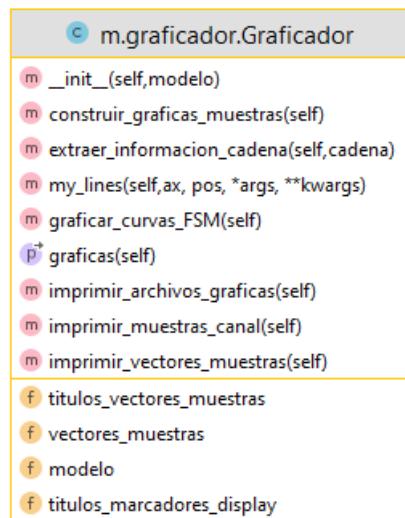


Figura 389. Métodos y atributos de la clase graficador.py.

Donde:

- c indica que es una **clase**.
- m indica que es un **método**.
- p⁺ indica que es una **propiedad**.
- f indica que es un **atributo**.

```

# ****
#
# Clase: --> Graficador
# Modulo: -> graficador.py
#
# Descripción:
# - Graficar los datos obtenidos por arduino
# - Dibujar, mostrar informacion
#
# Fecha: julio 28/2021
#
# ****

from m.calculadora_datos_graficas import calculadora_datos_graficas
from m.fsm import fsm

import numpy as np
import cv2

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.ticker import EngFormatter

import graphviz as gv

from collections import deque

import os

from engineering_notation import EngNumber
import math
import statistics

graficas = []

class Graficador():

```

```

# -----
#      CONSTRUCTOR
# -----
def __init__(self,modelo):

    print("")
    print(" CONSTRUCTOR: Clase: Graficador")
    self.modelo = modelo
    self.vectores_muestras = modelo.vectores_muestras
    self.titulos_vectores_muestras = modelo.titulos_vectores_muestras
    self.titulos_marcadores_display = modelo.titulos_marcadores_display

def construir_graficas_muestras(self):

    # --- Construccion AUTOMATICA de la graficas ---
    titulos_marcadores = list(self.titulos_marcadores_display)

    global graficas
    graficas = []

    muestras = self.vectores_muestras

    for i in range(len(titulos_marcadores)):
        titulo_grafica = self.titulos_marcadores_display[titulos_marcadores[i]]
        nombre_vectores_grafica = self.extraer_informacion_cadena(titulo_grafica)

        # --- Construccion AUTOMATICA de cada GRAFICA, a partir de la
        #   informacion proporcionada en la clase Modelo en las estructuras:
        #   -> titulos_vectores_muestras <- Vector
        #   -> titulos_marcadores_display <- Diccionario
        #   Estas estructuras de datos se conectan con los nombres de los
        #   vectores que se encuentran en el Diccionario muestras
        #   -> vectores_muestras <- Diccionario
        #
        plt.figure(i)
        # --- Graficas VECTOR A vs. muestras ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 1:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      label = nombre_vectores_grafica[0][0] + ' vs. muestras')
        # --- Graficas VECTOR A vs VECTOR B ---
        if len(nombre_vectores_grafica) == 1 and len(nombre_vectores_grafica[0]) == 2:
            plt.plot(muestras[nombre_vectores_grafica[0][0]],
                      muestras[nombre_vectores_grafica[0][1]],
                      label = nombre_vectores_grafica[0][0] + ' vs. ' +

```

```

        nombre_vectores_grafica[0][1])
# --- Graficas MULTIPLES tipo: VECTOR A vs VECTOR B ---
if len(nombre_vectores_grafica) > 2:
    for g in range(len(nombre_vectores_grafica)):
        plt.plot(muestras[nombre_vectores_grafica[g][0]],
                  muestras[nombre_vectores_grafica[g][1]],
                  label = nombre_vectores_grafica[g][0] + ' vs. ' +
                  nombre_vectores_grafica[g][1])
    plt.title = 'Grafica ' + str(i)
    plt.ylabel = 'y label'
    plt.xlabel = 'x label'
    plt.legend()

# --- Guardado de las gráficas en:
#   1. "Archivo en disco [*.jpg] para su uso posterior
#       por parte de la Clase Display
#   2. El arreglo GLOBAL: 'graficas', el cual contiene
#       la ruta en donde se guardó la gráfica correspondiente
#
grafica = 'g/grafica_' + str(i) + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

def extraer_informacion_cadena(self,cadena):

#print("") 
#print(" cadena original = ",cadena)

# --- Separa grupos de letras ---
subcadena = ""
subtitulos = []
ignorar_caracter = False
num_char = 0
for ch in cadena:
    num_char += 1
    # --- Ignorar los caracteres "espacio" [' '] y "coma" [,] ---
    if ch == ' ' or ch == ',':
        ignorar_caracter = True
        if num_char != 1:
            subtitulos.append(subcadena)
            subcadena = ""
    else:
        subcadena += ch
    if num_char == len(cadena): subtitulos.append(subcadena)

