Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de un Governor basado en Control Inteligente de Temperatura

Governor design based on intelligent control of temperature

Autor/es

Pablo Hernández Almudi

Director/es

Montijano Muñoz, Eduardo
Suárez Gracia, Darío

Escuela de Ingeniería y Arquitectura

2017

Pablo Hernández Almudi: *Diseño de un Governor basado en Control Inteligente de Temperatura*

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

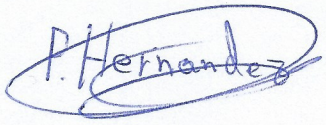
D./Dª. Pablo Hernández Almudi,

con nº de DNI 73159947Y en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Diseño de un Governor basado en Control Inteligente de Temperatura,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 28 de Agosto de 2017



Fdo: Pablo Hernández Almudi

Gracias a Eduardo y Darío por tantas horas ayudando y siendo pacientes conmigo.

Gracias también a mis padres, hermana y amigos por aguantarme siempre que estaba con el trabajo.

DISEÑO DE UN GOVERNOR BASADO EN CONTROL INTELIGENTE DE TEMPERATURA – RESÚMEN

Hoy en día los procesadores son tan potentes y pequeños que generan muchísimo calor. Disipar este calor se ha vuelto uno de los mayores desafíos actualmente. Con este trabajo queremos presentar una opción para tener un mayor control sobre la temperatura de los procesadores, pudiendo elegir a que temperatura queremos que trabajen.

En este proyecto se describen los pasos que se llevan a cabo para poder realizar un governor de frecuencias basado en un control PID para mantener una temperatura constante en el procesador.

Además de ser un trabajo del área de la Ingeniería Informática se realiza tanto un modelado de la disipación de un procesador comercial como el diseño de un controlador PID. Esto permite ampliar el área de conocimiento a la vez que se aplican técnicas de control en el área de la informática.

Así pues, en el trabajo se describen las distintas metodologías aplicadas para resolver el problema del control. Ejecución de pruebas para modelar la forma en la que el calor se disipa desde los transistores hasta el aire y como influye en él la frecuencia.

Con este modelo se diseña un control PID que se amolde a la disipación a la vez que sea sencillo computacionalmente para poder implementarlo en un sistema operativo. Al control PID se le realizan pruebas para poder ajustar los valores de control y ver que funciona adecuadamente.

Sabiendo que el control funciona correctamente, implementamos un governor de frecuencias en el sistema operativo *Linux*. De este modo se recogen directamente los datos de la temperatura y realiza el control PID dentro del espacio de *Kernel* sin intervención del usuario y posibilitando su distribución dentro del sistema operativo.

Por último, probamos este nuevo sistema y lo comparamos en términos de rendimiento con el governor por defecto para comprobar que nuestro control, a parte de permitir mantener una temperatura constante y ajustable, puede obtener un throughput mayor, y por tanto hacer que el sistema operativo funcione más rápido.

ÍNDICE

1 INTRODUCCIÓN 1
1.1 Motivación . . . 1
1.2 Objetivos . . . 2
1.3 Alcance . . . 3
1.4 Organización . . . 3
2 ESTADO DEL ARTE 5
2.1 Generación del calor . . . 5
2.2 Disipación del calor . . . 6
2.3 Gestión de Temperatura en Linux . . . 7
2.4 Gestión de Frecuencias en Linux . . . 8
2.5 Plataforma . . . 9
2.6 Carga de Trabajo . . . 10
3 ANÁLISIS Y DISEÑO DE UN CONTROLADOR PID 11
3.1 Modelado . . . 11
3.1.1 Descripción . . . 11
3.1.2 Obtención del modelo . . . 12
3.2 Control PID . . . 15
4 PRUEBAS CONTROLADOR 17
4.1 Control . . . 17
4.2 Pruebas . . . 18
4.2.1 Control P . . . 18
4.2.2 Control PI . . . 19
4.2.3 Control PID . . . 21
5 IMPLEMENTACIÓN EN KÉRNEL 22
5.1 Módulo . . . 22
5.2 Inicialización de Governor . . . 22
5.3 Sysfs . . . 23
5.4 Tarea Periódica . . . 24
5.5 PID . . . 25
6 RESULTADOS 26
6.1 Prueba de parámetros . . . 26
6.2 Comparación Rendimiento . . . 28
7 CONCLUSIONES 31

BIBLIOGRAFÍA 32

LISTADO DE FIGURAS Y TABLAS 34

GLOSARIO 36
APÉNDICES 36
A DIAGRAMA DE GANTT 37

B PRUEBAS PID 39
B.1 Pruebas solo Proporcional 39
B.2 Pruebas Proporcional e Integral 41
B.3 Pruebas Governor Proporcional, Integral y Derivativo . 42
C COMO USAR PROGRAMA PRUEBAS 45

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

Hoy en día la potencia computacional ha llegado a niveles tan altos que cualquier dispositivo móvil tiene más potencia que un superordenador de tan solo hace unos años. Esto se debe a la mejora tanto en el diseño, como en la manufacturación de los procesadores, lo que también ha provocado que la densidad de potencia de un procesador actual sea comparable a la de una placa vitrocerámica[2].

Disipar el calor se ha convertido en un gran problema en el que se invierte una gran cantidad de tiempo en investigación y fondos. Los centros de datos basan su infraestructura en la expulsión del calor, en la cual se va una gran parte de los recursos. En el mundo de la telefonía los procesadores no pueden alcanzar temperaturas muy altas dado que podrían quemar las manos de los usuarios. El problema radica en que, el rendimiento que se obtiene de un procesador está relacionado con el calor que genera. Cuanto más trabajo se pida a un procesador, más se calentará, siendo esta la relación sobre la que nos vamos a centrar.

Para controlar la temperatura de un procesador existen muchas técnicas. Las técnicas hardware emplean métodos mecánicos para incrementar la disipación, como los ventiladores o refrigeradores. La principal limitación de estas técnicas es su alto consumo eléctrico. También existen técnicas de decremento automático de la frecuencia o inserción de instrucciones vacías, pero sobre estas técnicas no tenemos control. Por tanto, en este trabajo nos vamos a centrar en técnicas software. Actualmente en *Linux* los governors que gestionan la temperatura emplean heurísticas empíricas sin fundamento sólido por detrás. La gestión se basa en puntos de activación que provocan acciones predefinidas tales como activar ventiladores, limitar las frecuencias o apagar el procesador para protegerlo.

En este trabajo exploramos un método pasivo para el control de la temperatura sin necesidad de usar dispositivos externos al procesador. Para ello usaremos el control de la frecuencia para ofrecer un alto rendimiento a la vez que se mantiene una temperatura controlada. Esto ofrecería a los centros de datos la capacidad de controlar

sus consumos eléctricos relacionados con la expulsión del calor pudiendo elegir la temperatura que alcanzan los procesadores.

Actualmente ya existe una aproximación que utiliza un controlador PID para regular la temperatura llamado *Power Allocator* [13]. La principal limitación de este controlador es que no regula directamente la frecuencia, sino que simplemente limita la frecuencia máxima que se puede seleccionar. La decisión final de la frecuencia de trabajo es llevada acabo por un governor diferente. Por lo que este control solo sirve para proteger el procesador de altas temperaturas.

El kernel de Linux considera por tanto, que el control de la temperatura y frecuencia deben ser independientes uno del otro y entre ellos no puede existir ningún tipo de intercambio de información. Si ambos trabajaran juntos y compartieran información, se podrían prever comportamientos a nivel térmico y de este modo, tener un mejor control de la temperatura del procesador a la vez que se consiguen mantener unos niveles de rendimiento estables. Es sobre ésta hipótesis donde nosotros vamos a trabajar.

1.2 OBJETIVOS

El objetivo principal de este trabajo consiste en mejorar el rendimiento de un procesador haciendo uso de un control de temperatura. Para ello se han planteado subobjetivos que ayuden a alcanzarlo.

Primero, debemos entender como se genera el calor en un procesador y como este se disipa desde un punto de vista termodinámico. Esto nos permitirá entender mejor que mecanismos entran en juego a la hora de calentar y enfriar un procesador.

Entendiendo esto deberemos estudiar como diseñar un control automático PID, proporcional, integral y derivativo. Los modelos que hacen falta para elegir el control, entender las variables para poder adecuar el comportamiento y los cálculos necesarios para aplicar el control en un ordenador.

Tras lo cual realizaremos una primera implementación en espacio de usuario para poder realizar pruebas. Y aplicar el mecanismo de control dentro del kernel de *Linux* para que este control se realice nativamente sin intervención del usuario.

Por último se evaluará en cuestión de rendimiento y se comparará con el governor usado actualmente.

1.3 ALCANCE

La dificultad de este trabajo se encuentra tanto en el conocimiento profundo del kernel de *Linux* como en la aplicación de conocimientos de la rama de automática en informática. Los mecanismos de control y la caracterización del intercambio de temperatura son conceptos existentes, pero de los que no se tienen conocimientos previos, por lo que se han debido estudiar a la hora de poder aplicarlos.

Para el modelado y el diseño del control se han utilizado las herramientas *Matlab* y *Simulink*. Matlab permite simplificar y automatizar la generación de modelos para la temperatura, y Simulink permite probar teóricamente el funcionamiento del control PID en base a los modelos previamente calculados.

Para la realización de las pruebas para poder modelar, se ha construido un programa de pruebas cuyo trabajo es homogéneo y controlable, que además obliga al procesador a disipar mucha energía.

Utilizando como punto de partida los governors existentes en Linux, en el TFG se ha programado un nuevo governor que implementa todas las funciones necesarias para el control de la temperatura utilizando un PID.

Por último se realiza una evaluación y comparación del control con respecto a las opciones usadas actualmente.

En el Apéndice A se encuentra el diagrama de Gant con la distribución de esfuerzos.

1.4 ORGANIZACIÓN

El documento se encuentra dividido en base a los pasos que se deben realizar para alcanzar el objetivo final.

En el Cap. 2 se habla sobre los fundamentos teóricos que existen tras la generación del calor y su disipación.

En el Cap. 3 se realiza un análisis del comportamiento de la temperatura y se diseña un control PID en base a esa caracterización.

En el Cap. 4 se describen e interpretan las pruebas realizadas con el control PID previamente diseñado.

En el Cap. 5 se implementa el controlador dentro del kernel de *Linux*

En el Cap. 6 se obtienen resultados finales sobre el controlador y se compara con los métodos de control actuales.

Por último en el Cap. 7 se da la conclusión sobre el proyecto y se hace un resumen de todos los objetivos que se han cumplido.

A continuación se adjuntan los anexos en los que se incluyen resultados de pruebas realizadas.

2

ESTADO DEL ARTE

Este capítulo presenta como los procesadores disipan energía y su impacto en la temperatura para después describir los mecanismos del sistema operativo *Linux* para gestionar la temperatura y el consumo energético.

2.1 GENERACIÓN DEL CALOR

En un procesador toda la energía consumida haciendo operaciones se transforma en calor. La energía se emplea tanto para la conmutación de los transistores como para la transmisión de información a través del cableado [15]. Esto provoca la transformación de la corriente eléctrica en calor y se puede entender como la suma de dos tipos de consumo: estático y dinámico, tal y como muestra la ecuación

$$P_{total} = P_{est} + P_{dyn}. \tag{1}$$

El consumo estático es el consumo de los transistores que no se emplea en la conmutación y se debe a fugas. Estas fugas se producen por culpa de la corriente subumbral. Esta se produce cuando la tensión aplicada a un transistor en *off* no alcanza el punto de activación. Pese a que se supone que el transistor está apagado, este, por motivos de fabricación, permite circular una pequeña corriente. Por tanto, la potencia estática simplificada se puede ver como,

$$P_{est} = I_{sub} V_{dd}. \tag{2}$$

Donde I_{sub} es la corriente subumbral y V_{dd} la tensión de alimentación. Por otro lado, la mayor fuente de calor del procesador es la conocida como *Potencia Dinámica* siendo esta causada por la conmutación de los transistores. La cantidad de potencia que se genera y se disipa en forma de calor se puede aproximar mediante la siguiente ecuación,

$$P_{dyn} = CV^2 f. \tag{3}$$

Los términos C, V y f representan la capacitancia, la tensión de alimentación y la frecuencia, respectivamente. La capacitancia varia dependiendo de la instrucción que se ejecuta, porque cada una hace conmutar a un conjunto distinto de transistores. Para la simplificación de los cálculos, lo que se hace es obtener un valor medio de la capacitancia para poder usar la variable como constante.

