

Informe IPRE 2024-1

Carlos Stappung
 Pontificia Universidad Católica de Chile
 Primer semestre, 2024

I. RESUMEN

Esta investigación aborda la obtención de perdidas en transistores MOSFET de carburo de silicio (SiC), para caracterizar un proceso de selección en base a simulaciones y cálculos analíticos. Para esto se realizaron pruebas a las condiciones de operación de pérdidas y Double Pulse Test (DPT) para transistores provenientes de tres fabricantes distintos (Infineon, ROHM y United SiC). Esto se comparó con los resultados de simulaciones con modelos entregados por los fabricantes y cálculos en base a la hoja de datos.

II. INTRODUCCIÓN

En este informe se analizará la precisión de diferentes métodos para estimar perdidas en los transistores MOSFET SiC de tres fabricantes. El objetivo es identificar cuál de los métodos de estimación se condice mejor con los resultados obtenidos experimentalmente, y por tanto, es fiable para utilizar en el diseño de convertidores de potencia. También se realizó una prueba de DPT para comparar los valores entregados por el fabricante, con los obtenidos en la experimentación.

III. DOUBLE PULSE TEST

Para esta prueba se tomó la decisión de realizarla como Full Bridge. Si bien se puede utilizar un Half-Bridge, nuestra aplicación será en Full-Bridge, por lo cual es más realista. Además, podemos usar la misma PCB que se utiliza para las medidas de eficiencia y no implica una mayor dificultad. Nuestro Set Up teórico debería verse como la siguiente imagen.

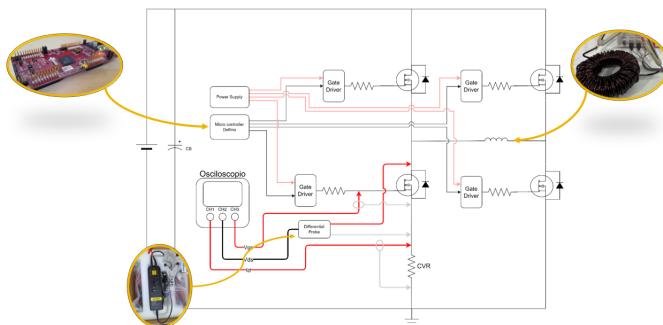


Figura 1: Set Up teórico DPT

Con esta configuración se realizarán dos pulsos en la carga los cuales realizarán un corto en el inductor hasta llegar a corriente nominal.

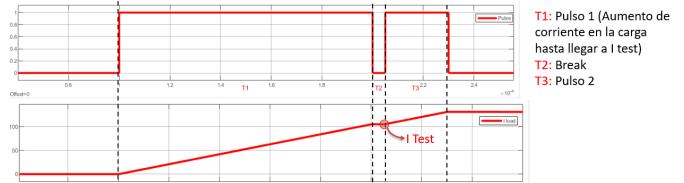


Figura 2: Señales de V_g y corriente en el inductor

La corriente en cada pulso se verá de la siguiente manera.

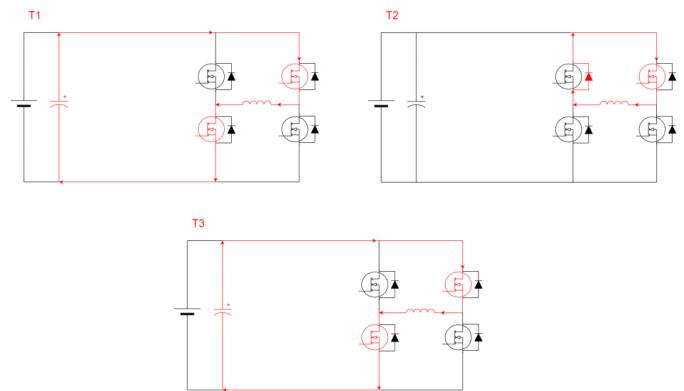


Figura 3: Camino de la corriente por etapa

Calculo de inductancia

También debemos estimar qué inductor se puede seleccionar para la prueba, con un voltaje Dc de 420 [V] y una corriente de prueba de 25 [A]. Con esto:

Condición para la Cota superior L: Primer pulso $\leq 100\mu s$
 (Para evitar sobre calentamientos)

$$L_{load} \leq \tau_1 \frac{V_{DC}}{I_{test}} \Rightarrow L_{load} \leq 1,7[mH] \quad (1)$$

Condición para la Cota Inferior L: I_{test} no caiga más de 5 %.

Además en nuestro caso $V_f = 3.3 [V]$; $I_{test} = 25[A]$; $\Delta I = 1,25 [A]$; $R_s^* = R_{ds} + R_d = 0.07 [\Omega]$; $T2^{**} = 5 [\mu s]$

$$L_{load} \geq \left[R_s \cdot \tau_2 \cdot \left(\frac{1}{\ln \left(\frac{\Delta I}{I_{test} + \frac{V_f}{R_s}} + 1 \right)} \right) \right] \Rightarrow L_{load} \geq 20,37 \mu H \quad (2)$$

* R_s se calcula como la resistencia del circuito generado en T2, para el puente H es la resistencia de conducción de un MOSFET más la resistencia de conducción del BodyDiode

**T2 se escoge tal que sea tiempo suficiente hasta que el switch termine su proceso de cerrado Nuestros MOSFET tiene un máximo de $t_f = 26$ ns, por lo que $5\mu s \gg 26$ ns

En nuestro caso, tenemos una carga de $464.6 [\mu H]$ Así que estamos en el rango.

Calculo de tiempos

T1 se debe calcular en función del I test que buscamos ($25 [A]$ en nuestro caso).

$$\tau_1 = L_{load} \cdot \frac{I_{test}}{V_{DC}} \Rightarrow \tau_1 = 27,65 [\mu s] \quad (3)$$

$T2 = 5 [\mu s]$ (como definimos antes).

$T3 = 5 [\mu s]$ (pequeño para no sobre cargar el transistor y darle el tiempo mínimo para que encienda).

$$\tau_3 \ll L_{load} \cdot \frac{I_{max}}{V_{DC}} - \tau_1 \quad (4)$$

La corriente máxima más pequeña de nuestros transistores es $98[A]$ y t_r más grande de $75 [\mu s]$.

$$\tau_3 \ll 80,752[\mu s] \quad 75[\mu s] \ll \tau_3 \quad (5)$$

En nuestro caso se estimo conveniente $T3 = 5 [\mu s]$

Calculo capacitancia

Además, debemos agregar un banco capacitivo para evitar caídas en el voltaje. Cota Inferior C_b : V_{DC} no caiga más de 1%

$$C_b \geq \frac{L_{load} I_{test}^2}{2 \cdot V_{DC} \cdot \Delta V_{DC} - \Delta V_{DC}^2} \Rightarrow C_b \geq 299,1 [\mu F] \quad (6)$$

Parámetros a medir con la prueba

Ahora debemos tener claros que es lo que podemos medir y que nos interesa. Parámetros de encendido.

- $t_{d(on)}$: Turn-on delay.
- **t_r : Rise time.**
- t_{on} : Turn-on time.
- **E_{on} : On Energy, dv/dt, di/dt.**

Parámetros de apagado.

- $t_{d(off)}$: Turn-off delay.
- **t_f : Fall time.**
- t_{off} : Turn-off time.
- **E_{off} : Off Energy, dv/dt, di/dt.**

Parámetros de Reverse-Recovery.

- t_{rr} : Reverse Recovery time.
- I_{rr} : Reverse Recovery current.
- Q_{rr} : Reverse Recovery charge.
- E_{rr} : Reverse Recovery energy, di/dt.
- V_{sd} : forward on voltage.