```

```

# --- Genera tuplas de dos elementos ---
tupla = []
arreglo_tuplas = []
for i in range(len(subtitulos)):
    # --- Detecta conector ['vs'] ó espacio [' '] --
    if subtitulos[i] == 'vs' or subtitulos[i] == " ":
        dummy = 0 # <-- No hacer nada ---
    else:
        tupla.append(subtitulos[i])
    if subtitulos[i] == " " or i == (len(subtitulos)-1):
        arreglo_tuplas.append(tupla)
        tupla = []

return arreglo_tuplas

# Tomado de https://stackoverflow.com/questions/20036161/can-we-draw-digital-waveform-graph-with-pyplot-in-python-or-matlab
def my_lines(self,ax, pos, *args, **kwargs):
    if ax == 'x':
        for p in pos:
            plt.axvline(p, *args, **kwargs)
    else:
        for p in pos:
            plt.axhline(p, *args, **kwargs)

def graficar_curvas_FSM(self):

    global graficas
    graficas = []

    muestras = self.vectores_muestras
    muestras_llaves = list(muestras)
    #print('muestras_llaves: '+str(muestras_llaves))

    # Se obtiene el numero de series de muestras para cada canal (ej. para 300 muestras hay 6 curvas).
    lim_max = len(muestras[muestras_llaves[0]])

    fsm1 = fsm()
    calc = calculadora_datos_graficas()

```

```

# -----
# Algunas referencias que consulte para realizar el código, son las siguientes:
# Draw digital waveforms
#     https://stackoverflow.com/questions/20036161/can-we-draw-digital-waveform-
graph-with-pyplot-in-python-or-matlab
#     https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.step.html
#
# Creating a list using range()
#     https://www.kite.com/python/answers/how-to-create-an-array-of-numbers-1-to-n-in-
python
#
# Modify axis ticks
#     https://stackoverflow.com/questions/12608788/changing-the-tick-frequency-on-x-or-
y-axis-in-matplotlib
#     https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html
#
# Available colors for matplotlib
#     https://matplotlib.org/stable/gallery/color/named_colors.html
#
# Create list of lists
#     https://thispointer.com/how-to-create-and-initialize-a-list-of-lists-in-python/
#
# Find indexes of same list elements
#     https://stackoverflow.com/questions/5419204/index-of-duplicates-items-in-a-python-
list
#
# Find coordinates from intersections between a circle and a line
#     https://stackoverflow.com/questions/30844482/what-is-most-efficient-way-to-find-
the-intersection-of-a-line-and-a-circle-in-py
# See answer by Peter
#
# Python knows maximum integer value
#     https://www.delftstack.com/howto/python/python-max-int/
#
# Graphviz for Python
#     https://graphviz.readthedocs.io/en/stable/manual.html#basic-usage
#     https://graphviz.readthedocs.io/en/stable/examples.html#fsm-py
#
# Python cv2 resize an image, concatenate an image
#     https://www.tutorialkart.com/opencv/python/opencv-python-resize-image/
#     https://www.geeksforgeeks.org/image-resizing-using-opencv-python/
#     https://www.geeksforgeeks.org/concatenate-images-using-opencv-in-python/
#