En el resto de variables vemos como la tensión es el factor con más peso dado que aumenta al cuadrado. La frecuencia por otro lado, varia de manera lineal a la potencia disipada. Sin embargo, en un procesador estas variables no son independientes. Para poder incrementar la frecuencia se requiere también un incremento en el voltaje. Esto es debido al comportamiento tanto de condensador como de resistencia que tienen los cables en un procesador, si no se alcanza la tensión umbral en el tiempo permitido, la información no llegaría a tiempo a los distintos circuitos y se producirían errores.

Con todo esto en cuenta podemos resumir cómo se genera el calor mediante la siguiente fórmula, donde t es el tiempo de medida:

$$Q = Q_{dinámica} + Q_{estática} = (CV^2 f + I_{sub} V_{sub})t. \tag{4}$$

2.2 DISIPACIÓN DEL CALOR

A la hora de disipar el calor generado lo que se busca es ampliar la superficie de disipación y de este modo poder intercambiar una mayor cantidad de calor con la sustancia refrigerante. En nuestro caso se trata del aire de la habitación, aunque también se usan refrigerantes tales como agua o aceites especiales. En la práctica totalidad de procesadores, estos vienen de fábrica encapsulados en un material muy conductivo térmicamente y con una superficie mucho mayor al silicio del procesador. Sobre el empaquetado se coloca un radiador con un ventilador para mejorar la disipación del calor. En los móviles y dispositivos pequeños esto es imposible dadas las limitaciones de espacio pero, por otro lado, la CPU es más pequeña y de menor consumo, por lo que el calor se disipa a través del chasis del móvil y la batería.

La plataforma sobre la que vamos a ejecutar las pruebas sólo tiene el procesador encapsulado por lo que la disipación se modela más sencillamente.

La disipación del calor se rige por la ecuación,

$$\dot{Q} = \dot{m} C_p \frac{dT_{\mu P}}{dt} + \frac{T_{\mu P} - T_{air}}{R_t} \tag{5}$$

donde:

- Q es el calor que se genera.
- m es la masa, representada por el empaquetado del procesador el cual es el que disipa el calor.
- C_p es el calor especifico del material del empaquetado.
- $\frac{dT_{\mu P}}{dt}$ es el incremento de la temperatura del procesador.
- $T_{\mu P}$ es la temperatura del procesador.
- T_{air} es la temperatura del aire.
- R_t es la resistencia térmica, la que determina la velocidad a la que se disipa el calor.

En la primera parte de la fórmula se representa el calor que se produce en el procesador y que es repartido al encapsulado. En la segunda parte se modela como el calor se disipa desde el encapsulado al aire. Dado que no sabemos de que material está formado el empaquetado no podemos usar ésta fórmula para entender la disipación de la temperatura, aunque nos indica cual es la dinámica que sigue el calor desde el procesador al aire.

2.3 GESTIÓN DE TEMPERATURA EN LINUX

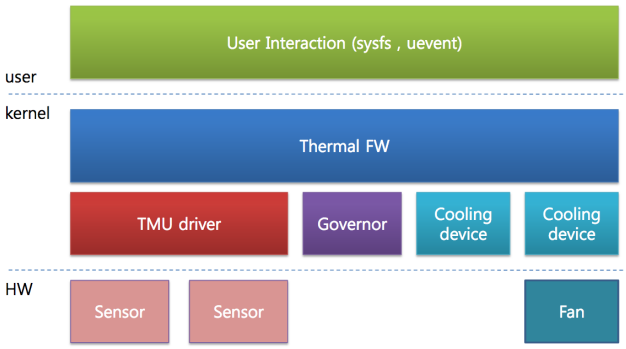


Fig. 1: Esquema componentes del Kernel [6]

Linux utiliza dos mecanismos distintos para gestionar la frecuencia, por un lado un sistema que comprueba la temperatura y por otro un sistema que se encarga de elegir la frecuencia de trabajo del procesador. En la Fig. 1 podemos ver un resumen de los elementos. Por un lado, en el hardware tenemos sensores y dispositivos mecánicos de enfriamiento. Después, en el sistema operativo en espacio de usuario tenemos una interfaz para modificar el comportamiento del sistema

térmico. Por último, tenemos el sistema térmico como tal en espacio de Kernel en el que se encuentra el governor y dispositivos de enfriamiento.

El sistema que monitoriza la temperatura se conoce como *Thermal Governor*. La tarea de este sistema es muy simple, comprueba la temperatura del dispositivo, y en base a unos puntos de disparo activa distintos mecanismos de enfriamiento. Estos eventos pueden controlar la velocidad de ventiladores o limitar la frecuencia máxima.

El *thermal governor* se organiza en zonas térmicas. La Fig. 2 muestra los componentes de una zona térmica. El sensor obtiene la temperatura y el governor regula y actúa sobre los dispositivos de enfriamiento que, a su vez, puede que controlen elementos mecánicos como ventiladores. De este modo se posibilita asignar dispositivos a un governor en la configuración del kernel personalizándolos para cada procesador.

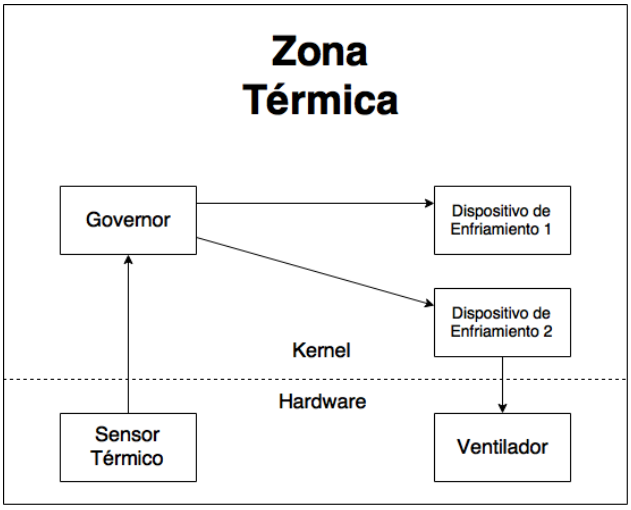


Fig. 2: Esquema de zonas térmicas

Uno de estos governors es *Power Allocator*. Este basa su funcionamiento en un control PID que limita la frecuencia máxima a la que puede funcionar el procesador, dejando el control a un governor de frecuencia. Por el contrario nuestro controlador esta diseñado para actuar directamente sobre la frecuencia, realizando un control directo con respecto a los valores de entrada.

2.4 GESTIÓN DE FRECUENCIAS EN LINUX

El governor de frecuencias es un sistema que se encarga de elegir las frecuencias de trabajo del procesador. La elección se realiza de

acuerdo a distintos parámetros y siempre eligiendo entre las frecuencias que se encuentran disponibles. Los principales son:

- **Ondemand** es el que viene por defecto en la mayoría de sistemas, elige la frecuencia en base a la carga de trabajo del procesador.
- **Performance** elige siempre la frecuencia más alta disponible para mejorar el rendimiento.
- **Userspace** no es un governor como tal, sino un sistema que permite al usuario elegir la frecuencia del procesador.
- **Powersave** mantiene la mínima frecuencia disponible.

2.5 PLATAFORMA

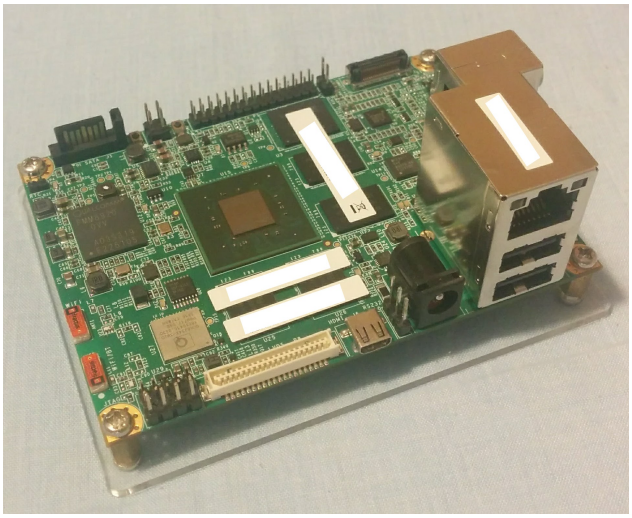


Fig. 3: Imagen de la plataforma

Para la realización del proyecto se va a utilizar la plataforma de desarrollo *IFC6410* de la empresa *Inforce*, la cual es un ordenador de placa reducida o como se conoce a estos dispositivos *Single Board Computer o* ***SBC***. Cuenta entre otras cosas con un procesador *Qualcomm Snapdragon 600* con cuatro núcleos krait 300 hasta 1,7 GHz de frecuencia de reloj. Este procesador viene sin disipador interno, por lo que tan solo cuenta con la disipación que aporta el empaquetado tal y como se aprecia en la Fig. 3.

En el apartado de software, sobre la placa se está ejecutando *Debian 8.3* junto a una versión realizada por la organización *Linaro* de la versión del Kernel 4.4 de Linux.

2.6 CARGA DE TRABAJO

A la hora de realizar tanto pruebas de rendimiento como de control, necesitamos establecer una carga de trabajo estable que permita simplificar el desarrollo del control. Introducir la variable de carga de trabajo en un control añade una complejidad más allá de los límites de este trabajo.

Nosotros pretendemos simular el comportamiento en un centro de datos en los cuales la carga es siempre muy alta. En nuestro caso, se va a tratar de multiplicación de matrices, las cuales se usan en muchas aplicaciones entre las que se incluye Deep Learning.

Para la creación del programa aprovechamos una librería matemática propia del fabricante del procesador llamada *Snapdragon Math Libraries*[11]. Esta librería nos permite realizar operaciones de álgebra lineal de manera muy eficiente sobre plataformas Qualcomm, entre las que se incluyen multiplicación de matrices. Esta multiplicación se va a realizar con números en coma flotante dado que este tipo de operaciones son las más exigentes con los procesadores.

En concreto la aplicación crea dos matrices aleatorias de $8192 * 8192$ y las multiplica sucesivamente[7] dentro de un bucle con 10 iteraciones. Este tamaño de matriz se ha escogido para que en cada iteración se deje suficiente tiempo para calentar el procesador y llegar a régimen permanente. A su vez, gracias al uso de la librería matemática estas multiplicaciones se realizan de manera simultánea en varios cores.

3

ANÁLISIS Y DISEÑO DE UN CONTROLADOR PID

En este capítulo se explica el funcionamiento del control de la temperatura a través de la modificación de la frecuencia. Lo primero que se hace es un estudio del sistema con el objetivo de obtener un modelo empírico que nos sirva para poder simular el comportamiento de la temperatura en el procesador. En la segunda parte del capítulo se describe el diseño del control PID, de manera que se obtenga una respuesta que maximice el rendimiento a la vez que se mantiene una temperatura correcta.

3.1 MODELADO

3.1.1 *Descripción*

El primer paso a la hora de crear cualquier tipo de control sobre un sistema es entender cómo se comporta. Por ello, la primera tarea de cara a proponer una estrategia de control es obtener un modelo matemático del comportamiento de la temperatura en función de la frecuencia. Puesto que no podemos modelar el comportamiento térmico del procesador basándonos en las características de los materiales, debido a que no los conocemos y es una tarea muy compleja, en este TFG se propone realizar un modelo empírico del sistema a través de la ejecución de experimentos.