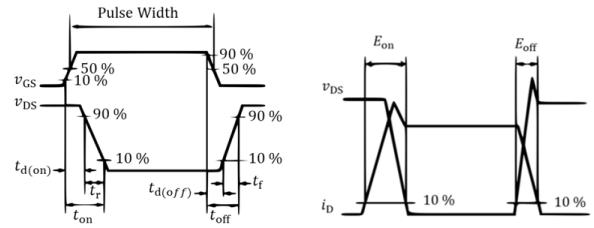


Figura 4: [1]

Con esto determinaremos las pérdidas de encendido y apagado. Mediaremos t_r y t_f con el objetivo de ajustar nuestros cálculos teóricos y compararemos resultados experimentales con las especificaciones dadas por el fabricante.

Simulaciones

Para verificar el funcionamiento del experimento se utilizo Simulink de Matlab para posteriormente usarla en la programación del Microcontrolador Delfino. Sin embargo, finalmente se uso Code Composer por razones que se explicaran más adelante.

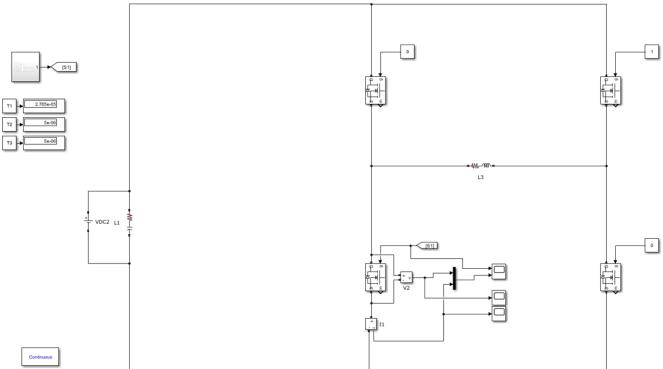


Figura 5: Esquemático en Simulink DPT

Con los siguientes resultados para el voltaje V_{ds} e I_d .

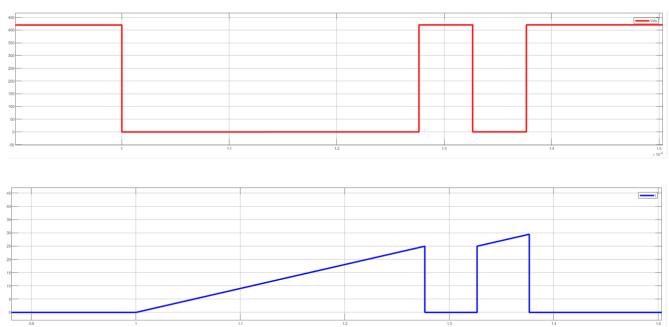


Figura 6: Señales Simulink DPT

Flujo de señales Delfino

Para el flujo de señales debemos que fijarnos que el Gate superior izquierdo no switchea mientras que el inferior si, por lo que no podemos utilizar la clásica lógica negada por pierna. Esto es algo a tener en cuenta, por lo que se reprogramo la tarjeta CPLD. Y se genero el siguiente diagrama de flujos para las señales enviadas por la Delfino.

Flujo de Variable PWM1A

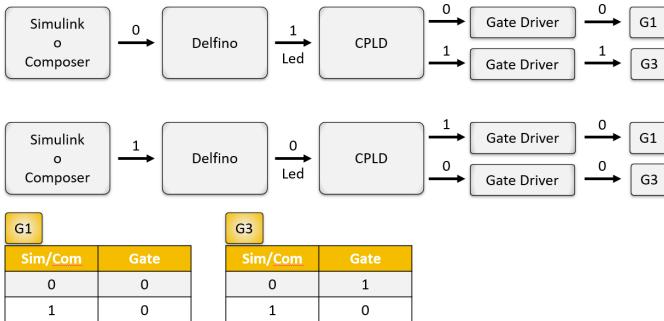


Figura 7: Flujo de Variable PWM1A

Flujo de Variable PWM1B

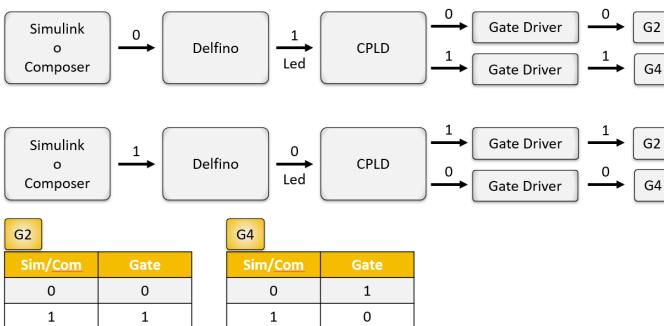


Figura 8: Flujo de Variable PWM1B

Implementación en Delfino

Como se comentó antes, el problema que se tuvo en la implementación fue que Simulink solo puede conseguir pulsos de 5[us] a lo menos. Lo cual es un problema, dado que debemos aproxima 27 [us] a 25 o 30. Para solucionar esto se prefirió utilizar Code Composer, dado que permite intervalos < 1 [us]. Con esto se realizaron las pruebas respectivas en laboratorio para verificar que esté realizando pulsos.

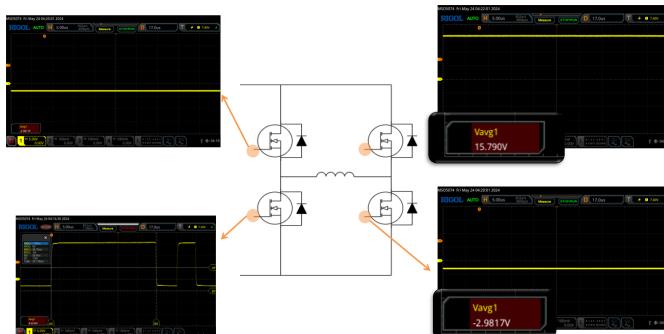


Figura 9: Pruebas de voltajes Vgs en laboratorio

También se verifico el ancho de estos pulsos.



Figura 10: Verificación Ancho de pulsos

Calentador

Para realizar esta prueba debemos llevar a la temperatura deseada el transistor, por lo que se realizó un programa básico con Arduino para que un Heatear lleve el transistor a temperatura nominal.

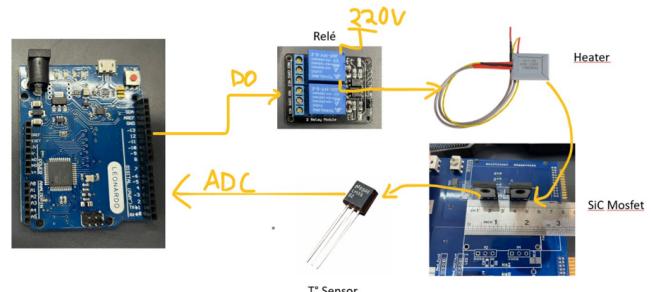


Figura 11: Esquemático Circuito Heater

Set up Experimental

Con todo esto llegamos al siguiente set up experimental para la prueba de DPT (Utilizando una fuente NHR de 400V).

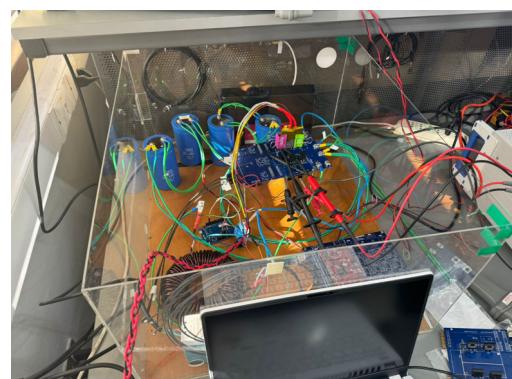


Figura 12: Vista general Set Up DPT

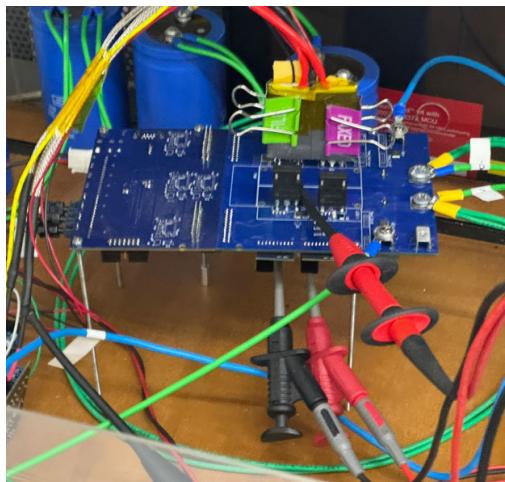


Figura 13: Zoom a Set Up Experimental

Script para resultados de DPT

Se programo un script de Python que recibe tres inputs y nos entrega todos los resultados de la prueba de DPT. Es capaz de detectar los valores máximos y mínimos de cada señal. **IMPORTANTE:** Se asume VGS entre -3.3 y 15.4, ya VGS presenta un comportamiento que se explicara más adelante.