```

```

# Delete a file
# https://careerkarma.com/blog/python-delete-file/
#     https://stackoverflow.com/questions/3430372/how-do-i-get-the-full-path-of-the-
current-files-directory
# -----
-----
colores_lineas = ['tab:blue','tab:orange','tab:green']

# --- Grafica 0 : Entrada, Estado y Salida en el tiempo -----
plt.figure(0)
for i in range(0,len(muestras_llaves)):
    plt.subplot(3, 1, i+1)

    #Se colocan las lineas auxiliares de las graficas.
    self.my_lines('x',      list(range(len(muestras[muestras_llaves[i]])+1)),      color=' .5',
linewidth=0.5)
    self.my_lines('y',      list(range(int(max(muestras[muestras_llaves[i]])+1))),   color=' .5',
linewidth=0.5)

    # Se grafican entrada, estado y salida.
    plt.step(range(0,lim_max),muestras[muestras_llaves[i]],      linewidth      =
2,color=colores_lineas[i], where='post', label=muestras_llaves[i])

    # Se modifican los ejes para que muestren cada numero entero.
    plt.xticks(np.arange(0,len(muestras[muestras_llaves[i]])+1,step = 1))
    plt.yticks(np.arange(0,max(muestras[muestras_llaves[i]])+1,step = 1))

    plt.ylabel(muestras_llaves[i])
    plt.legend()

    plt.xlabel('Muestras')

plt.subplot(3, 1, 1)
plt.title("Señales logicas en el tiempo")

grafica = 'g/00_Senales_binarias_en_el_tiempo' + ".jpg"
plt.savefig(grafica)
graficas.append(grafica)

# --- Grafica 1 : Tabla de estados -----
-----
# --- Generacion de un canvas de w = 240 x h = 180 ---

```

```

canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# Se colocan los titulos de la tabla.
cv2.putText(canvas2,"Tabla de estados de la maquina de estados finitos", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Estado", (30, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Salida", (140, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"Entrada", (250, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,":", (320, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Estado actual", (345, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,">", (465, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Estado futuro", (500, 65),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

# Se dibujan lineas para dividir la tablas.
cv2.rectangle(canvas2, (20,50),(620,48),(0,0,0), -1, cv2.LINE_AA)
cv2.rectangle(canvas2, (20,71),(620,73),(0,0,0), -1, cv2.LINE_AA)

ajuste_espacio = 0
str_transiciones = [""] for i in range(0,len(fsm1.tabla_transiciones_estado_futuro))]
estados_transiciones = [[0]] for i in range(0,len(fsm1.tabla_transiciones_estado_futuro))]

for i in range(0,fsm1.num_estados):
    # Se obtiene el estado y salida correspondiente.
    str_bin_estado =
    calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
    str_bin_salida =
    calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida)

    # Se escriben los textos del estado y salida correspondiente.

```

```

        cv2.putText(canvas2,"q"+str(i)+":           "+str_bin_estado,      (30,
70+(i+ajuste_espacio+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1,
cv2.LINE_AA)
        cv2.putText(canvas2,str_bin_salida,           (140,
70+(i+ajuste_espacio+1)*20),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1,
cv2.LINE_AA)

        #print("fsm1.tabla_transiciones_estados["+str(i)+"]:
"+str(fsm1.tabla_transiciones_estados[i])+"
"+str(len(fsm1.tabla_transiciones_estados[i])))           , len:

j = 0

while j < len(fsm1.tabla_transiciones_estado_futuro[i]):
    # Se obtiene el texto de la transicion correspondiente, si hay mas de una posible,
se agregan condiciones de no importa con "X".
    transiciones = calc.list_duplicates_of(fsm1.tabla_transiciones_estado_futuro[i],fsm1.tabla_transiciones_e
stado_futuro[i][j])
    diff = calc.obtener_bits_cambiantes(fsm1,transiciones,i)
    str_transicion = calc.obtener_str_de_binario_transicion(fsm1.bits_transicion-
1,fsm1.tabla_transiciones_estados[i][transiciones[0]],diff)

    y_coor = 70+(i+ajuste_espacio+1)*20

    # Se colocan los textos para la parte de transiciones de estados.
    cv2.putText(canvas2,str_transicion,           (265,
y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,":", (320, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0,
0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"q"+str(i),           (395,
y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"->", (465, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5,
(0, 0, 0), 1, cv2.LINE_AA)

    cv2.putText(canvas2,"q"+str(fsm1.tabla_transiciones_estado_futuro[i][transiciones[0]]),
(542, y_coor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    # Se guardan los textos transiciones calculadas y los estados a los que cambia.
    str_transiciones[i].append(str_transicion)
    estados_transiciones[i].append(fsm1.tabla_transiciones_estado_futuro[i][j])

    j = j+len(transiciones)
    ajuste_espacio += 1