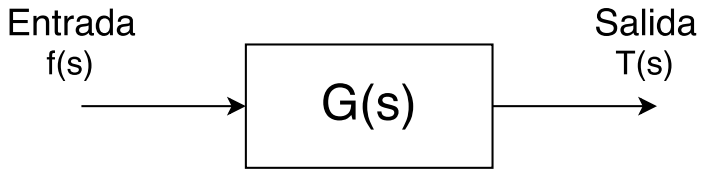


Fig. 4: Función de tranferencia

Este modelo empírico se obtiene en forma de función de transferencia, que describe como se comporta la temperatura en base a la frecuencia, Fig. 4. La función que vamos a usar se trata de un primer orden con un retraso, dado que se sabe que los sistemas que intercambian calor se pueden modelar con este tipo de funciones[4]. Es importante notar que utilizando esta función, estamos considerando que

el modelo de comportamiento de nuestro sistema es lineal. Además, este modelo no tienen en cuenta la presencia de perturbaciones externas, como puede ser la temperatura exterior. La función en cuestión es

$$G(s) = \frac{k_p}{1 + T_{p1}s} e^{-T_d s}. \tag{6}$$

Esta describe el comportamiento en base a k_p, el cual relaciona la entrada con la salida en régimen permanente. En nuestro caso indica el valor final de la temperatura que se alcanza en cada frecuencia si la mantenemos constante. El parámetro T_{p1} indica la velocidad a la que se alcanza esa temperatura y T_d el retraso que existe entre que se aplica la acción hasta que la salida cambia. Por último, s es una variable compleja que se obtiene al aplicar la transformada de Laplace a la ecuación diferencial equivalente. Esta transformación es habitual en el ámbito de la ingeniería de sistemas y tiene como objetivo simplificar el procedimiento de ajuste del controlador, transformando las ecuaciones diferenciales en ecuaciones algebraicas.

Por tanto, el problema de modelado del sistema se puede resumir en encontrar los valores adecuados de k_p T_{p1} y T_d.

3.1.2 *Obtención del modelo*

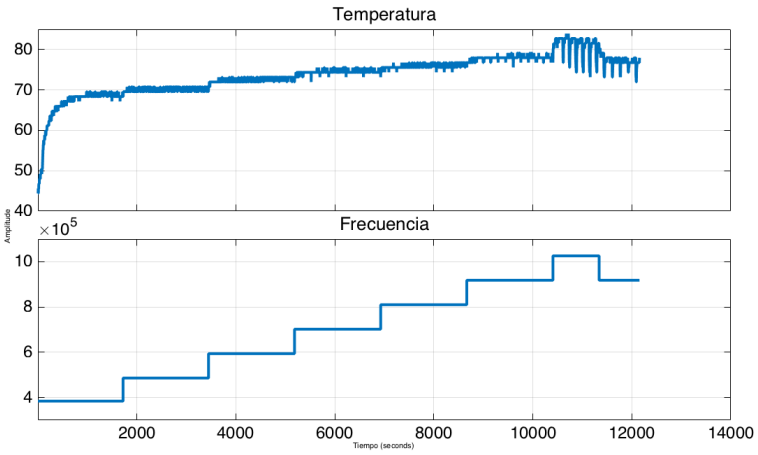


Fig. 5: Relación frecuencia temperatura de la ejecución continua

Tras entender el modelo, nos disponemos a realizar los experimentos. Estos, tienen como objetivo tomar muchas medidas de temperatura para diferentes frecuencias. De este modo, tendremos mucha información con la cual poder ajustar los parámetros del modelo. Dado que el modelo puede cambiar dependiendo de la carga de trabajo que se tenga, vamos a realizar todas las pruebas con la misma carga

de trabajo utilizando el benchmark de manera continuada. De esta manera simplificamos un problema no trivial.

El primer experimento realizado consiste en, partiendo desde la frecuencia más baja, ir incrementándola de manera paulatina. En cada frecuencia damos tiempo a que la temperatura pueda alcanzar el valor en régimen permanente. Tras ejecutar la prueba podemos ver en la Fig. 5 la temperatura real y la frecuencia. Podemos observar como, para cada frecuencia se produce un escalón en la temperatura. Este escalón es la temperatura en régimen permanente, cuya relación con la frecuencia se define en Ec. 6 como k_p. También podemos ver como, cuando se llega a 1026 MHz, el procesador alcanza una temperatura crítica y disminuye la frecuencia automáticamente para protegerse.

Partiendo de los datos de la figura realizamos la estimación de los parámetros del modelo. Para ello utilizamos la *System Identificacion Toolbox de Matlab* [3] [8]. En la gráfica Fig. 6 vemos la comparación entre la temperatura real y la calculada por el modelo. Los valores devueltos por el modelo son:

- $K_p = 9.91 \cdot 10^{-5}$
- $T_{p1} = 1 \cdot 10^{-6}$
- $T_d = 11.601$

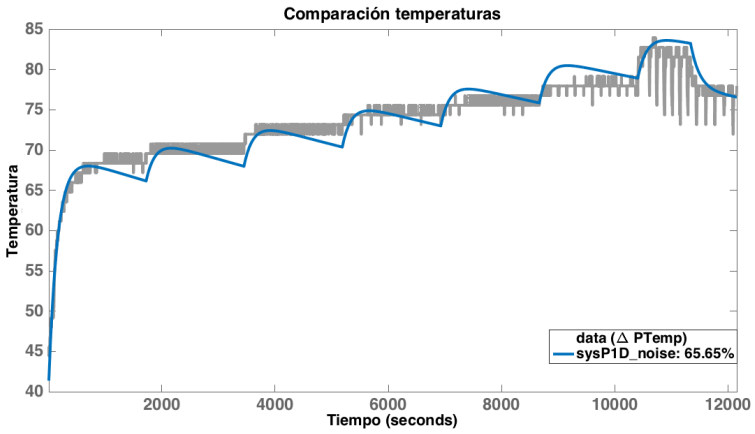


Fig. 6: Relación Temperatura real y modelada de la ejecución continua

En la Fig. 6 se observa que el ajuste no es todo lo bueno que nos gustaría que fuera. Esto se puede deber a la no linealidad del sistema y a la presencia perturbaciones. Por tanto, se procede a realizar un ajuste de los parámetros del modelo considerando varias pruebas, en las que cada experimento se ejecuta con una frecuencia constante y distinta en cada prueba. Dando como resultado la Tabla 1 donde se ven los valores que corresponden del modelo a cada frecuencia.

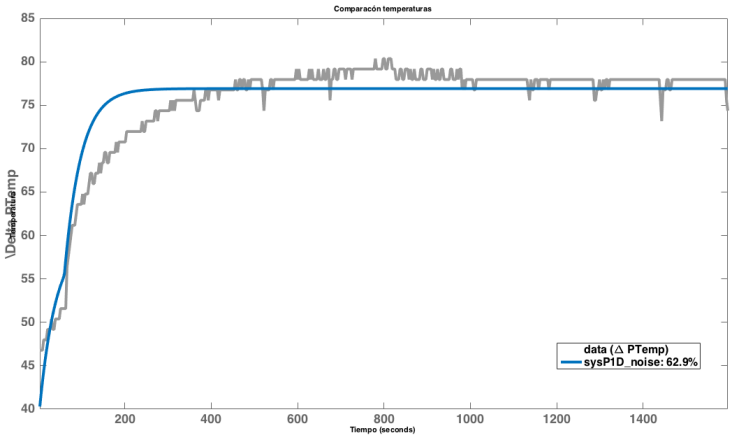
Tabla 1: Resultados del modelado

Frecuencia (MHz)	K_p	T_{p1}	T_d
702	$10.4 \cdot 10^{-5}$	122.31	8.72
810	$9.37 \cdot 10^{-5}$	140.91	0.0
918	$8.50 \cdot 10^{-5}$	119.93	7.89
1026	$8.27 \cdot 10^{-5}$	114.79	0.0

Aún ejecutando solo una frecuencia constante es muy difícil dar con un modelo preciso, podemos observar que para 810 y 1026 el valor de T_d es 0 por lo que el modelo no es muy exacto. Por tanto, decidimos que, dado que vamos a operar en el rango de temperaturas alrededor de 80-85°C parece conveniente utilizar unos valores próximos a los obtenidos para 918 MHz. En concreto, los valores escogidos son

- $K_p = 8.5\ 10^{-5}$,
- $T_{p1} = 120$,
- $T_d = 8$.

En la Fig. 7 podemos ver como se ajusta nuestro modelo a los datos reales en este caso. La gráfica no concuerda exactamente pero se acerca mucho al comportamiento real y la temperatura en régimen permanente es correcta.



(a) Modelado a 918 MHz

Fig. 7: Resultado del modelo a 918 MHz

3.2 CONTROL PID

Un control PID [5] es un mecanismo de control de lazo cerrado, esto quiere decir que la salida del sistema se realimenta a la entrada, utilizando el error con respecto al valor deseado de la temperatura para el cálculo de la frecuencia adecuada. En la Fig. 8 podemos ver como queda situado el controlador PID junto a la función de transferencia. El control PID aplica tres acciones distintas:

- *Acción Proporcional*, para controlar la dinámica del sistema.
- *Acción Integral*, utilizada principalmente para eliminar errores en el régimen permanente y la influencia de perturbaciones.
- *Acción Derivativa*, cuyo objetivo suele ser mejorar el régimen transitorio.

La función que describe un PID, en función de, s es

$$\mathrm{PID}(s) = \frac{K(T_i s + 1)(T_d s + 1)}{T_i s(\alpha T_d s + 1)} \tag{7}$$

donde $K > 0$ es la acción proporcional, $T_i \geqslant 0$ es la constante de tiempo asociada a la acción integral, $T_d \geqslant 0$ la constante de tiempo de la acción derivativa y $0 < \alpha < 1$ un parámetro de filtrado necesario para mitigar limitaciones físicas de la parte derivativa [5]. Estos son los valores que debemos ajustar para conseguir un control adecuado.

Mediante la implementación de este diagrama Fig. 8 en la herramienta *simulink* de *Matlab* y los parámetros del modelo de la sección anterior se han realizado diferentes pruebas para encontrar unos valores adecuados del controlador.

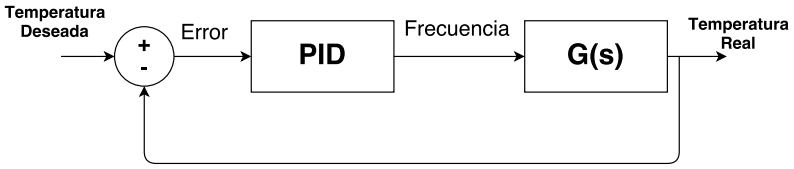


Fig. 8: Diagrama de un controlador PID

Es importante notar que este control no lo podemos aplicar directamente en un algoritmo en un ordenador, debido a que se trata de un control diseñado para un proceso continuo. Es por ello que debemos discretizarlo, tomando muestras periódicas y aplicando el control en una tarea del sistema. Llamando $u(k)$ a la acción calculada por el PID en la iteración k, la fórmula del PID queda

$$u(k) = -Eu(k-1) - Fu(k-2) + Ae(k) + Be(k-1) + Ce(k-2), \tag{8}$$

donde $e(k)$ representa el error, temperatura deseada menos temperatura real del procesador, en la misma iteración y las variables, $A, \ldots, E$ se obtienen discretizando (7) con el periodo de muestreo elegido.

El último paso del algoritmo de control consiste en discretizar la acción calculada, $u(k)$, aproximándola por el valor de la frecuencia del procesador más cercano posible. En el siguiente trozo de código podemos ver el algoritmo PID completo en pseudocódigo.

Algoritmo 1 Algoritmo PID

```
1: while True do
2:     leer_temperatura(T)
3:     e_k ← temperatura_deseada − T
4:     u_k ← −E ∗ u_{k−1} − F ∗ u_{k−2} + A ∗ e_k + B ∗ e_{k−1} + C ∗ e_{k−2}
5:     u_{k−2} ← u_{k−1}
6:     u_{k−1} ← u_k
7:     e_{k−2} ← e_{k−1}
8:     e_{k−1} ← e_k
9:     freq ←DISCRETIZAR(u_k)
10:    cabiar_frecuencias(freq)
11:    esperar(periodo)
12: end while
```

4

PRUEBAS CONTROLADOR

Dada a la complejidad que supone trabajar en modo kernel para realizar las pruebas de ajuste del controlador, debido a que se trabaja en modo sistema y es necesario cambiar la imagen de arranque cada vez que es necesario hacer pruebas. Se ha implementado un banco de pruebas en modo usuario. En el esquema Fig. 9 podemos ver las diferencias que existen entre el controlador en espacio de usuario y el integrado en kernel. El que usamos de pruebas utiliza el sistema de fichero sysfs para cambiar la frecuencia de trabajo gracias al governor *userspace*.