Series necesarias:

- Ids-csv.csv: Serie de corriente Id
- Vgs-csv.csv: Serie de Voltaje Vgs
- Vds-csv.csv: Serie de Voltaje Vds

Estos tres archivos se deben encontrar en la misma carpeta con nombre libre, el cual después ingresa como parámetro al script.

*Dado las oscilaciones de VGS se realizaron ciertas aproximaciones a la señal en el script, estas se pueden ver en el gráfico de salida.

Resultados DPT

Cabe destacar que las pruebas con United SiC fueron descartadas por problemas de micro-cortos que no se pudo solucionar.

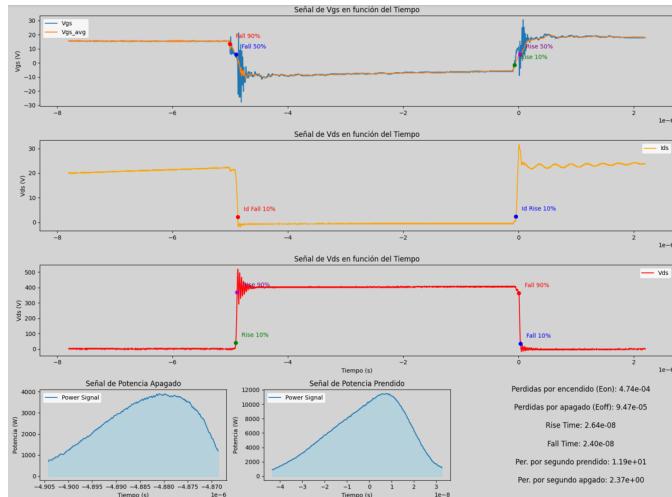


Figura 14: Resultados Prueba DPT Rohm

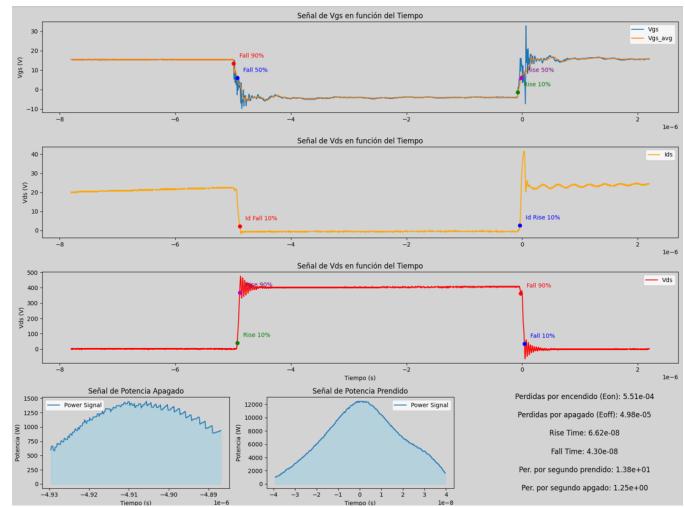


Figura 15: Resultados Prueba DPT Infineon

Los resultados se pueden ver tabulados en la tabla I

| | Infineon | Rohm |
|-------------------------------|-----------------|----------------|
| Pérdidas por apagado | 48,91 μJ | 93,93 μJ |
| Pérdidas por encendido | 551,31 μJ | 473,47 μJ |
| tr | 37,92 ns | 23,52 ns |
| tf | 41,28 ns | 23,04 ns |

Tabla I: Comparación de resultados entre Infineon y Rohm

Dentro de estas pruebas hay dos cosas importantes de recalcar. Primero tenemos que entender a que se debe el peak de corriente al momento de encender el transistor. Como se puede ver en la figura 16, existe un peak de corriente la cual corresponde a la REverse Recovery Current del bode diodo del transistor superior, la cual pasa por el transistor inferior al dejar de conducir.

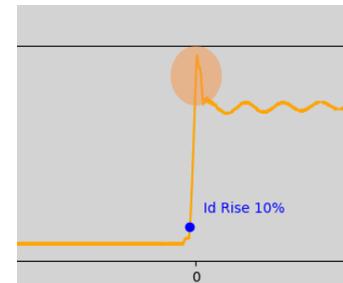


Figura 16: Reverse Recovery Current Rohm

En segundo lugar, podemos visualizar que en la prueba de Rohm existe un voltaje muy negativo para Vgs (Voltaje por debajo de los -3.3V, casi en -10V) y disminuye lentamente, el cual se puede observar en la figura 17.

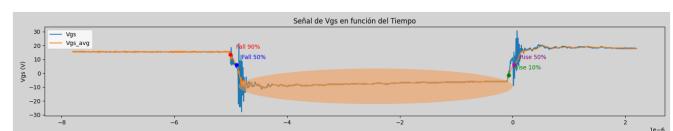


Figura 17: Señal de Vgs para prueba DPT Rohm

Durante el desarrollo de la investigación, quedo pendiente el análisis de este fenómeno. Sin embargo, se tienen las

siguiente hipótesis.

1.- Mucho Under Shoot.

2.- Corriente entrante al gate driver.

Camparación resultados con DataSheet

Nuestro objetivo no es comprar los resultados con los DataSheets. Sin embargo, nos parece importante mostrar que existen grandes diferencias por las condiciones a las cuales se realizan las pruebas. Algo que se debe tener en cuenta si se utilizan estos datos para calculos teoricos.

IV. EFICIENCIA Y PÉRDIDAS DE DISTINTOS FABRICANTES

se realizaron varios métodos para estimar las perdidas de los distintos fabricantes. Todos estos bajo las siguientes condiciones:

$$F = 50[\text{Hz}] \quad V_{dc} = 400[\text{V}] \quad L = 460[\mu\text{H}]$$

$$ma = 0,89 \quad P = 5[\text{kW}a] \quad fc = 25[\text{kHz}] \quad R = 12,2[\Omega]$$

Los métodos de comparación, con el objetivo de poder discernir en cual de esto nos aporta mayor fidelidad con las pruebas en laboratorio, fueron los siguiente:

- Método 1: Cálculos Analíticos
- Método 2: Simulaciones con los modelos de los fabricantes
- Método 3: Simulaciones ingresando a mano los parámetros del DataSheet

Método 1: Cálculos Analíticos

Para los cálculos Analíticos se utilizó la siguiente formula para obtener la corriente RMS que cruza por los transistores, la que luego se utilza en los cálculos.

$$I_{RMS} = \sqrt{\sum_{n=1}^N [I_{RMS-n}^2 + I_{DC}^2]} \quad (7)$$

Sacando la amplitud de cada armónico con ayuda de Plecs. Se obtuvo una corriente de 13.19 Irms.