```

```

str_transicion = ""

# Ajuste de las listas, ya que el primer dato no contiene informacion util.
str_transiciones[i].pop(0)
estados_transiciones[i].pop(0)

# Se colocan lineas que dividen cada estado.
cv2.rectangle(canvas2, (238,48),(240,68+(i+ajuste_espacio+1)*20),(0,0,0), -1,
cv2.LINE_AA)
cv2.rectangle(canvas2,
(20,65+(i+ajuste_espacio+1)*20),(620,67+(i+ajuste_espacio+1)*20),(0,0,0), -1,
cv2.LINE_AA)

# Se colocan textos de notas adicionales.
cv2.putText(canvas2,"Los estados son: "+str(fsm1.tabla_nombres_estados),(20,
390),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las salidas son: "+str(fsm1.tabla_nombres_salidas),(20,
405),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las entradas son: "+str(fsm1.tabla_nombres_entradas),(20,
420),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

cv2.putText(canvas2,"La tabla de transiciones indica con cual entrada pasa de un
estado a otro estado. ",(20, 445),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Si en la entrada hay bits en X, significa que no importa el valor
de ese bit para pasar al siguiente estado. ",(20,
460),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de entrada no incluidas en la tabla indican
que el estado no cambia. ",(20, 475),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)

#print("str_transiciones: "+str(str_transiciones))
#print("estados_transiciones: "+str(estados_transiciones))

filename = "g/01_Tabla_estados_binarios" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Grafica 2 : Diagrama de estados -----
-----
```

Opcion 1. Diagrama de estados con funciones desarrolladas en calculadora_datos_graficas.py, sin utilizar la librería graphviz

```

# --- Generacion de un canvas de w = 480 x h = 640 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

# Se coloca el titulo.
cv2.putText(canvas2,"Diagrama de estados de la maquina de estados finitos", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)

# Se colocan las coordenadas de los circulos a utilizar, uno por cada estado y otros
parametros.
xcoords = [int(640/8),int(640*7/8),320,320]
ycoords = [90,90,220,380]
length_lineas = 10
c_radio = [40,58]

for i in range(0,fsm1.num_estados):
    # Se dibuja el circulo del estado actual.
    #print("estado actual: "+str(i))
    cv2.circle(canvas2, (xcoords[i],ycoords[i]), c_radio[0], (0,0,0), 2)
    #cv2.circle(canvas2, (xcoords[i],ycoords[i]), c_radio[1], (0,0,0), 2)

    # Se obtienen los textos del estado actual y de la salida.
    estado = calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
    salida = calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida)

    #cv2.line(canvas2,(xcoords[i],ycoords[i]-50),(xcoords[i],ycoords[i]+50),(0,0,0), 3)

    # Se colocan los textos del estado actual y de la salida
    cv2.putText(canvas2,"q"+str(i)+":"+str(estado),(xcoords[i]-25,ycoords[i]-15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,"Salida: ",(xcoords[i]-26,ycoords[i]+7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(canvas2,str(salida),(xcoords[i]-14,ycoords[i]+22),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    # Se escribe un texto adicional para indicar si es el estado inicial.
    if i == fsm1.estado_inicial:
        cv2.putText(canvas2,"Estado inicial",(xcoords[i]-60,ycoords[i]+c_radio[1]),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