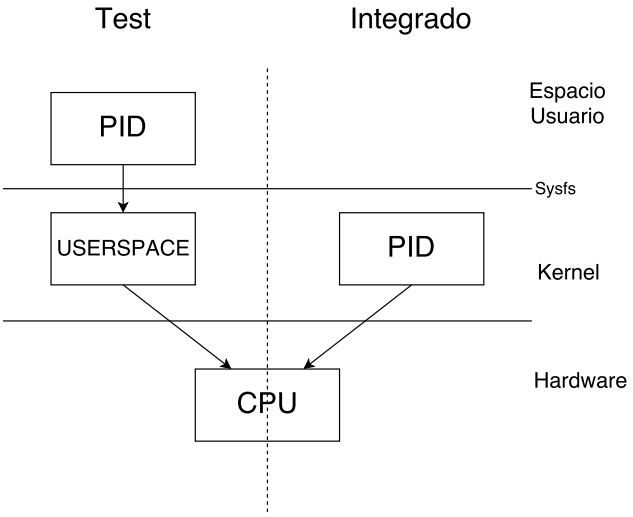


Fig. 9: Esquema diferencias PID en espacio de usuario y PID en kernel

4.1 CONTROL

Las pruebas de control se van a realizar haciendo uso del governor *userspace*. Este governor permite seleccionar a que frecuencia se quiere trabajar mediante la escritura de la frecuencia escogida en un fichero situado en *"/sys/class/system/cpu/cpu*/cpufreq/scaling_setspeed"*. Para obtener información relativa a la temperatura tan solo se debe leer del fichero situado en *"/sys/class/thermal/thermal_zoneo/temp"*.

Gracias a esto, se puede programar un controlador sencillo en espacio de usuario [1], lo que permite poder modificarse con gran fa-

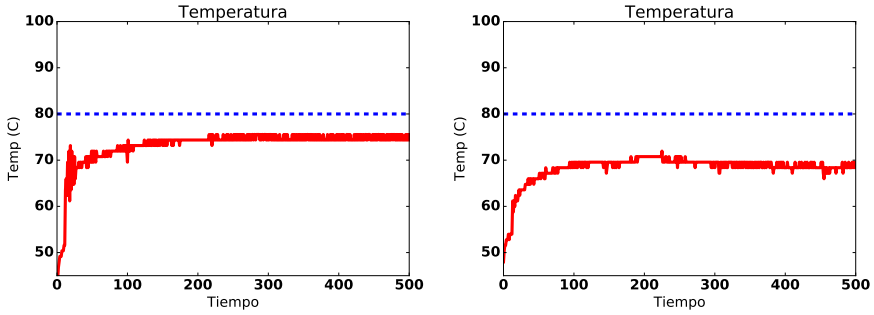
cilidad. Además se realiza en C que es el mismo lenguaje con el que se programa el kernel, por lo que se facilita luego compartir código cuando se deba programar el governor.

Con el propio control vamos a tomar la información que nos sirva para estudiar el comportamiento del PID. La más importante es la frecuencia discreta a la que trabaja el procesador y la temperatura. A la que se añade el error que se introduce en el controlador y la frecuencia teórica calculada.

Para poder estudiar los datos, se muestran en gráficas mediante un script de python [9]. De este modo podemos observar mejor la evolución de los distintos parámetros a lo largo del tiempo, y realizar decisiones en base a los resultados. En el Apéndice B se encuentran todas las pruebas realizadas.

4.2 PRUEBAS

4.2.1 *Control P*



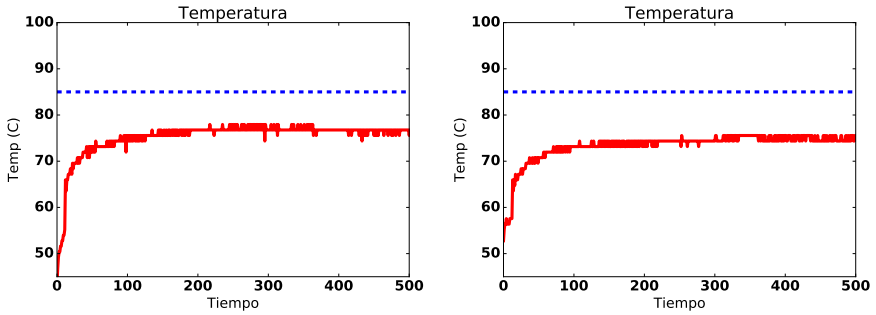
(a) Temperaturas con 100K

(b) Temperaturas con 50K

Fig. 10: Gráficas de prueba con K=100k y 50k y objetivo de 80 grados

Para la batería de pruebas vamos a empezar por probar solo el componente proporcional de PID. Además, vamos a modificar la temperatura objetivo para poder ver como se comporta el control con diferentes temperaturas objetivo. Las temperaturas elegidas son 80, 85 y 90°C, y el proporcional 50000, 75000 y 100000 en la Sección B.1 se recogen todas las pruebas.

Lo primero que vemos en los resultados es, que en ningún caso se termina alcanzando la temperatura objetivo. Esto es debido a que es imposible alcanzar con solo el proporcional el objetivo sin que se produzcan sobreoscilaciones en la temperatura [5]. En las gráficas Fig. 10 y Fig. 11 podemos ver como para un proporcional más alto la temper-



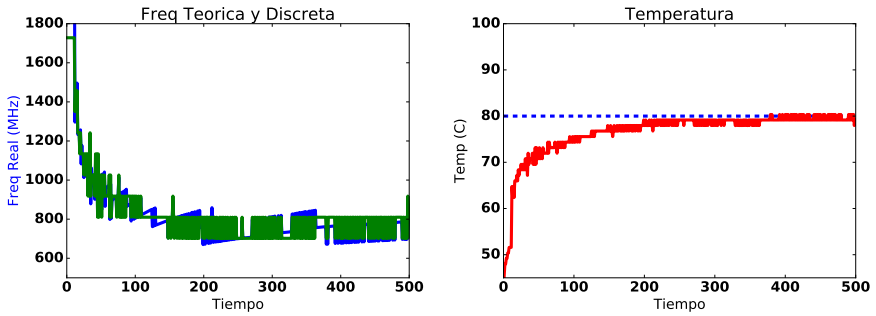
(a) Temperaturas con 75K (b) Temperaturas con 50K

Fig. 11: Gráficas de prueba con K=75k y 50k y objetivo de 85 grados

atura asciende antes. También se acerca más al objetivo cuanto más alto, pero como podemos ver en la gráfica de 100k la temperatura se comporta erráticamente, ampliar el valor solo haría empeorar esos desajustes.

4.2.2 *Control PI*

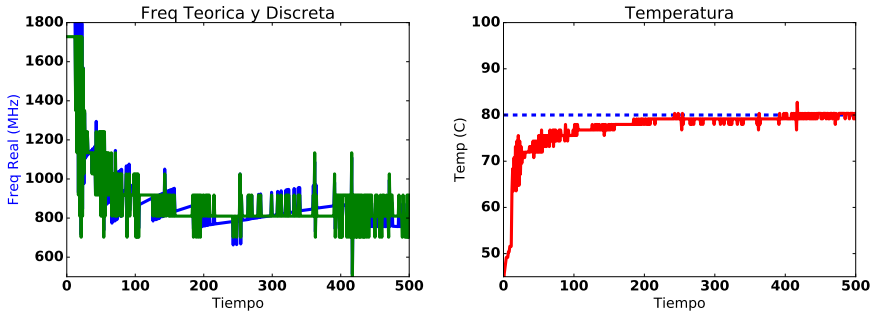
A continuación, vamos a ver como se comporta el control añadiendo la acción integral. En este caso vamos a hacer las pruebas con una T_i de 180 y las pruebas se encuentran en la sección Sección B.2.



(a) Gráfica de frecuencia (b) Gráfica de temperatura

Fig. 12: Gráficas de prueba con T=80 K=75K Ti=180 A=75000 B=-74583 C=0.0 E=-1.0 F=0.0

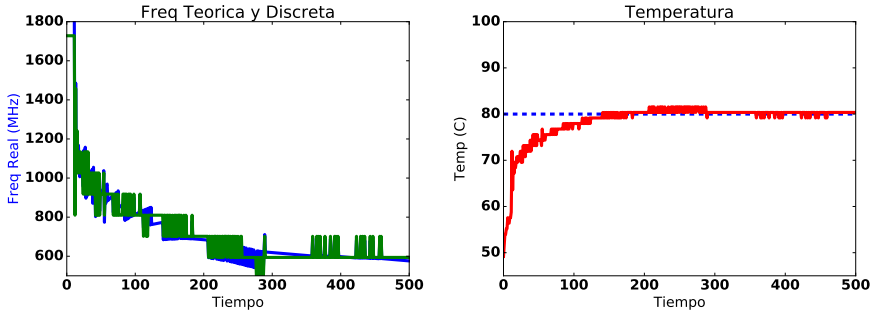
Lo primero que podemos observar es, que en este caso si que alcanzamos la temperatura deseada. Además, podemos ver la diferencia que existe entre usar un valor proporcional alto y uno más bajo. En Fig. 12 con una K de 75000 tenemos una curva de temperatura suave que alcanza la temperatura deseada. Mientras que en Fig. 13 se alcanza antes el valor deseado pero a coste de mucha variación en la temperatura.



(a) Gráfica de frecuencia (b) Gráfica de temperatura

Fig. 13: Gráficas de prueba con T=80 K=100K Ti=180 A=100000 B=-99444 C=0.0 E=-1.0 F=0.0

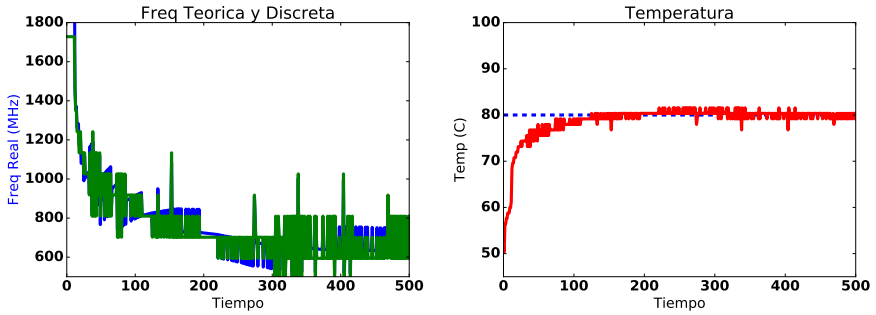
Vamos a probar también a disminuir T_i a 90. Y vemos como la respuesta es mucho más rápida y se alcanza la temperatura deseada en mucho menos tiempo del que ha requerido para los otros valores. Entre la Fig. 14 y Fig. 15 vemos que el valor de K hace que la velocidad sea más alta.



(a) Gráfica de frecuencia (b) Gráfica de temperatura

Fig. 14: Gráficas de prueba con T=80 K=75K Ti=90 A=75000 B=-74167 C=0.0 E=-1.0 F=0.0

Tal y como podemos observar al aumentar el proporcional se aumenta la velocidad con la que se acerca a la temperatura elegida, al igual que ha sucedido antes. En este caso además, vemos como al añadir el valor integral se consigue alcanzar la temperatura deseada. Cuando se incrementa el integral el tiempo que cuesta alcanzar la temperatura deseada aumenta y al decrementar ocurre lo contrario. Esto se ve en las pruebas, con una T_i de 90 se llega antes a la temperatura pero como se ve produce un pequeño sobreajuste. Con una T_i de 180 se ve como se llega correctamente a la temperatura aunque un poco más lentamente, es por eso que se eligen estos valores de integral para seguir haciendo pruebas.