Para el calculo de las perdidas se utilizaron las siguientes ecuaciones.

$$P_{con} = I_{RMS}^2 R_{ds(on)} \quad (8)$$

$$\begin{aligned} P_{switching} &= P_{COSS} + P_{Qg} + P_{on} + P_{off} \\ &= \frac{COSS V_{peak}^2 f_{sw}}{2} + V_{gs} Q_g f_{sw} \\ &\quad + \frac{1}{2} (t_r V_{on} I_{on} + t_f V_{off} I_{off}) f_{sw} \end{aligned} \quad (9)$$

Además, se considero la aproximación de la corriente I_{on} y I_{off} a la corriente RMS. Con lo que se obtuvo los resultados visibles en la tabla II

| Fabricante | $P_{on}[\text{W}]$ | $P_{off}[\text{W}]$ | $P_{cond}[\text{W}]$ | $P_{swi}[\text{W}]$ | $P_{tot}[\text{W}]$ |
|------------|--------------------|---------------------|----------------------|---------------------|---------------------|
| Infineon | 1.563 | 1.177 | 3.513 | 3.103 | 6.617 |
| ROHM | 4.069 | 1.499 | 2.404 | 6.036 | 8.440 |
| United SiC | 5.354 | 1.856 | 2.958 | 7.712 | 10.671 |

Tabla II: Pérdidas de energía de diferentes fabricantes

Además, se estimaron las perdidas del diodo que en general fueron lo suficientemente bajas como para despreciarlas.

$$P_{VF} = V_f I_D \quad (10)$$

$$P_{Qrr} = Q_{rr} V_D f_S \quad (11)$$

| Fabricante | $P_{Qrr} [\text{W}]$ | $P_{VF} [\text{W}]$ | $P_{total} [\text{W}]$ |
|------------|----------------------|---------------------|------------------------|
| Infineon | 0.0323 | 0.0008 | 0.0331 |
| ROHM | 0.0347 | 0.0007 | 0.0353 |
| United SiC | 0.0222 | 0.0003 | 0.0225 |

Tabla III: Pérdidas Diodo diferentes fabricantes

Juntando ambas perdidas, tenemos los siguientes resultados.

| Fabricante | $P_{diodo} [\text{W}]$ | $P_{mosfet} [\text{W}]$ | $P_{total} [\text{W}]$ |
|------------|------------------------|-------------------------|------------------------|
| Infineon | 0.0331 | 6.6170 | 6.6501 |
| ROHM | 0.0353 | 8.4406 | 8.4760 |
| United SiC | 0.0225 | 10.6713 | 10.6939 |

Tabla IV: Pérdidas totales por fabricante

Método 2: Simulaciones con los modelos entregados por los fabricantes

Simulaciones con los modelos de plecs entregados por el fabricante, estos son modelos térmicos y permiten el uso del bloque Switch Loss Calculator incorporado a Plecs.

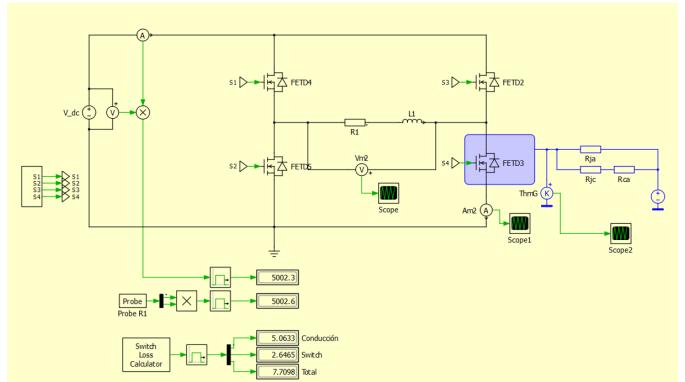


Figura 18: Esquemático Plecs de simulaciones con modelos del fabricante

Método 3: Simulaciones ingresando a mano los parámetros del DataSheet

Simulaciones traspasando a mano los parámetros del DataSheet al modelo de Mosfet entregado por Plecs. Para calcular las perdidas consideramos la diferencia entre la entrada y la salida de potencia y la dividimos en la cantidad de transistores.

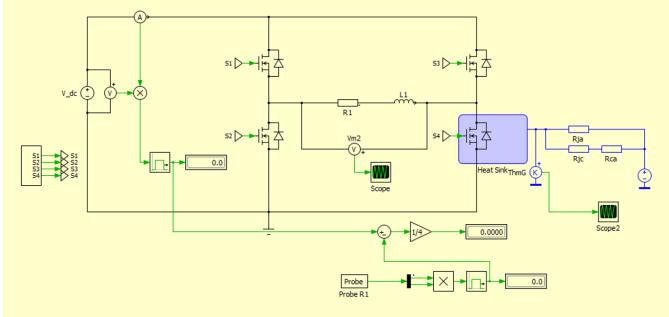


Figura 19: Esquemático Plecs de simulaciones transando el Datasheet

Resultados de cada Método

| Método/Pérdida | Infineon [W] | ROHM [W] | United SiC [W] |
|--------------------|--------------|----------|----------------|
| Analítico | 6.650 | 8.476 | 10.694 |
| Modelos Fabricante | 7.814 | 6.748 | 17.948 |
| Plecs (DataSheet) | 16.460 | 13.150 | 8.311 |

Tabla V: Comparación de métodos de cálculo de pérdidas

Set up Experimental

Con esto se procedió a realizar los experimentos en laboratorio. Lo primero fue programar la CPLS para tener un dead time de 1 [us]. Además, por complicaciones en la implementación, se agregó un código para detectar fallas por ready y se documentó.

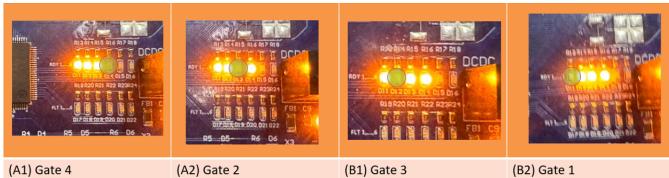


Figura 20: Luces de falla por Ready

El set up experimental quedo de la siguiente manera.

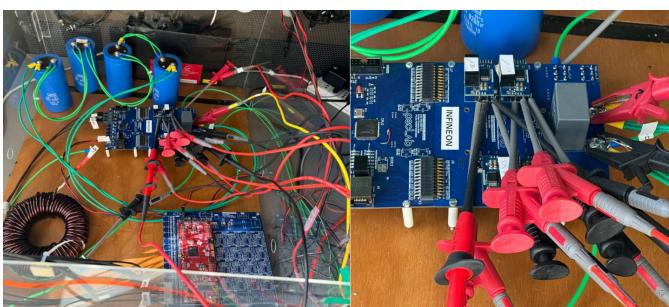


Figura 21: Set Up Experimental prueba de pérdidas

Resultados

En las pruebas de laboratorio, se consiguieron los resultados visibles en la tabla VI



Figura 22: Capturas de Analizador de potencia, izquierda Rohm, derecha Infineon

Promedio de eficiencias:

- Rohm: 98,56 %
- Infineon: 98,26 %

| Método/Pérdida | Infineon [W] | ROHM [W] | United SiC [W] |
|-----------------------------|--------------|----------|----------------|
| Analítico | 6.65 | 8.476 | 10.694 |
| Analítico Datos DPT* | 11.704 | 6.505 | - |
| Datos de Eon y Eoff (DPT)** | 18.533 | 16.621 | - |
| Modelos Fabricante | 7.814 | 6.748 | 17.948 |
| Plecs (Datos DataSheet) | 16.46 | 13.15 | 8.311 |
| Experimental *** | 21.75 | 18 | - |

Tabla VI: Comparación de métodos de cálculo de pérdidas

* El tr y tf obtenidos del DPT usados en los cálculos analíticos.

** Usando directamente Eon y Eoff obtenidos del DPT por la cantidad de switcheos

*** Multiplicando $1 - \eta$ por la potencia (5kW) y dividido en la cantidad de transistores.

Visualización resultados

Para una mejor comparación, se graficaron los resultados a modo de comparación y además se normalizaron.