```

```

# Se itera sobre las transiciones de este estado actual hacia otros.
for j in range(0,len(estados_transiciones[i])):
    #print("Estado a actualizar: "+str(j))

    coords_labels = 0

    # Se calculan las coordenadas de las flechas para indicar los cambios de estado.
    lineas_transiciones =
    calc.obtener_puntos_lineas_transicion(canvas2,xcoords,ycoords,c_radio,length_lineas,i,e
    stados_transiciones[i][j])
    #print("lineas_transiciones : "+str(lineas_transiciones))

    # Se calcula la distancia de las flechas con las coordenadas anteriores.
    d = calc.distancia_entre_puntos(lineas_transiciones[0],lineas_transiciones[2])

    # Se calcula el ajuste para la cabeza de la flecha (para que todas las flechas
    tengan el mismo tamano de flecha).
    tip_l = 10 / d
    #print(d)

    # Puesto que se calcularon coordenadas para dos flechas (una de ida y otra de
    vuelta), se elige alguna de las dos.
    # Si el estado actual es menor que el estado al que cambia, se utiliza la primera
    flecha, de lo contrario, se utiliza la segunda.
    # Se dibuja la flecha.
    if i > estados_transiciones[i][j]:
        canvas2 =
        cv2.arrowedLine(canvas2,tuple(lineas_transiciones[0]),tuple(lineas_transiciones[2]),(0,0,0)
        , 3, tipLength = tip_l)
        coords_labels = 0
    else:
        canvas2 =
        cv2.arrowedLine(canvas2,tuple(lineas_transiciones[1]),tuple(lineas_transiciones[3]),(0,0,0)
        , 3, tipLength = tip_l)
        coords_labels = 1

    # Se coloca el texto correspondiente al numero binario con el que cambia de
    estado.

    cv2.putText(canvas2,str_transiciones[i][j],calc.obtener_coordenadas_labels(lineas_transici
    ones[coords_labels],lineas_transiciones[coords_labels+2]),cv2.FONT_HERSHEY_SIMPL
    EX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

    # Se escriben textos de notas adicionales.

```

```

cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio
de estado, indicado por el numero adyacente." ,(20,
445),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de transiciones entre estados que no
aparecen en el diagrama indica que no hay cambio" ,(20,
460),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)
cv2.putText(canvas2,"de estado cuando ocurren.Si hay 'X' en el numero de una flecha
significa que no importa el valor de ese bit." ,(20,
475),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/02_Diagrama_estados_binarios" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Grafica 2 : Diagrama de estados -----
# Opcion 2. Diagrama de estados con funciones desarrolladas en
calculadora_datos_graficas.py, utilizando la librería graphviz

# Nombre del diagrama de estados
g = gv.Digraph('g/diagrama_estados',format='png')

# Configuracion del nodo inicial
g.node('ini', shape="point")

# Configuracion del diagrama de estados, que vaya de izquierda a derecha
g.graph_attr['rankdir'] = 'LR'

str_completo = ["" for i in range(0,fsm1.num_estados)]

for i in range(0,fsm1.num_estados):

    # Calculo de las variables estado y salida en binario
    estado = calc.decimal_a_binario(i,math.ceil(math.log(fsm1.num_estados,2)))
    salida = calc.decimal_a_binario(fsm1.tabla_output_por_estado[i],fsm1.bits_salida)

    # Concatenacion del string completo para cada estado
    str_completo[i] = 'Estado
'+str(estado)+'\n'+str(fsm1.tabla_nombres_estados[i])+'\n'+ "Salida "+str(salida)

    # Se asignan los nombres de los estados.
    # El estado final tiene forma de doble circulo, mientras que se indica con una flecha
    # con punto para el estado inicial
    if i == fsm1.estado_final:

```

```

        g.node(str_completo[i], shape="doublecircle")
    else:
        g.node(str_completo[i])
    if i == fsm1.estado_inicial:
        g.edge('ini',str_completo[i])
    # Se asignan las flechas de transicion entre estados
    for i in range(0,fsm1.num_estados):
        for j in range(0,len(estados_transiciones[i])):
            #print('flecha      de      A:' +str(fsm1.tabla_nombres_estados[i])+'
B:' +str(fsm1.tabla_nombres_estados[estados_transiciones[i][j]]))

    g.edge(str_completo[i],str_completo[estados_transiciones[i][j]],str(str_transiciones[i][j]))

    # Genera el diagrama, pero no lo muestra en el visor de fotos
    g.render(view=False)

    # Se lee la imagen del diagrama de estados generado
    diagrama = cv2.imread('g/diagrama_estados.gv.png')
    height, width, channels = image.shape
    scale_percent = 0
    dim = 0

    # Se recalculan las dimensiones de la imagen.
    # Se pregunta si se recalculan los limites para que ocupe todo el largo (640) o todo el
    alto (320) establecidos
    # Se calcula el porcentaje de escalamiento de la imagen
    if(diagrama.shape[1]*320/diagrama.shape[0] <= 640):
        scale_percent = 320/diagrama.shape[0]
        dim = (int(diagrama.shape[1]*scale_percent), 320)
    else:
        scale_percent = 640/diagrama.shape[1]
        dim = (640, int(diagrama.shape[0]*scale_percent))

    # Se asignan las nuevas dimensiones de la imagen, de acuerdo al porcentaje de
    estaca calculado anteriormente
    diagrama_resized = cv2.resize(diagrama, dim, interpolation = cv2.INTER_AREA)
    #print('scale percent: ',scale_percent)
    #print('diagrama_resized Dimensions : ',diagrama_resized.shape)

    # Se guarda la imagen escalada
    filename = "g/diagrama_estados_resized" + ".jpg"
    cv2.imwrite(filename,diagrama_resized)