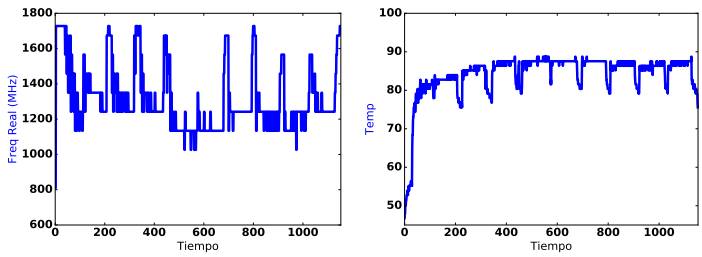


(a) Gráfica de frecuencia (b) Gráfica de temperatura

Fig. 15: Gráficas de prueba con T=80 K=100K Ti=90 A=100000 B=-98889 C=0.0 E=-1.0 F=0.0

4.2.3 *Control PID*

También se ha probado a añadir el derivativo al control, pero, como podemos ver en todas las pruebas en el Sección B.3, no se consigue mantener una temperatura constante. Esto se puede deber principalmente al hecho de que la acción derivativa es muy sensible a ruidos en la medida proporcionada por el sensor. Es por ello que en este TFG, aunque la implementación desarrollada permite su utilización, se ha decidido descartar su uso en la versión implementada en Kernel, considerando únicamente las acciones proporcional e integral.



(a) Gráfica de frecuencia (b) Gráfica de Temperatura

Fig. 16: Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.5 A=200000 B=-396700 C=196710 E=-2.0 F=1.0

En la Fig. 16 tenemos un ejemplo de lo que le sucede a la temperatura. Los picos de la gráfica en los que la temperatura desciende corresponden con el final de cada iteración. Esto indica que la librería de cálculo realiza otras tareas que no calientan tanto el procesador. Con los otros controles esto no se nota, pero el derivativo al ampliar el error nos deja estos errores.

5

IMPLEMENTACIÓN EN KÉRNEL

Ahora que hemos comprobado que el control funciona, vamos a implementarlo en espacio de kernel. De esta manera el control tendrá acceso a toda la información de manera más rápida y además, se permite la distribución del governor incluido en el Kernel. Para la realización nos hemos basado tanto en cpufreq_ondemand.c como cpufreq_conservative.c en la versión 3.7.10 de *Linux*. El código puede encontrarse en *github* [1].

5.1 MÓDULO

Para incluir el Governor dentro del Kernel de *Linux* se debe programar como un módulo. Así, se podrá compilar junto con el resto del kernel y será cargado en el arranque. La programación requiere seguir unas pautas [12] en las que introducen unas llamadas al sistema y declaraciones para ayudar al compilador y al kernel, con el añadido de tener que registrar el governor con la llamada *cpufreq_register_governor*. En el siguiente fragmento de código se puede ver la inicialización del módulo.

```
struct cpufreq_governor cpufreq_gov_pid = {
        .name                   = "PID_GOVERNOR",
        .governor               = cpufreq_governor_dbs,
        .owner                  = THIS_MODULE,
};

static int __init cpufreq_gov_dbs_init(void)
{
        return cpufreq_register_governor(&cpufreq_gov_pid);
}

module_init(cpufreq_gov_dbs_init);
```

5.2 INICIALIZACIÓN DE GOVERNOR

Con el registro del módulo el sistema puede usar el governor PID cuando se requiera. Esto lo realiza mediante una llamada al método

principal que registramos previamente, el cual debe gestionar tanto la inicialización como la parada del governor, mediante la recepción de eventos.

En la inicialización se preparan los datos para cada cpu dado que es cada cpu el que debe gestionar su propia frecuencia. Además se crean los ficheros de configuración que permitirán alterar el funcionamiento y se inicializa el trabajo que posteriormente se encole para gestionar la periodicidad.

5.3 SYSFS

Una de las cosas que queremos facilitar es la capacidad de modificar el comportamiento del governor, posibilitando tanto elegir la acción del PID como la elección de la temperatura deseada. Para ello, hacemos uso del sistema de archivos virtual. Este sistema añadido a partir de la versión 2.6 del kernel de Linux permite crear ficheros virtuales que actúan como intermediarios entre espacio de usuario y kernel.

Los ficheros se crean mediante la llamada *sysfs_create_group* a la que se le incluye un listado con los nombres de los ficheros que queremos que se muestren. Aquí las estructuras y la llamada que permiten registrar los ficheros.

```
static struct attribute *dbs_attributes[] = {
        &sampling_rate.attr,
        &E_value.attr,
        &F_value.attr,
        &A_value.attr,
        &B_value.attr,
        &C_value.attr,
        &temp_obj.attr,
        NULL
};

static struct attribute_group dbs_attr_group = {
        .attrs = dbs_attributes,
        .name = "PID_governor",
};

sysfs_create_group(cpufreq_global_kobject,
                                        &dbs_attr_group);
```

A eso se debe añadir métodos que sirvan para mostrar y leer la información. La muestra de información se realiza fácilmente usando *sprintf* sobre el fichero, y en la lectura *sscanf*. Estos métodos serán utilizados por el Kernel cada vez que se realice una lectura o escritura sobre los ficheros virtuales. Aquí un ejemplo de lectura en un fichero

lo que para el módulo sería escritura en él. Nótese que se usan mutex para evitar la corrupción de datos dado que es un sistema concurrente.

```
static ssize_t store_temp_obj(struct kobject *a, struct attribute
    *b,
                                  const char *buf, size_t count)
{
        int input;
        int ret;
        ret = sscanf(buf, "%d", &input);

        if(ret != 1) return -EINVAL;

        mutex_lock(&dbs_mutex);
        dbs_tuners_ins.temp_obj = input;
        mutex_unlock(&dbs_mutex);

        return count;
}
```

5.4 TAREA PERIÓDICA

Para poder realizar correctamente el control PID éste se debe realizar de manera periódica. En el Kernel esto lo podemos conseguir mediante el uso de colas de trabajo. Las colas de trabajo permiten almacenar un trabajo para que se realice en el futuro y de este modo, podemos decidir en que momento se ejecuta.

Para nuestro governor se debe crear un trabajo por cada cpu dado que cada uno debe gestionar su propia frecuencia. Esto se realiza mediante *schedule_delayed_work_on*. Al que se le indica la cpu, el trabajo a realizar y un retraso en nanosegundos. De este modo, cada vez que se realice el trabajo de cálculo de PID se deberá realizar la llamada para volver a encolar el trabajo.

En el siguiente fragmento de código se puede ver la transformación de microsegundos a *jiffies* los cuales son la medida de tiempo dentro del kernel, y la inicialización y encolado de los trabajos. Dado que el tiempo de cálculo del PID puede ser variable y se quiere que todas las cpus cambien la frecuencia a la vez, se realiza una operación de módulo de los *jiffies*. Gracias a esto podemos sincronizar de manera sencilla todos los trabajos.

```
static inline void dbs_timer_init(struct cpu_dbs_info_s *dbs_info
    )
{
```

```
        int delay = usecs_to_jiffies(dbs_tuners_ins.sampling_rate
            );
        delay -= jiffies % delay; // Sincronizacion de todos los
            trabajos

        dbs_info->enable = 1;
        INIT_DEFERRABLE_WORK(&dbs_info->work, do_dbs_timer);
        schedule_delayed_work_on(dbs_info->cpu, &dbs_info->work,
            delay);
}
```

5.5 PID

El cálculo del control PID es muy sencillo gracias a que el esquema básico ya lo tenemos creado. Lo que hacemos es obtener las variables mediante un *mutex*, para evitar la corrupción de datos debido a accesos simultáneos, calcular la acción del control, discretizar la frecuencia y aplicarla.

La acción de control se realiza igual a como lo hemos hecho previamente en espacio de usuario con la salvedad que en el kernel no se pueden usar números de coma flotante. Por ello debemos convertir la temperatura de miligrados a grados. Aunque a priori puede parecer que se pierde información, el sensor no es exacto y los saltos de temperatura son más altos de un grado. Para realizar rápidamente la división por mil, nos hemos ayudado con un número mágico que nos permite realizar la división usando una multiplicación[14].

```
long long int acum = (long long int) error * 0x418938;
int error = acum >> 32;
```

Para discretizar la frecuencia y a la vez seleccionarla para el procesador, utilizamos el método *__cpufreq_driver_target* al cual se le pasa la frecuencia y el método de discretización. En éste caso *CPUFREQ_RELATION_C* el cual selecciona la frecuencia más cercana tanto por lo alto como por lo bajo.

Pese a que cada CPU gestiona su frecuencia, las acciones de los procesadores son exactamente las mismas dado que comparten lectura de temperatura y el cálculo se realiza al mismo tiempo. Hacer que cada CPU realice una acción distinta en base a sus propios datos cambia por completo el modelo por lo que es excesivamente complejo para este trabajo.

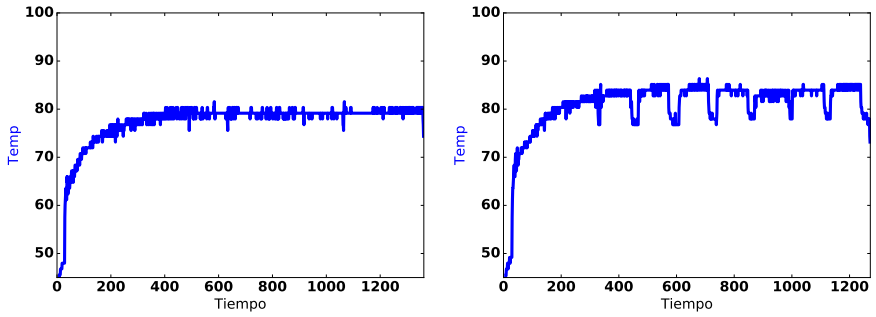
6

RESULTADOS

Vamos a pasar a probar como se comporta nuestro governor ya no solo centrándonos en cómo es el control, dado que ya sabemos que funciona correctamente tal y como hemos visto en el Cap. 4, sino haciendo el foco en el rendimiento que obtiene. Para comparar los distintos parámetros y el rendimiento con el governor por defecto, *ondemand*, vamos a almacenar los resultados del rendimiento en GFLOPS, la temperatura y frecuencia. Todo esto automatizado con un *script bash* el cual se explica su uso en Apéndice C.

6.1 PRUEBA DE PARÁMETROS

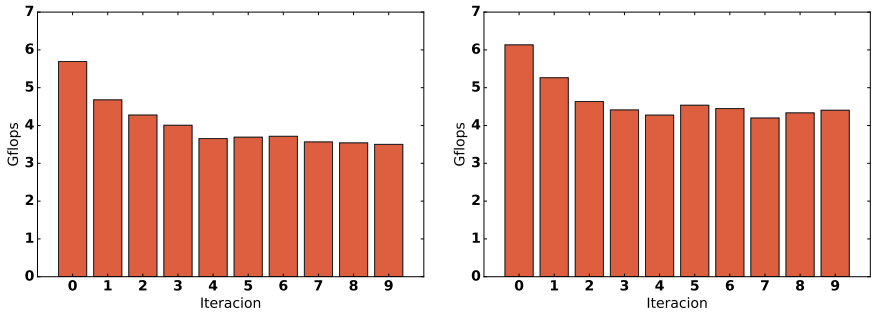
Primero vamos a comprobar el comportamiento del governor con distintos parámetros para buscar cuáles son los que mejor se comportan con nuestra placa, de estas pruebas solo realizamos una por cada configuración ya que se ha observado que múltiples ejecuciones obtienen los mismos resultados. Para las pruebas sabemos por los resultados previos que los mejores valores son, para el proporcional 100.000 y 75.000, y para el valor integral 90 y 180.