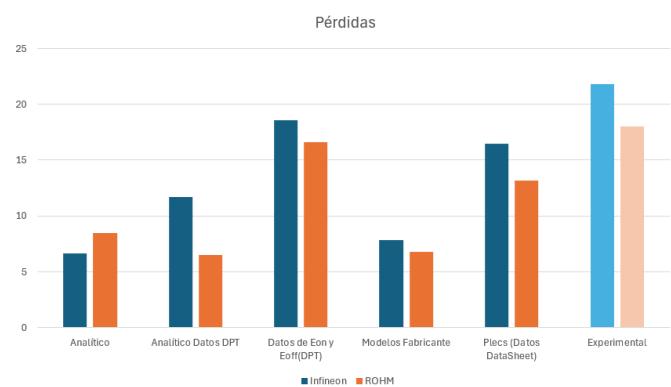


Figura 23: Gráfico comparativo perdidas por método

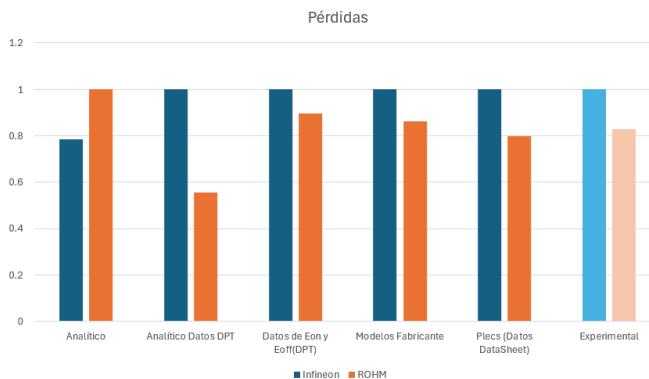


Figura 24: Gráfico comparativo perdidas por método normalizado

REFERENCIAS

- [1] R. . S. G. . C. Kg, “Tips Tricks on Double-Pulse Testing.” [Online]. Available: https://www.rohde-schwarz.com/se/applications/tips-tricks-on-double-pulse-testing-application-note_56280-1049218.html

V. CONCLUSIONES

En esta investigación se realizaron diversas pruebas para estimar las perdidas de cada uno de los transistores MOSFET SiC, para luego validar experimentalmente cual o cuales se ajustaban mejor a la realidad. Los resultados mostraron que las estimaciones por simulaciones en plecs fueron bastante certeras, mientras que la estimación con cálculos analíticos no lo fue. Esto ultimo, se debe principalmente a que se tomaron los valores entregados por el fabricante en el datasheet sin considerar bajo que condiciones realizaron las pruebas, lo que resulto en una mala estimación. Esto se comprobó al volver a realizar los cálculos con los datos obtenidos con la prueba de DPT, que se ajustaron de mejor manera.

Finalmente se logro validar, tanto por simulaciones, como experimentalmente, que el transistor Rohm tenía las menos perdidas en comparación a los otros fabricantes. Sin embargo, aún nos quedan cosas por trabajar. Falta realizar una tercera prueba a un nuevo transistor para validar que los resultados son confiables y no casualidad. Además, debemos entender a fondo que es lo que esta sucediendo en la prueba de DPT para el transistor Rohm con su señal V_{gs} .

Por ultimo, queda agradecer al Profesor Javier, a Francisco y a Emmanuel por su buena disposición y toda la ayuda brindada durante la investigación.

VI. COSAS POR HACER

1. Realizar una tercera prueba para contrastar resultados.(Selección de un transistor SiC)
2. Entender el fenómeno que esta ocurriendo con los transistores Rohm en V_{gs} (Comparar con otras series de tiempo disponibles)
3. Ver posible existencia de delay que desfaza las señales Id y V_{ds} en la prueba de DPT.
4. Analizar porque el gráfico de potencia de apagado de Infineon no se ve suave.
5. Entender a profundidad que ocasiona los fallos por la señal Ready de los Gate Drivers

ANEXOS

Código Code Composer DPT: .

```

1 #include <device.h>
2 #include <driverlib.h>
3
4
5
6 void main(void)
7 {
8     Device_init();
9     Device_initGPIO();
10
11     GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
12     GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);
13
14     // Config Gate 3 Senal Invertida con GPIO
15     GPIO_setPadConfig(2, GPIO_PIN_TYPE_STD);
16     GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);
17
18     // Config Gate 2 y 4
19     GPIO_setPadConfig(3, GPIO_PIN_TYPE_STD);
20     GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);
21
22     // Config Enable
23     GPIO_setPadConfig(0, GPIO_PIN_TYPE_STD);
24     GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);
25
26     Interrupt_initModule();
27     Interrupt_initVectorTable();
28
29     EINT;
30     ERTM;
31
32     // Config Pierna izquierda inicio
33     GPIO_writePin(2, 1);
34
35     // Config pierna derecha
36     GPIO_writePin(3, 1);
37
38     // Config Enable
39     GPIO_writePin(0, 1);
40
41     for(;;)
42     {
43         DELAY_US(500000);
44
45         // Config Enable
46         GPIO_writePin(0, 0);
47
48         DELAY_US(1000);
49
50         // Comenzamos T1
51         GPIO_writePin(2, 0);
52
53         // Esperamos exactamente T1. (Para calcular que valor poner a delay usar la siguiente formula)
54         // Delay = (T1-0.55)/2 todo en microsegundos
55         DELAY_US(13.225);
56
57         // Comenzamos T2
58         GPIO_writePin(2, 1);
59
60         // Delay = (T2-0.55)/2 todo en microsegundos
61         DELAY_US(2.225);
62
63         // Comenzamos T3
64         GPIO_writePin(2, 0);
65
66         // Delay = (T3-0.55)/2 todo en microsegundos
67         DELAY_US(2.225);
68
69         // Esperamos a la siguiente prueba
70         GPIO_writePin(2, 1);
71
72         DELAY_US(1000);
73

```

```

74     GPIO_writePin(0, 1);
75     DELAY_US(1000000);
76 }
77 }
```

Código Arduino Heater: .

```

1 float tempC; // Variable para almacenar el valor obtenido del sensor (0 a 1023)
2 int pinLM35 = 0; // Variable del pin de entrada del sensor (A0)
3 int pin_switch = 3; // Variable del pin de entrada del sensor (A3)
4 float temp_set = 70.0;
5 float ventana = 0.2;
6 int estado = 0;
7
8 void setup() {
9     // Configuramos el puerto serial a 9600 bps
10    Serial.begin(9600);
11    pinMode(pin_switch, OUTPUT);
12    digitalWrite(pin_switch, LOW);
13}
14
15 void loop() {
16     // Con analogRead leemos el sensor, recuerda que es un valor de 0 a 1023
17     tempC = analogRead(pinLM35);
18
19     // Calculamos la temperatura con la formula
20     tempC = (5.0 * tempC * 100.0)/1024.0;
21
22
23     // Esperamos un tiempo para repetir el loop
24     if (estado == 0 & tempC > temp_set + ventana){
25         digitalWrite(pin_switch, HIGH);
26         estado = 1;
27     } else if (estado == 1 & tempC < temp_set - ventana){
28         digitalWrite(pin_switch, LOW);
29         estado = 0;
30     }
31
32     delay(50);
33 }
```