```

```

#cv2.imshow("Resized image", resized)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

# En caso de requerirlo, se concatenan partes en blanco a la imagen del diagrama de
estados a los lados
if(diagrama_resized.shape[1] < 640):
    canvas2 = np.ones((diagrama_resized.shape[0],int((640-diagrama_resized.shape[1])/2),3), dtype = "uint8")*255
    filename = "g/blanco1" + ".jpg"
    cv2.imwrite(filename,canvas2)

# Se verifica si se necesitan dos imagenes de igual tamaño, esto es si las
dimenciones de la imagen del diagrama de estados es par
# De lo contrario se crean dos imagenes en blanco de diferente tamano
if diagrama_resized.shape[1]%2 == 1:
    canvasaux = np.ones((diagrama_resized.shape[0],int((640-diagrama_resized.shape[1])/2)+1,3), dtype = "uint8")*255
    #print("blanco2 = blanco1 + 1")
    filename = "g/blanco2" + ".jpg"
    cv2.imwrite(filename,canvasaux)
else:
    #print("blanco2 = blanco1")
    filename = "g/blanco2" + ".jpg"
    cv2.imwrite(filename,canvas2)

img_blanco1 = cv2.imread('g/blanco1.jpg')
img_blanco2 = cv2.imread('g/blanco2.jpg')
#print('blanco1 Dimensions : ',img_blanco1.shape)
#print('blanco2 Dimensions : ',img_blanco2.shape)

im_h = cv2.hconcat([img_blanco1,diagrama_resized,img_blanco2])
#print('im_h Dimensions : ',im_h.shape)
else:
    im_h = diagrama_resized.copy()
    #print('im_h Dimensions : ',im_h.shape)

#cv2.imshow("Resized image", im_h)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

# Se crea la imagen del titulo del diagrama de estados
canvas2 = np.ones((60,640,3), dtype = "uint8")*255
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

```

```

cv2.putText(canvas2,"Diagrama de estados de la maquina de estados finitos", (30,
35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1, cv2.LINE_AA)
filename = "g/titulo" + ".jpg"
cv2.imwrite(filename,canvas2)
img_titulo = cv2.imread('g/titulo.jpg')

# Se crea la imagen que contiene las anotaciones del diagrama de estados
canvas2 = np.ones((420-diagrama_resized.shape[0],640,3), dtype = "uint8")*255
cv2.putText(canvas2,"Los estados son: "+str(fsm1.tabla_nombres_estados),(20, 330-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las salidas son: "+str(fsm1.tabla_nombres_salidas),(20, 345-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las entradas son: "+str(fsm1.tabla_nombres_entradas),(20,
360-diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)

cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio
de estado, indicado por el numero adyacente.",(20, 385-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"Las combinaciones de transiciones entre estados que no
aparecen en el diagrama indica que no hay cambio", (20, 400-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)
cv2.putText(canvas2,"de estado cuando ocurren.Si hay 'X' en el numero de una flecha
significa que no importa el valor de ese bit.", (20, 415-
diagrama_resized.shape[0]),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1,
cv2.LINE_AA)

filename = "g/anotaciones" + ".jpg"
cv2.imwrite(filename,canvas2)

# Se leen las imagenes de las anotaciones y del titulo
img_anotaciones = cv2.imread('g/anotaciones.jpg')
img_titulo = cv2.imread('g/titulo.jpg')

# Se concatenan las imagenes del diagrama de estados, titulo y anotaciones
im_v = cv2.vconcat([img_titulo,im_h,img_anotaciones])

# Se guarda la imagen final
filename = "g/03_Diagrama_estados_binarios_alternativo" + ".jpg"
cv2.imwrite(filename,im_v)
graficas.append(filename)