(a) Objetivo 80ºC (b) Objetivo 85ºC

Fig. 17: Comparación 80-85ºC K=75k Ti=90

Empezamos probando con un proporcional de 75.000 y un integral de 90, con diferentes temperaturas objetivo, 80 y 85ºC. Para 80ºC, se puede observar en la figura Fig. 17 que la temperatura alcanza el objetivo deseado y se mantiene constante, mientras que para 85ºC pese a que se alcanza la temperatura hay mucha variación. Sin embargo,

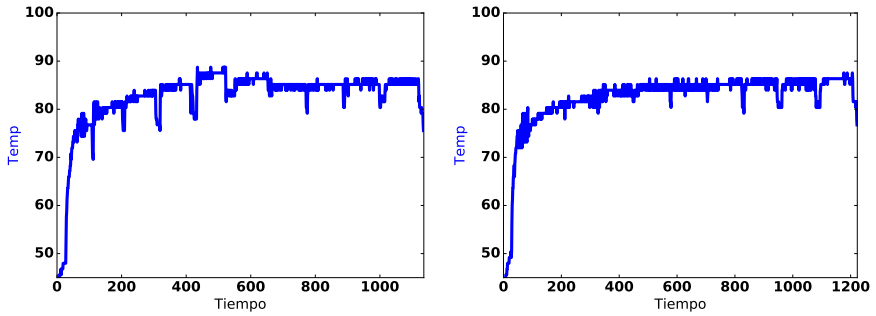


(a) Objetivo 80ºC (b) Objetivo 85ºC

Fig. 18: Comparación 80-85ºC K=75k Ti=90

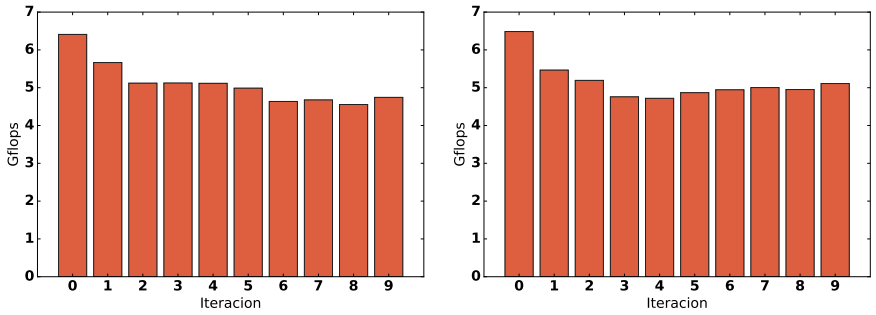
si nos fijamos en los resultados de rendimiento, figura Fig. 18, vemos como a 85ºC es cuando se obtienen los mejores resultados. Con un objetivo de 80°C se obtienen 4.03 GFLOPS de media mientras que a 85°C 4.66. Es decir, manteniendo al procesador lo más cerca de la temperatura máxima de funcionamiento obtenemos el mejor resultado.

Dado que incrementando la temperatura sabemos que podemos obtener unos mejores valores vamos a intentar mejorar el comportamiento a 85ºC con una K de 100.000 y variando el componente integral 90 y 180. Viendo las temperaturas registradas Fig. 19 podemos observar que el comportamiento es mucho más estable con una Ti de 180. Si nos fijamos en los rendimientos en la Fig. 20, podemos observar que pese a que con una Ti de 90 se obtiene un mejor rendimiento al inicio el comportamiento a largo plazo de la Ti 180 es mejor.



(a) Ti=90 (b) Ti=180

Fig. 19: Comparación Ti=90 y 180



(a) Ti=90 (b) Ti=180

Fig. 20: Comparación Ti=90 y 180

6.2 COMPARACIÓN RENDIMIENTO

Ahora vamos a pasar a comparar los rendimientos con el governor por defecto *ondemand*. A la hora de comparar ambos governors, realizamos test intercalados unos detrás de otro siempre dejando enfriar la placa apagada y el inicio de test nada más arrancarla. De este modo, dado que la temperatura exterior influye en los resultados, las pruebas son más justas.

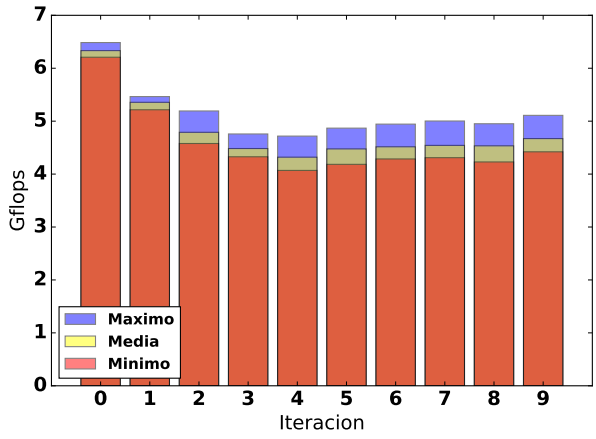


Fig. 21: Resultado ejecución K=100k Ti=180

Primero vamos a comprobar el rendimiento de varias ejecuciones del governor con la K a 100k y la Ti a 180 dado que hemos visto que son los valores que hacen que se mantenga mejor la temperatura. En la Fig. 21 tenemos el mejor resultado, el peor y la media de las 10 iteraciones de cada ejecución. Podemos observar como, tanto la primera como la segunda iteración son las más rápidas debido a que el procesador está frío y se empieza a calentar, luego, por la acción del PID el rendimiento baja al aproximarse a la temperatura para volver a incrementarse cuando la temperatura se estabiliza.

Tras ejecutar varios test con el governor por defecto lo podemos comparar con los resultados que tenemos. En la Fig. 22 tenemos en las barras de la izquierda los resultados de nuestro governor y en las de la derecha el de serie. Podemos observar que cuando el governor se acerca a la temperatura objetivo en las iteraciones 2, 3 y 4 disminuye el rendimiento. Esto puede deberse a que, como la temperatura sube muy rápido la acción se disminuye para no sobrepasar la temperatura objetivo, una vez estabilizada el PID incrementa la acción. Sin embargo, en el resto de la ejecución nuestro governor tiene un rendimiento ligeramente superior. Esto es debido a que el governor por defecto cambia muy rápido entre la mayor frecuencia y la menor de todas, mientras que el nuestro se mantiene en una frecuencia más constante Fig. 23. Recordemos que los cambios de frecuencia tienen un coste de tiempo en el que el procesador no puede hacer nada.

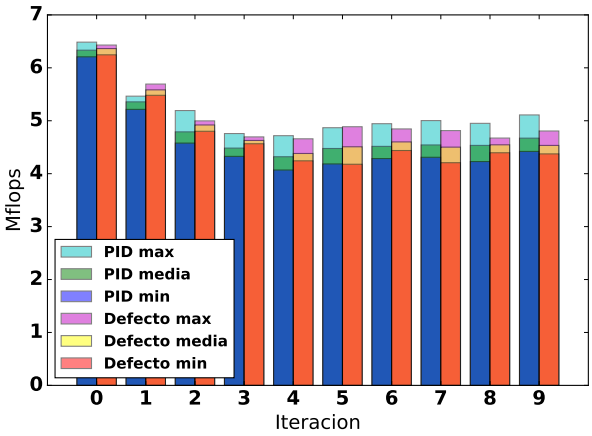
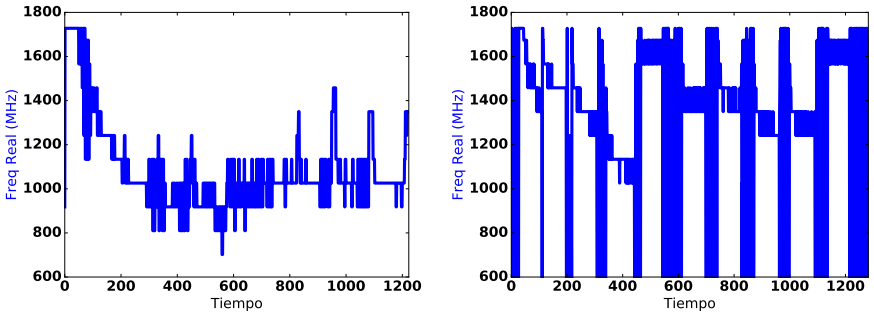


Fig. 22: Comparación K=100k Ti=180 con governor por defecto



(a) Frecuencias governor propio (b) Frecuencias governor defecto

Fig. 23: Comparación frecuencias governor

Como podemos ver en la comparación Fig. 22 el governor propio tiene mucha variación de rendimiento, esto puede ser debido a que toma valores de frecuencia más estables, mientras que el governor por defecto varía entre el máximo y el mínimo.

Para analizar esta variación del rendimiento, se ha calculado el coeficiente de variación de la distribución de tiempos de los 2 governors, el propio y el por defecto, *ondemand*. La Tabla 2 muestra la comparación tanto del cálculo total, como el coeficiente suavizado eliminando los extremos. Podemos ver como los valores son más dispares en el governor PID. Una posible razón es que el controlador PID después de la reducción de rendimiento al principio de la ejecución consigue recuperar algo del rendimiento, mientras el governor *ondemand* comienza a fluctuar entre frecuencias altas y bajas lo que hace que junto a lo uniforme de la carga de trabajo que el rendimiento sea más constante.

Tabla 2: Comparación Coeficiente de Variación

	PID	Ondemand
CV_{todos}	0.116	0.111
$CV_{sinextremos}$	0.104	0.100

Para analizar más en detalle este efecto, también se ha realizado la comparación de los percentiles de rendimiento. En la Tabla 3 podemos ver como, el governor propio obtiene mejor rendimiento en la mayoría de casos (percentiles 100%, 75% y 50%). Es decir, en un 50% de las iteraciones obtiene un valor mayor de GFLOPS, aunque tiene algunas iteraciones con menor rendimiento.

Tabla 3: Comparación percentiles de rendimiento

Percentil	PID	Ondemand
100%	6.48	6.43
75%	5.08	4.91
50%	4.69	4.66
25%	4.43	4.50
0%	4.06	4.17

7

CONCLUSIONES

A lo largo de este proyecto se ha realizado un modelado térmico de un procesador comercial, con el cual se ha podido crear un control PID con el que poder controlar la temperatura a la que funciona el procesador. Además éste controlador se ha implementado con éxito en un sistema operativo de gran extensión.

Este trabajo fin de grado combina conocimientos de 2 grandes áreas de trabajo, la Ingeniería Informática y la automática. Dentro de esta última se ha realizado un modelado térmico y un control PID los cuales se desconocían antes de realizar este trabajo. Esto ha servido por una parte para ampliar el área de conocimiento y adquirir experiencia en otras áreas, y por otro lado, la inclusión de otras áreas en un proyecto informático permite obtener mejores resultados dado que, se permiten incluir nuevas técnicas.

Todos los objetivos se han cumplido, tanto modelado térmico, como diseño de un control PID y en última instancia hemos logrado obtener un rendimiento que si bien, no es muy superior al por defecto, consigue su objetivo de mantener una temperatura constante con un rendimiento ligeramente superior.

Pese a que ya existía un sistema con la idea de control PID integrado para controlar la temperatura, el desarrollado aquí mantiene un control centralizado con una gestión precisa de la temperatura con un rendimiento excelente. El governor esta disponible en *github* [1] por lo que cualquiera puede probar y adaptar a su procesador.

Para un trabajo futuro quedaría modificar el governor para permitir que cada procesador lleve una gestión individual de su temperatura además de la búsqueda automática de las zonas de temperatura. Esto permitiría aplicar el governor en un servidor donde existen varios procesadores.

BIBLIOGRAFÍA

[1] *Controlador PID*. https://github.com/Pablololo12/governor-tfg/blob/master/governor/PID_governor.c.

[2] Murali Annavaram Ed Grochowski. *Energy per Instruction Trends in Intel® Microprocessors*. https://www3.intel.com/pressroom/kits/core2duo/pdf/epi-trends-final2.pdf. [Online; accedido 15-Junio-2017].

[3] *Estimating Transfer Function Models for a Heat Exchanger Example*. https://es.mathworks.com/help/ident/examples/estimating-transfer-function-models-for-a-heat-exchanger.html?prodcode=ID&language=en.

[4] *Estimating Transfer Function Models for a Heat Exchanger*. https://www.mathworks.com/examples/sysid/mw/ident_featured-ex61706059-estimating-transfer-function-models-for-a-heat-exchanger. [Online; accedido 6-Junio-2017].

[5] Gene F. Franklin, David J. Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. 4th. Prentice Hall PTR, 2001.

[6] Jonghwa Lee. *Thermal management using 'Generic Thermal FW'*. http://events.linuxfoundation.org/sites/events/files/lcjpcojp13_jonghwa.pdf. [Online; accedido 16-Junio-2017].

[7] *Programa Benchmark*. https://github.com/Pablololo12/governor-tfg/blob/master/benchmark/main.c.