Código Python para análisis de series de tiempo DPT: .

```

1 import csv
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 ## CARACTERISTICAS A SETEAR ##
6
7 CARPETA = "Series_Rohm"
8
9 VGS_MIN = -3.3
10 VGS_MAX = 15.4
11
12 #VDS_MAX = 400 # VDC
13 #VDS_MIN = 0 # Tierra
14
15 VDS_MAX = None
16 VDS_MIN = None
17
18 #I_TEST = 23
19 I_TEST = None
20
21 time_ids = []
22 signal_ids = []
23
24 time_vgs = []
25 signal_vgs = []
26
27 time_vds = []
28 signal_vds = []
29
30 SAMPLE_TIME = None
31 info = {}
32
33 p_Vgs_fall_90_signal = 0
```

```

34 p_Vgs_fall_90_time = 0
35 p_Vgs_fall_90_bool = False
36
37 p_Vgs_fall_50_signal = 0
38 p_Vgs_fall_50_time = 0
39 p_Vgs_fall_50_bool = False
40
41 p_Vgs_rise_10_signal = 0
42 p_Vgs_rise_10_time = 0
43 p_Vgs_rise_10_bool = False
44
45 p_Vgs_rise_50_time = 0
46 p_Vgs_rise_50_signal = 0
47 p_Vgs_rise_50_bool = False
48
49 p_Vsg_Temp_bool = False
50 p_Vgs_start_time_bool = False
51 p_Vgs_end_time_bool = False
52
53 #####
54
55 p_Vds_rise_10_signal = 0
56 p_Vds_rise_10_time = 0
57 p_Vds_rise_10_bool = False
58
59 p_Vds_rise_90_signal = 0
60 p_Vds_rise_90_time = 0
61 p_Vds_rise_90_bool = False
62
63 p_Vds_fall_90_signal = 0
64 p_Vds_fall_90_time = 0
65 p_Vds_fall_90_bool = False
66
67 p_Vds_fall_10_time = 0
68 p_Vds_fall_10_signal = 0
69 p_Vds_fall_10_bool = False
70
71 p_Vdg_Temp_bool = False
72
73
74 #####
75
76
77 p_Id_fall_10_time = 0
78 p_Id_fall_10_signal = 0
79 p_Id_fall_10_bool = False
80
81 p_Id_rise_10_signal = 0
82 p_Id_rise_10_time = 0
83 p_Id_rise_10_bool = False
84
85 p_Id_fall_90_time = 0
86 p_Id_fall_90_signal = 0
87 p_Id_fall_90_bool = False
88
89 p_Id_rise_90_signal = 0
90 p_Id_rise_90_time = 0
91 p_Id_rise_90_bool = False
92
93 p_Id_Temp_bool = False
94
95 Id_bool_start = False
96
97 p_Vgs_start_time = 0
98
99
100 def moving_average(signal, window_size):
101     """
102         Calcula el promedio movil de una senal manteniendo el mismo tamano de la lista.
103
104     Parameters:
105         signal (list): La lista de datos de la senal.
106         window_size (int): El tamano de la ventana para calcular el promedio.
107
108     Returns:
109         list: Una lista con el promedio movil de la senal.
110

```

```

111 """
112 avg_signal = []
113 half_window = 0
114 half_window_max = window_size // 2
115 half_window = half_window_max
116 for i in range(len(signal)):
117     #half_window += 1
118     #if half_window >= half_window_max:
119     #    half_window = half_window_max
120     start = max(0, i - 2*half_window)
121     end = min(len(signal), i + 1)
122     avg = sum(signal[start:end]) / (end - start)
123     avg_signal.append(avg)
124 return avg_signal
125
126 def normalizar_lista(lista):
127 """
128 Normaliza una lista de valores para que esten en el rango [0, 1].
129
130 Parametros:
131 lista (list): Lista de valores numericos a normalizar.
132
133 Retorna:
134 list: Lista normalizada.
135 """
136 min_valor = min(lista)
137 max_valor = max(lista)
138
139 # Asegurarse de que max y min no sean iguales para evitar division por cero
140 if max_valor == min_valor:
141     return [0.5] * len(lista) # Retorna 0.5 para todos si todos los valores son iguales
142
143 return [(valor - min_valor) / (max_valor - min_valor) for valor in lista]
144
145 with open(f'{CARPETA}/id-csv.csv', newline='') as csvfile:
146     spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
147     n = 0
148     for row in spamreader:
149         if n >= 10:
150             time_ids.append(float(row[0]))
151             signal_ids.append(float(row[1]))
152         else:
153             temp = list(row)
154             n += 1
155             try:
156                 info[temp[0]] = temp[1]
157             except:
158                 pass
159
160 ##ACTUALIZAMOS EL TS SI ES QUE NOS LO ENTREGAN
161
162 try:
163     SAMPLE_TIME = float(info["Sample Interval"])
164 except:
165     print("NO SE DETECTO SAMPLE TIME")
166     SAMPLE_TIME = input("INGRESE SAMPLE TIME: ")
167     pass
168
169 print(f"Sample Time = {SAMPLE_TIME}")
170
171 with open(f'{CARPETA}/vgs-csv.csv', newline='') as csvfile:
172     spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
173     n = 0
174     for row in spamreader:
175         if n >= 10:
176             time_vgs.append(float(row[0]))
177             signal_vgs.append(float(row[1]))
178         else:
179             n += 1
180
181 with open(f'{CARPETA}/vds-csv.csv', newline='') as csvfile:
182     spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
183     n = 0
184     for row in spamreader:
185         if n >= 10:
186             time_vds.append(float(row[0]))
187             signal_vds.append(float(row[1]))

```

```

188     else:
189         n += 1
190
191 for time, signal in zip(time_vgs, signal_vgs):
192
193     if signal < 0.95*(VGS_MAX - VGS_MIN) + VGS_MIN and not p_Vgs_start_time_bool:
194         p_Vgs_start_time = time
195         p_Vgs_start_time_bool = True
196
197     if p_Vgs_start_time_bool and signal > 0.04*(VGS_MAX - VGS_MIN) + VGS_MIN and time > p_Vgs_start_time + 1.0e-6 and not p_Vgs_end_time_bool:
198         p_Vgs_end_time = time
199         p_Vgs_end_time_bool = True
200
201
202 AVG_SAMPLE = 5000
203 avg_vds = moving_average(signal_vds, AVG_SAMPLE)
204
205 if VDS_MAX == None:
206     VDS_MAX = max(avg_vds[AVG_SAMPLE:])
207     print(f"Maximo VDS = {VDS_MAX}") # VDC
208
209 if VDS_MIN == None:
210     VDS_MIN = min(avg_vds[AVG_SAMPLE:])
211     print(f"Minimo VDS = {VDS_MIN}") # VDC
212
213 if I_TEST == None:
214     I_TEST = 0
215     for i in signal_ids:
216         if i > I_TEST:
217             I_TEST = i
218         if i < 1:
219             break
220     print(f"I_test encontrado = {I_TEST}")
221
222
223 ### MODIFICAR EN FUNCION DE LA CALIDAD DE LA SENAL
224 PROMEDIADOR_BAJADA = 300
225 PROMEDIADOR_SUBIDA = 600
226
227 avg_time_vgs = time_vgs.copy()
228 avg_signal_vgs = signal_vgs.copy()
229 index_time_vgs = time_vgs.index(p_Vgs_start_time)
230
231
232
233 avg_signal_vgs[index_time_vgs:] = moving_average(signal_vgs[index_time_vgs:], PROMEDIADOR_BAJADA)
234
235 index_time_vgs = time_vgs.index(p_Vgs_end_time)
236 avg_signal_vgs[index_time_vgs:] = moving_average(signal_vgs[index_time_vgs:], PROMEDIADOR_SUBIDA)
237
238
239 for time, signal in zip(time_vgs, avg_signal_vgs):
240
241     if signal < 0.9*(VGS_MAX - VGS_MIN) + VGS_MIN and p_Vgs_fall_90_bool == False:
242         p_Vgs_fall_90_time = time
243         p_Vgs_fall_90_signal = signal
244         p_Vgs_fall_90_bool = True
245
246     if p_Vgs_fall_90_bool and signal < 0.5*(VGS_MAX - VGS_MIN) + VGS_MIN and p_Vgs_fall_50_bool == False:
247         p_Vgs_fall_50_time = time
248         p_Vgs_fall_50_signal = signal
249         p_Vgs_fall_50_bool = True
250
251     if p_Vgs_fall_50_bool and signal < 0.01*(VGS_MAX - VGS_MIN) + VGS_MIN:
252         p_Vsg_Temp_bool = True
253
254     if p_Vsg_Temp_bool and signal > 0.1*(VGS_MAX - VGS_MIN) + VGS_MIN and p_Vgs_rise_10_bool == False:
255         p_Vgs_rise_10_time = time
256         p_Vgs_rise_10_signal = signal
257         p_Vgs_rise_10_bool = True
258
259     if p_Vgs_rise_10_bool and signal > 0.5*(VGS_MAX - VGS_MIN) + VGS_MIN and p_Vgs_rise_50_bool == False:
260         p_Vgs_rise_50_time = time
261         p_Vgs_rise_50_signal = signal
262         p_Vgs_rise_50_bool = True
263