```

```

# Se obtiene la ruta desde donde se ejecuta el archivo main.py
path = os.path.abspath(os.getcwd())
#print(path)

# Se eliminan las imagenes que ya no se van a usar, utilizando la ruta calculada
# anteriormente
for i in
['g/diagrama_estados.gv','g/diagrama_estados.gv.png','g/diagrama_estados_resized.jpg','
g/blanco1.jpg','g/blanco2.jpg','g/titulo.jpg','g/anotaciones.jpg']:
    try:
        os.remove(path+'/'+i)
    except:
        #print("Archivo "+i+" no removido")
        pass

#cv2.imshow('g/03_Diagrama_estados_binarios_alternativo.jpg', im_v)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

# --- Grafica 3 : Voltaje y binario -----
# --- Generacion de un canvas de w = 480 x h = 640 ---
canvas2 = np.ones((480,640,3), dtype = "uint8")*255

# --- RECTANGULO ---
cv2.rectangle(canvas2, (20,20),(620,40),(0,0,0), -1, cv2.LINE_AA)

# --- Estados binarios ---

# Se escribe el titulo.
cv2.putText(canvas2,"Transiciones de estados de acuerdo al vector de transiciones de
entrada utilizado", (30, 35),cv2.FONT_HERSHEY_SIMPLEX,0.5, (255, 255, 255), 1,
cv2.LINE_AA)
#cv2.circle(canvas2, (320,260), 215,(255,0,0),2)
#cv2.circle(canvas2, (320,260), 185,(255,0,0),2)

angle = np.linspace(0,2*math.pi,int(len(muestras[muestras_llaves[0]])/2+1))
#print("angle: "+str(angle))
#print("muestras_llaves[0]: "+str(muestras_llaves[0])+
'+str(muestras[muestras_llaves[0]]))')
#print("muestras_llaves[1]: "+str(muestras_llaves[1])+
'+str(muestras[muestras_llaves[1]]))')

```

```

    #print("muestras_llaves[2]: "+str(muestras[muestras_llaves[2]]))

    # Dibuja cada uno de los circulos y la informacion del estado correspondiente que va dentro de ella. Tambien dibuja las flechas entre circulos.

    for i in range(0,int(len(muestras[muestras_llaves[0]])/2)):

        # Se obtienen las coordenadas del circulo
        xcoor = int(320+185*math.cos(angle[i]))
        ycoor = int(260+185*math.sin(angle[i]))

        # Se dibuja el circulo.
        cv2.circle(canvas2, (xcoor,ycoor), 35, (0,0,0), 2)

        # Se obtienen los textos del estado actual y de la salida a partir del vector de muestras leido.

        str_bin_estado = calc.decimal_a_binario(int(muestras[muestras_llaves[1]][i]),math.ceil(math.log(fsm1.num_estados,2)))
        str_bin_salida = calc.decimal_a_binario(int(muestras[muestras_llaves[2]][i]),fsm1.bits_salida)

        # Se escriben los textos del estado actual y de la salida.

        cv2.putText(canvas2,"q"+str(int(muestras[muestras_llaves[1]][i]))+":"+str_bin_estado,(xcoor-24,ycoor-15),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
        cv2.putText(canvas2,"Salida:",',(xcoor-26,ycoor+7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)
        cv2.putText(canvas2,str_bin_salida,(xcoor-15,ycoor+22),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

        # Encuentra las coordenadas de la flecha entre dos circulos.

        i1 = calc.get_intersections(320,260,180,xcoor,ycoor,40)
        i2 = calc.get_intersections(320,260,180,int(320+185*math.cos(angle[i+1])),int(260+185*math.sin(angle[i+1])),40)
        dist = 100
        flecha_coords = [0]*4
        "for j in range(0,1):
            if math.dist([i1[j*2],i1[j*2+1]],[i2[j*2],i2[j*2+1]]) < dist:
                print('1, j='+str(j))
                dist = math.dist([i1[j*2],i1[j*2+1]],[i2[j*2],i2[j*2+1]])
                flecha_coords[0] = i1[j*2]
                flecha_coords[1] = i1[j*2+1]
                flecha_coords[2] = i2[j*2]