[8] *Programa cálculo función de transferencia*. https://github.com/Pablololo12/governor-tfg/blob/master/Matlab_files/transferencia.m.

[9] *Script Python para mostrar gráficas*. https://github.com/Pablololo12/governor-tfg/blob/master/benchmark/plot_results.py.

[10] *Script bash para ejecutar tests*. https://github.com/Pablololo12/governor-tfg/blob/master/benchmark/test_governor.sh.

[11] *Snapdragon Math Libraries*. https://developer.qualcomm.com/software/snapdragon-math-libraries. [Online; accedido 6-Junio-2017].

[12] *The Linux Kernel Module Programming Guide*. http://www.tldp.org/LDP/lkmpg/2.6/html/.

[13] Xin Wang. *Intelligent Power Allocator*. http://infocenter.arm.com/help/topic/com.arm.doc.dto0052a/DTO0052A_intelligent_power_allocation_white_paper.pdf. [Online; accedido 6-Junio-2017].

[14] Henry S. Warren. *Hacker's Delight*. 2nd. Addison-Wesley Professional, 2012. ISBN: 0321842685, 9780321842688.

[15] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0321547748, 9780321547743.

LISTADO DE FIGURAS

Fig. 1 Esquema componentes del Kernel [6] 7
Fig. 2 Esquema de zonas térmicas 8
Fig. 3 Imagen de la plataforma 9
Fig. 4 Función de tranferencia 11
Fig. 5 Relación frecuencia temperatura de la ejecución continua 12
Fig. 6 Relación Temperatura real y modelada de la ejecución continua 13
Fig. 7 Resultado del modelo a 918 MHz 14
Fig. 8 Diagrama de un controlador PID 15
Fig. 9 Esquema diferencias PID en espacio de usuario y PID en kernel 17
Fig. 10 Gráficas de prueba con K=100k y 50k y objetivo de 80 grados 18
Fig. 11 Gráficas de prueba con K=75k y 50k y objetivo de 85 grados 19
Fig. 12 Gráficas de prueba con T=80 K=75K Ti=180 A=75000 B=-74583 C=0.0 E=-1.0 F=0.0 19
Fig. 13 Gráficas de prueba con T=80 K=100K Ti=180 A=100000 B=-99444 C=0.0 E=-1.0 F=0.0 20
Fig. 14 Gráficas de prueba con T=80 K=75K Ti=90 A=75000 B=-74167 C=0.0 E=-1.0 F=0.0 20
Fig. 15 Gráficas de prueba con T=80 K=100K Ti=90 A=100000 B=-98889 C=0.0 E=-1.0 F=0.0 21
Fig. 16 Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.5 A=200000 B=-396700 C=196710 E=-2.0 F=1.0 21
Fig. 17 Comparación 80-85ºC K=75k Ti=90 26
Fig. 18 Comparación 80-85ºC K=75k Ti=90 27
Fig. 19 Comparación Ti=90 y 180 27
Fig. 20 Comparación Ti=90 y 180 28
Fig. 21 Resultado ejecución K=100k Ti=180 28
Fig. 22 Comparación K=100k Ti=180 con governor por defecto 29
Fig. 23 Comparación frecuencias governor 29
Fig. 24 Diagrama de Gantt 38
Fig. 25 Gráficas de prueba con T=80 A=50K 39
Fig. 26 Gráficas de prueba con T=80 A=75K 39
Fig. 27 Gráficas de prueba con T=80 A=100K 39
Fig. 28 Gráficas de prueba con T=85 A=50K 40

Fig. 29 Gráficas de prueba con T=85 A=75K 40
Fig. 30 Gráficas de prueba con T=85 A=100K 40
Fig. 31 Gráficas de prueba con T=90 A=50K 40
Fig. 32 Gráficas de prueba con T=90 A=75K 41
Fig. 33 Gráficas de prueba con T=90 A=100K 41
Fig. 34 Gráficas de prueba con T=80 K=50K Ti=180 A=50000 B=-49722 C=0.0 E=-1.0 F=0.0 41
Fig. 35 Gráficas de prueba con T=80 K=75K Ti=180 A=75000 B=-74583 C=0.0 E=-1.0 F=0.0 42
Fig. 36 Gráficas de prueba con T=80 K=100K Ti=180 A=100000 B=-994444 C=0.0 E=-1.0 F=0.0 42
Fig. 37 Gráficas de prueba con T=85 K=100K Ti=180 A=100000 B=-994444 C=0.0 E=-1.0 F=0.0 42
Fig. 38 Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.5 A=200000 B=-397790 C=197790 E=-2.0 F=1.0 43
Fig. 39 Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.75 A=133330 B=-265190 C=131860 E=-2.0 F=1.0 43
Fig. 40 Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.9 A=111111 B=-220990 C=109880 E=-2.0 F=1.0 43
Fig. 41 Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.5 A=200000 B=-396700 C=196710 E=-2.0 F=1.0 44
Fig. 42 Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.75 A=23330 B=-264460 C=131130 E=-2.0 F=1.0 44
Fig. 43 Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.9 A=111110 B=-220380 C=109270 E=-2.0 F=1.0 44

LISTADO DE TABLAS

Tabla 1 Resultados del modelado 14
Tabla 2 Comparación Coeficiente de Variación 30
Tabla 3 Comparación percentiles de rendimiento . . . 30

GLOSARIO

BENCHMARK es un programa utilizado para medir el rendimiento de un sistema, como por ejemplo, velocidad de cálculo, velocidad de lectura de disco, velocidad de internet, ... 13

C se trata de un lenguaje de programación compilado muy utilizado para aplicaciones de bajo nivel. 18

DEEP LEARNING es un tipo de aprendizaje automático de ordenadores. 10

GFLOPS miles de millones de operaciones de coma flotante por segundo. 26, 27

GOVERNOR es un programa encargado de controlar unas determinadas variables en base a un control. 1–3, 8, 17, 18, 22–24, 26, 28–31, 45

PYTHON es un lenguaje de programación interpretado que permite crear de manera muy rápida y sencilla programas. 18

A

DIAGRAMA DE GANTT

El proyecto se dividió en las partes que se entendieron que eran necesarias para completarlo: trabajo previo, modelado, control PID, implementación Kernel y Memoria. Como además de ser un proyecto informático tiene partes de automática eran necesarios estudios previos para poder adquirir los conocimientos necesarios para poder desarrollar el proyecto.

Durante el trabajo previo se mantuvieron reuniones en las que se expusieron los problemas y los conocimientos a adquirir. El profesor Jose Maria Marín Herrero ayudó en esta tarea. Esta fase se alargo debido a que se comenzó el proyecto con otra plataforma hasta que se descubrió que solo permitía elegir 2 frecuencias.

El modelado fue muy costoso dado que hubo que realizar muchas pruebas hasta dar con el correcto y no se estaba acostumbrado a esa clase de herramientas de modelado.

El desarrollo y pruebas del control PID fue sencillo. Dado que las pruebas se automatizaron y el primer control se realizó en espacio de usuario, era fácil comprobar el funcionamiento.

Por acercarse la fecha de entrega, la implementación en *kernel* y la memoria se hicieron paralelamente. Pero, la implementación resultó extremadamente difícil y se tomó la decisión de postergar la entrega de la memoria a finales del mes de junio. Gracias a esto se pudieron realizar más pruebas de las que se hubieran realizado de otro modo.

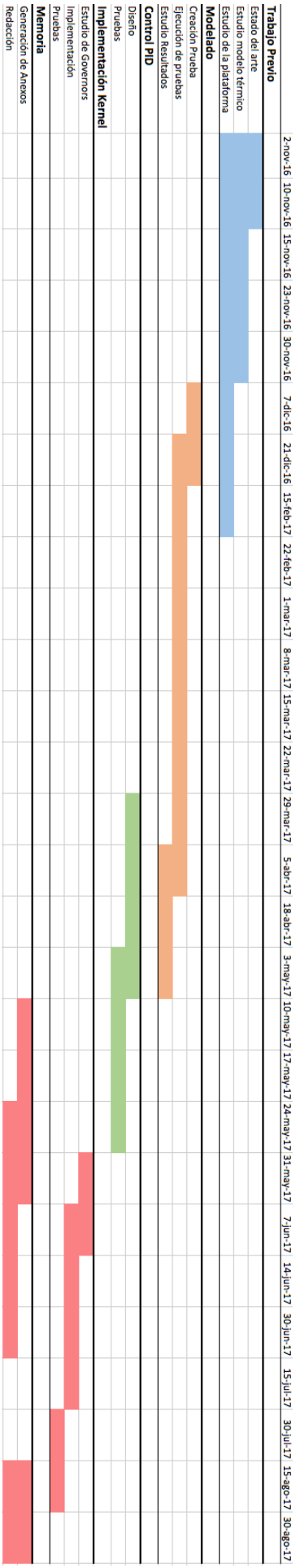
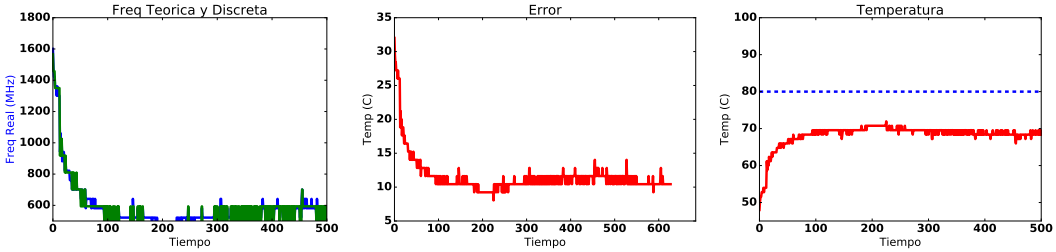


Fig. 24: Diagrama de Gantt

B

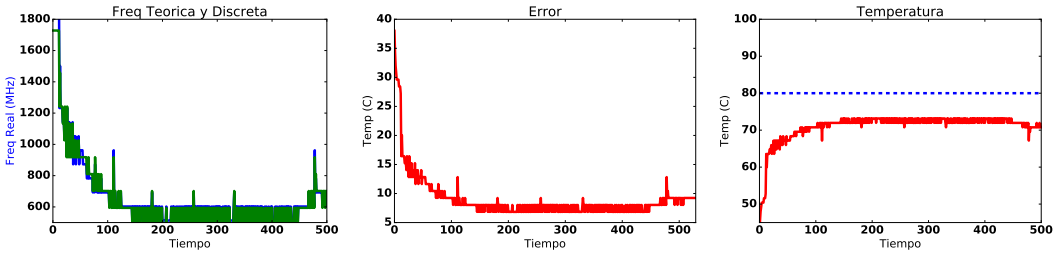
PRUEBAS PID

B.1 PRUEBAS SOLO PROPORCIONAL



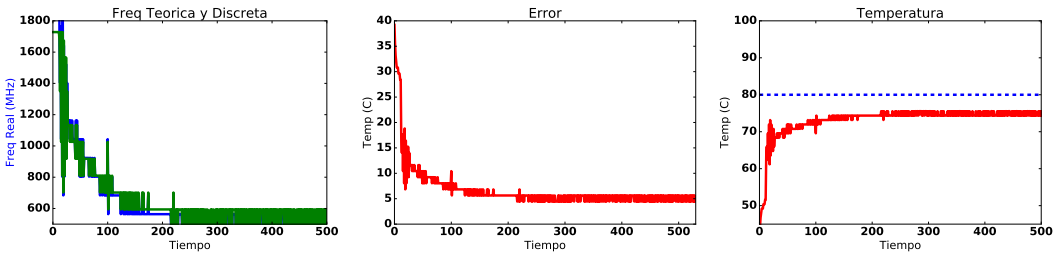
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 25: Gráficas de prueba con T=80 A=50K