```

```

264
265
266 for time, signal in zip(time_vds, signal_vds):
267     if signal > 0.1*(VDS_MAX - VDS_MIN) + VDS_MIN and p_Vds_rise_10_bool == False:
268         p_Vds_rise_10_time = time
269         p_Vds_rise_10_signal = signal
270         p_Vds_rise_10_bool = True
271
272     if p_Vds_rise_10_bool and signal > 0.9*(VDS_MAX - VDS_MIN) + VDS_MIN and p_Vds_rise_90_bool == False:
273         p_Vds_rise_90_time = time
274         p_Vds_rise_90_signal = signal
275         p_Vds_rise_90_bool = True
276
277
278     if p_Vds_rise_90_bool and signal > 0.95*(VDS_MAX - VDS_MIN) + VDS_MIN and time > p_Vds_rise_90_time + 1.0e-6:
279         p_Vdg_Temp_bool = True
280
281     if p_Vdg_Temp_bool and signal < 0.9*(VDS_MAX - VDS_MIN) + VDS_MIN and p_Vds_fall_90_bool == False:
282         p_Vds_fall_90_time = time
283         p_Vds_fall_90_signal = signal
284         p_Vds_fall_90_bool = True
285
286     if p_Vds_fall_90_bool and signal < 0.1*(VDS_MAX - VDS_MIN) + VDS_MIN and p_Vds_fall_10_bool == False:
287         p_Vds_fall_10_time = time
288         p_Vds_fall_10_signal = signal
289         p_Vds_fall_10_bool = True
290
291
292 ######
293
294 for time, signal in zip(time_ids, signal_ids):
295
296     if signal > 0.95*I_TEST and Id_bool_start == False:
297         Id_bool_start = True
298
299     if Id_bool_start and signal < 0.9*I_TEST and p_Id_fall_90_bool == False:
300         p_Id_fall_90_time = time
301         p_Id_fall_90_signal = signal
302         p_Id_fall_90_bool = True
303
304
305     if p_Id_fall_90_bool and signal < 0.1*I_TEST and p_Id_fall_10_bool == False:
306         p_Id_fall_10_time = time
307         p_Id_fall_10_signal = signal
308         p_Id_fall_10_bool = True
309
310     if p_Id_fall_10_bool and signal < 0.05*I_TEST:
311         p_Id(Temp_bool = True
312
313     if p_Id(Temp_bool and signal > 0.1*I_TEST and p_Id_rise_10_bool == False:
314         p_Id_rise_10_time = time
315         p_Id_rise_10_signal = signal
316         p_Id_rise_10_bool = True
317
318     if p_Id_rise_10_bool and signal > 0.9*I_TEST and p_Id_rise_90_bool == False:
319         p_Id_rise_90_time = time
320         p_Id_rise_90_signal = signal
321         p_Id_rise_90_bool = True
322
323 n = 1
324
325 time_ids = time_ids[n::n]
326 signal_ids = signal_ids[n::n]
327
328 COLOR_GRAF = "lightgray"
329
330 COLOR_LINEAS = "black"
331
332 fig = plt.figure(figsize=(16, 12), facecolor=COLOR_GRAF)
333 ax1 = plt.subplot2grid((4, 3), (0, 0), colspan=3, facecolor=COLOR_GRAF)
334 ax2 = plt.subplot2grid((4, 3), (1, 0), colspan=3, facecolor=COLOR_GRAF)
335 ax3 = plt.subplot2grid((4, 3), (2, 0), colspan=3, facecolor=COLOR_GRAF)
336 ax4 = plt.subplot2grid((4, 3), (3, 0), facecolor=COLOR_GRAF)
337 ax4_right = plt.subplot2grid((4, 3), (3, 1), facecolor=COLOR_GRAF)
338 ax4_text = plt.subplot2grid((4, 3), (3, 2), facecolor=COLOR_GRAF)
339

```

```

340 ax1.plot(time_vgs, signal_vgs, label='Vgs')
341 ax1.plot(avg_time_vgs, avg_signal_vgs, label='Vgs_avg')
342 ax1.plot(p_Vgs_fall_90_time, p_Vgs_fall_90_signal, 'ro')
343 ax1.plot(p_Vgs_fall_50_time, p_Vgs_fall_50_signal, 'bo')
344 ax1.plot(p_Vgs_rise_10_time, p_Vgs_rise_10_signal, 'go')
345 ax1.plot(p_Vgs_rise_50_time, p_Vgs_rise_50_signal, 'mo')
346
347 # Anotaciones para el subplot 1
348 ax1.annotate('Fall 90%', xy=(p_Vgs_fall_90_time, p_Vgs_fall_90_signal), xytext=(10, 10), textcoords='offset points', color='red')
349 ax1.annotate('Fall 50%', xy=(p_Vgs_fall_50_time, p_Vgs_fall_50_signal), xytext=(10, 10), textcoords='offset points', color='blue')
350 ax1.annotate('Rise 10%', xy=(p_Vgs_rise_10_time, p_Vgs_rise_10_signal), xytext=(10, 10), textcoords='offset points', color='green')
351 ax1.annotate('Rise 50%', xy=(p_Vgs_rise_50_time, p_Vgs_rise_50_signal), xytext=(10, 10), textcoords='offset points', color='purple')
352
353 ax1.set_title('Senal de Vgs en funcion del Tiempo', color=COLOR_LINEAS)
354 ax1.set_ylabel('Vgs (V)', color=COLOR_LINEAS)
355 ax1.tick_params(axis='x', colors=COLOR_LINEAS)
356 ax1.tick_params(axis='y', colors=COLOR_LINEAS)
357 # Anadir una leyenda
358 ax1.legend()
359
360 # Mostrar la grafica
361 ax2.plot(time_ids, signal_ids, label='Ids', color = "orange")
362 ax2.plot(p_Id_fall_10_time, p_Id_fall_10_signal, 'ro')
363 ax2.plot(p_Id_rise_10_time, p_Id_rise_10_signal, 'bo')
364 ax2.plot(p_Id_fall_90_time, p_Id_fall_90_signal, 'mo')
365 ax2.plot(p_Id_rise_90_time, p_Id_rise_90_signal, 'go')
366
367
368 # Anotaciones para el subplot 2
369 ax2.annotate('Id Fall 10%', xy=(p_Id_fall_10_time, p_Id_fall_10_signal), xytext=(10, 10), textcoords='offset points', color='red')
370 ax2.annotate('Id Rise 10%', xy=(p_Id_rise_10_time, p_Id_rise_10_signal), xytext=(10, 10), textcoords='offset points', color='blue')
371
372 ax2.annotate('Id Fall 90%', xy=(p_Id_fall_90_time, p_Id_fall_90_signal), xytext=(10, 10), textcoords='offset points', color='purple')
373 ax2.annotate('Id Rise 90%', xy=(p_Id_rise_90_time, p_Id_rise_90_signal), xytext=(10, 10), textcoords='offset points', color='green')
374
375
376
377 ax2.set_title('Senal de Vds en funcion del Tiempo', color=COLOR_LINEAS)
378 ax2.set_ylabel('Vds (V)', color=COLOR_LINEAS)
379 ax2.tick_params(axis='x', colors=COLOR_LINEAS)
380 ax2.tick_params(axis='y', colors=COLOR_LINEAS)
381 ax2.legend()
382
383 ax3.plot(time_vds, signal_vds, label='Vds', color= "red")
384 ax3.plot(p_Vds_fall_90_time, p_Vds_fall_90_signal, 'ro')
385 ax3.plot(p_Vds_fall_10_time, p_Vds_fall_10_signal, 'bo')
386 ax3.plot(p_Vds_rise_10_time, p_Vds_rise_10_signal, 'go')
387 ax3.plot(p_Vds_rise_90_time, p_Vds_rise_90_signal, 'mo')
388
389 ax3.set_title('Senal de Vds en funcion del Tiempo', color=COLOR_LINEAS)
390 ax3.set_xlabel('Tiempo (s)', color=COLOR_LINEAS)
391 ax3.set_ylabel('Vds (V)', color=COLOR_LINEAS)
392 ax3.tick_params(axis='x', colors=COLOR_LINEAS)
393 ax3.tick_params(axis='y', colors=COLOR_LINEAS)
394 ax3.legend()
395
396 # Anotaciones para el subplot 3
397 ax3.annotate('Fall 90%', xy=(p_Vds_fall_90_time, p_Vds_fall_90_signal), xytext=(10, 10), textcoords='offset points', color='red')
398 ax3.annotate('Fall 10%', xy=(p_Vds_fall_10_time, p_Vds_fall_10_signal), xytext=(10, 10), textcoords='offset points', color='blue')
399 ax3.annotate('Rise 10%', xy=(p_Vds_rise_10_time, p_Vds_rise_10_signal), xytext=(10, 10), textcoords='offset points', color='green')
400 ax3.annotate('Rise 90%', xy=(p_Vds_rise_90_time, p_Vds_rise_90_signal), xytext=(10, 10), textcoords='offset points', color='purple')
401
402
403 power_signal = [a * b for a, b in zip(signal_vds, signal_ids)]
404