```

```

flecha_coords[3] = i2[j*2+1]
if math.dist([i1[(2-j*2)],i1[(2-j*2)+1]], [i2[j*2],i2[j*2+1]]) < dist:
    print("2, j="+str(j))
    dist = math.dist([i1[j*2],i1[j*2+1]], [i2[j*2],i2[j*2+1]])
    flecha_coords[0] = i1[(2-j*2)]
    flecha_coords[1] = i1[(2-j*2)+1]
    flecha_coords[2] = i2[j*2]
    flecha_coords[3] = i2[j*2+1]"""

# Se puede comprobar con el codigo comentado que esta es la solucion que
proporciona las coordenadas correctas.
flecha_coords[0] = i1[2]
flecha_coords[1] = i1[3]
flecha_coords[2] = i2[0]
flecha_coords[3] = i2[1]

# Se calcula un ajuste a las coordenadas de las etiquetas de las flechas.
fxcoor = int((flecha_coords[0] + flecha_coords[2])/2 + int(-15+32*math.cos(angle[i])))
fycoor = int((flecha_coords[1] + flecha_coords[3])/2 + int(7+18*math.sin(angle[i])))

#print('coordenadas de la flecha: '+str(flecha_coords))
#print('distancia: '+str(dist))

# Dibuja la flecha y escribe el texto correspondiente.
canvas2
cv2.arrowedLine(canvas2,(int(flecha_coords[0]),int(flecha_coords[1])),(int(flecha_coords[2])
),int(flecha_coords[3])),(0,0,0), 3, tipLength = 0.25)
str_bin_entrada_transicion
calc.decimal_a_binario(int(muestras[muestras_llaves[0]][i+1]),fsm1.bits_salida)

cv2.putText(canvas2,str_bin_entrada_transicion,(fxcoor,fycoor),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

#print("str_bin_estado: "+str(str_bin_estado)+str(str_bin_entrada_transicion))
# Si es el primer estado el que se dibuja, se anade el texto que indique que es el
inicio.
if i == 0:
    canvas2
cv2.arrowedLine(canvas2,(xcoor+40+60,ycoor),(xcoor+40,ycoor),(0,0,0), 3, tipLength = 0.25)
    cv2.putText(canvas2,"Inicio",(xcoor+55,ycoor-7),cv2.FONT_HERSHEY_SIMPLEX,0.5, (0, 0, 0), 1, cv2.LINE_AA)

```

```

    cv2.putText(canvas2,"Cada circulo es un estado. Cada flecha representa un cambio
de     estado,     indicado     por     el     numero     adyacente.      "      ,(20,
477),cv2.FONT_HERSHEY_SIMPLEX,0.35, (0, 0, 0), 1, cv2.LINE_AA)

filename = "g/04_Traniciones_fsm" + ".jpg"
cv2.imwrite(filename,canvas2)
graficas.append(filename)

# --- Auxiliar para depuracion ---
plt.show()

@property
def graficas(self):
    return graficas # --- Nombre de la grafica y ruta donde se guarda ---

def imprimir_archivos_graficas(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" --- Archivos de imagenes de graficas *.jpg ---")
    print("*****")
    for g in graficas:
        print("")
        print(g)

def imprimir_muestras_canal(self):

    print(" ")
    print(" En CLASE Graficador::")
    print("*****")
    print(" --- Vectores de muestras por canal ---")
    print("*****")
    # --- ESTRUCTURA del diccionario:
    #   datos = {canal: i, llaveMuestras[i]: vector de datos}
    # -----
    muestras = self.vectores_muestras
    llaves = list(muestras)
    print(" llaves = ",llaves)
    for k in range(len(llaves)):
        print("")
        print(" --- ",llaves[k]," : ",muestras[llaves[k]])

def imprimir_vectores_muestras(self):

```

```
print(" ")
print(" En CLASE Graficador:")
print("*****")
print(" -- Muestras leidas de la Clase Modelo --")
print("*****")
muestras = self.vectores_muestras
llaves = self.titulos_vectores_muestras
print(" llaves = ",llaves)
for k in range(len(llaves)):
    print("")
    print(" --- ",llaves[k],": ",muestras[llaves[k]])
```

Figura 390. Código completo de la clase graficador.py