```

```

405 ## GRAFICOS DE POTENCIAS
406
407 INICIO_PERDIDA_APAGADO = time_ids.index(p_Vds_rise_10_time)
408 FIN_PERDIDA_APAGADO = time_ids.index(p_Id_fall_10_time)
409
410 INICIO_PERDIDA_PRENDIDO = time_ids.index(p_Id_rise_10_time)
411 FIN_PERDIDA_PRENDIDO = time_ids.index(p_Vds_fall_10_time)
412
413 ##GRAFICO APAGADO
414 ax4.plot(time_ids[INICIO_PERDIDA_APAGADO:FIN_PERDIDA_APAGADO], power_signal[INICIO_PERDIDA_APAGADO:FIN_PERDIDA_APAGADO], label='Power Signal')
415 ax4.fill_between(time_ids[INICIO_PERDIDA_APAGADO:FIN_PERDIDA_APAGADO], power_signal[INICIO_PERDIDA_APAGADO:FIN_PERDIDA_APAGADO], color='skyblue', alpha=0.4)
416 ax4.set_title('Senal de Potencia Apagado', color=COLOR_LINEAS)
417 ax4.set_xlabel('Tiempo (s)', color=COLOR_LINEAS)
418 ax4.set_ylabel('Potencia (W)', color=COLOR_LINEAS)
419 ax4.tick_params(axis='x', colors=COLOR_LINEAS)
420 ax4.tick_params(axis='y', colors=COLOR_LINEAS)
421 ax4.legend()
422
423 ##GRAFICO PRENDIDO
424 ax4_right.plot(time_ids[INICIO_PERDIDA_PRENDIDO:FIN_PERDIDA_PRENDIDO], power_signal[INICIO_PERDIDA_PRENDIDO:FIN_PERDIDA_PRENDIDO], label='Power Signal')
425 ax4_right.fill_between(time_ids[INICIO_PERDIDA_PRENDIDO:FIN_PERDIDA_PRENDIDO], power_signal[INICIO_PERDIDA_PRENDIDO:FIN_PERDIDA_PRENDIDO], color='skyblue', alpha=0.4)
426 ax4_right.set_title('Senal de Potencia Prendido', color=COLOR_LINEAS)
427 ax4_right.set_xlabel('Tiempo (s)', color=COLOR_LINEAS)
428 ax4_right.set_ylabel('Potencia (W)', color=COLOR_LINEAS)
429 ax4_right.tick_params(axis='x', colors=COLOR_LINEAS)
430 ax4_right.tick_params(axis='y', colors=COLOR_LINEAS)
431 ax4_right.legend()
432
433
434 Eoff = sum(power_signal[INICIO_PERDIDA_APAGADO:FIN_PERDIDA_APAGADO])*SAMPLE_TIME
435 Eon = sum(power_signal[INICIO_PERDIDA_PRENDIDO:FIN_PERDIDA_PRENDIDO])*SAMPLE_TIME
436
437 print(f"Perdidas por apagado {Eoff}")
438 print(f"Perdidas por encendido {Eon}")
439
440 tr = p_Vds_fall_10_time - p_Vds_fall_90_time
441 tf = p_Vds_rise_90_time - p_Vds_rise_10_time
442 print(f"tr = {tr}")
443 print(f"Nuevo tr = {p_Id_rise_90_time - p_Id_rise_10_time}")
444 print(f"tf = {tf}")
445 print(f"Nuevo tf = {p_Id_fall_10_time - p_Id_fall_90_time}")
446
447
448 texts = [
449     f"Perdidas por encendido (Eon): {{'{{:.2e}}'.format(Eon)}},",
450     f"Perdidas por apagado (Eoff): {{'{{:.2e}}'.format(Eoff)}},",
451     f"Rise Time: {{'{{:.2e}}'.format(tr)}},",
452     f"Fall Time: {{'{{:.2e}}'.format(tf)}},",
453     f"Per. por segundo prendido: {{'{{:.2e}}'.format(Eon*25000)}},",
454     f"Per. por segundo apgado: {{'{{:.2e}}'.format(Eoff*25000)}}"
455 ]
456
457 # Ajustar la posicion y el tamano del texto
458 for i, text in enumerate(texts):
459     ax4_text.text(0.5, 1 - i * 0.2, text, ha='center', va='center', color=COLOR_LINEAS, fontsize=12,
460                 transform=ax4_text.transAxes)
461
462 ax4_text.axis('off') # Ocultar los ejes para el subplot de texto
463
464 ax1.spines['top'].set_color(COLOR_LINEAS)
465 ax1.spines['right'].set_color(COLOR_LINEAS)
466 ax1.spines['bottom'].set_color(COLOR_LINEAS)
467 ax1.spines['left'].set_color(COLOR_LINEAS)
468
469 ax2.spines['top'].set_color(COLOR_LINEAS)
470 ax2.spines['right'].set_color(COLOR_LINEAS)
471 ax2.spines['bottom'].set_color(COLOR_LINEAS)
472 ax2.spines['left'].set_color(COLOR_LINEAS)
473
474 ax3.spines['top'].set_color(COLOR_LINEAS)
475 ax3.spines['right'].set_color(COLOR_LINEAS)
476 ax3.spines['bottom'].set_color(COLOR_LINEAS)
477 ax3.spines['left'].set_color(COLOR_LINEAS)

```

```
477
478 ax4.spines['top'].set_color(COLOR_LINEAS)
479 ax4.spines['right'].set_color(COLOR_LINEAS)
480 ax4.spines['bottom'].set_color(COLOR_LINEAS)
481 ax4.spines['left'].set_color(COLOR_LINEAS)
482
483 ax4_right.spines['top'].set_color(COLOR_LINEAS)
484 ax4_right.spines['right'].set_color(COLOR_LINEAS)
485 ax4_right.spines['bottom'].set_color(COLOR_LINEAS)
486 ax4_right.spines['left'].set_color(COLOR_LINEAS)
487
488 with open(CARPETA+'.txt', "w") as arch:
489     for i in texts:
490         arch.write(i + "\n")
491
492
493
494
495
496 plt.tight_layout()
497 plt.show()
```