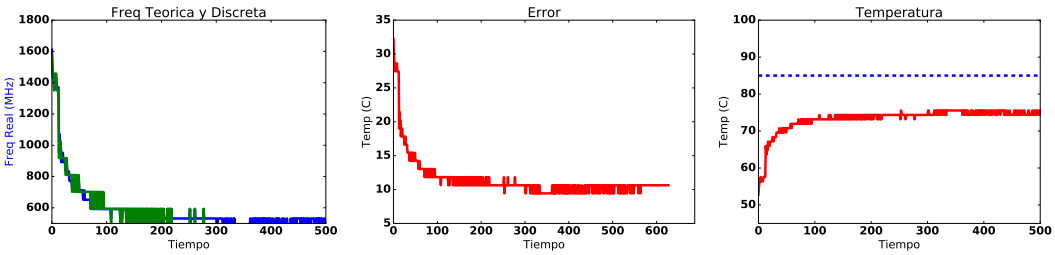
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 26: Gráficas de prueba con T=80 A=75K



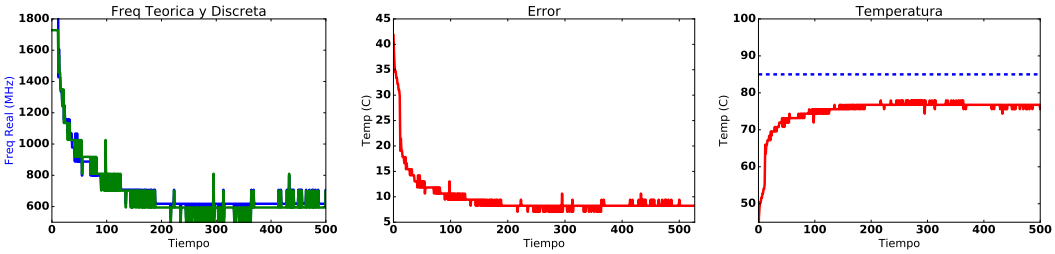
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 27: Gráficas de prueba con T=80 A=100K



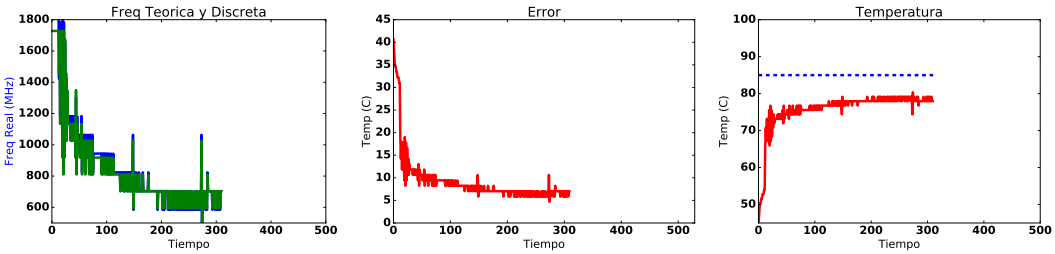
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 28: Gráficas de prueba con T=85 A=50K



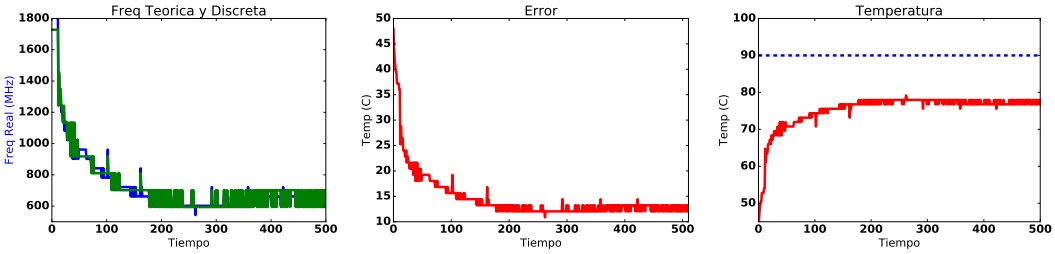
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 29: Gráficas de prueba con T=85 A=75K



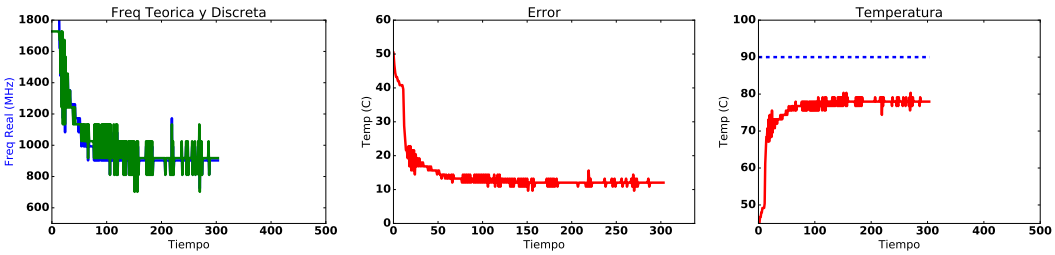
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 30: Gráficas de prueba con T=85 A=100K



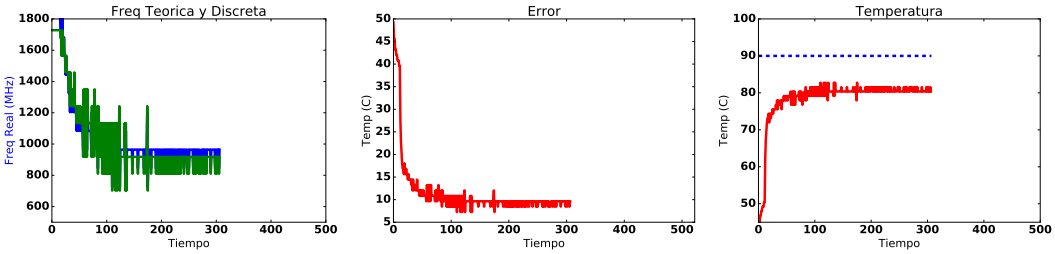
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 31: Gráficas de prueba con T=90 A=50K



(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 32: Gráficas de prueba con T=90 A=75K

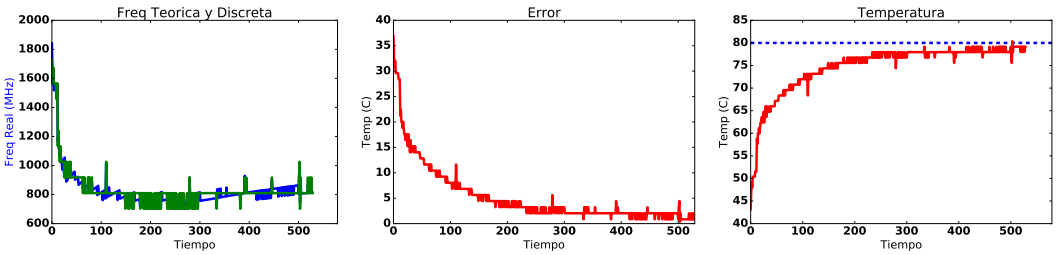


(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 33: Gráficas de prueba con T=90 A=100K

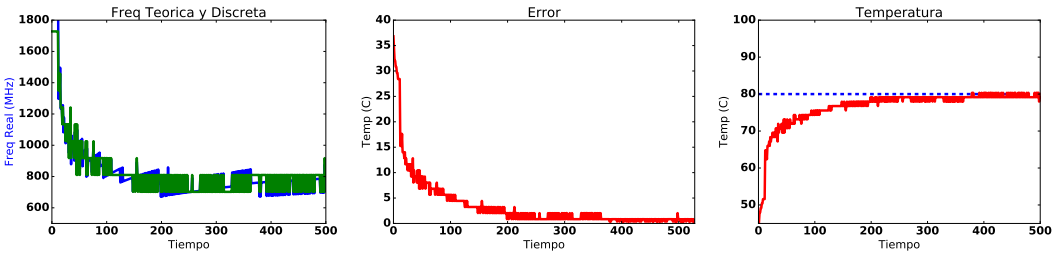
B.2 PRUEBAS PROPORCIONAL E INTEGRAL

Los siguientes resultados se obtienen añadiendo el control integral.



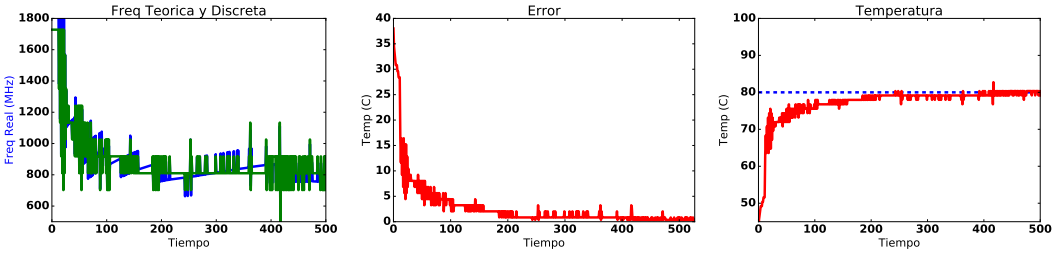
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 34: Gráficas de prueba con T=80 K=50K Ti=180 A=50000 B=-49722 C=0.0 E=-1.0 F=0.0



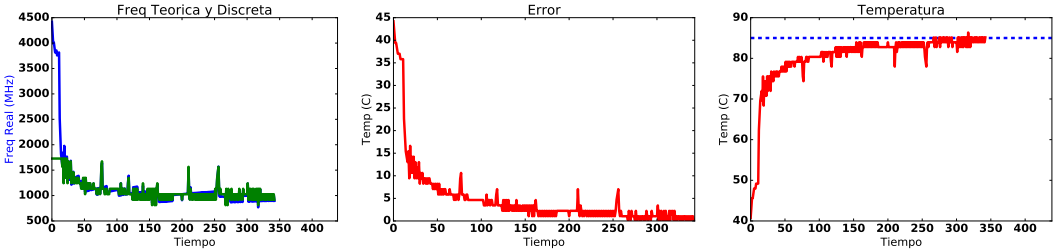
(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

Fig. 35: Gráficas de prueba con T=80 K=75K Ti=180 A=75000 B=-74583 C=0.0 E=-1.0 F=0.0



(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

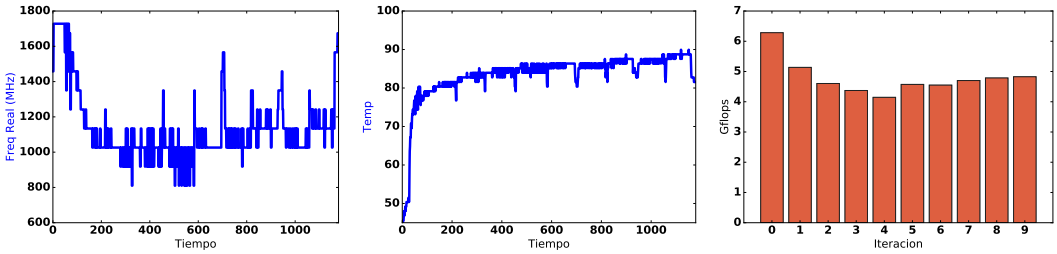
Fig. 36: Gráficas de prueba con T=80 K=100K Ti=180 A=100000 B=-994444 C=0.0 E=-1.0 F=0.0



(a) Gráfica de frecuencia (b) Gráfica de error (c) Gráfica de temperatura

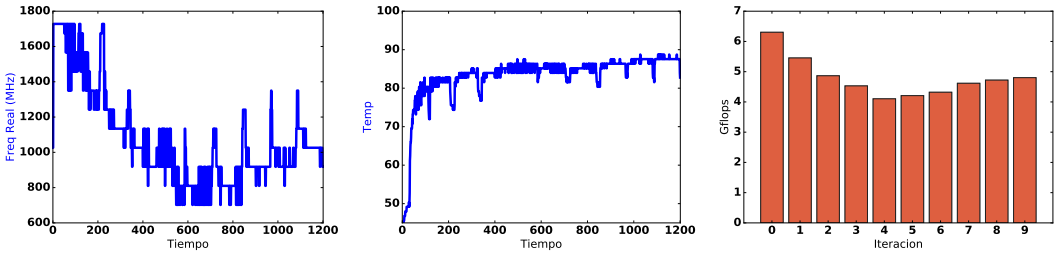
Fig. 37: Gráficas de prueba con T=85 K=100K Ti=180 A=100000 B=-994444 C=0.0 E=-1.0 F=0.0

B.3 PRUEBAS GOVERNOR PROPORCIONAL, INTEGRAL Y DERIVATIVO



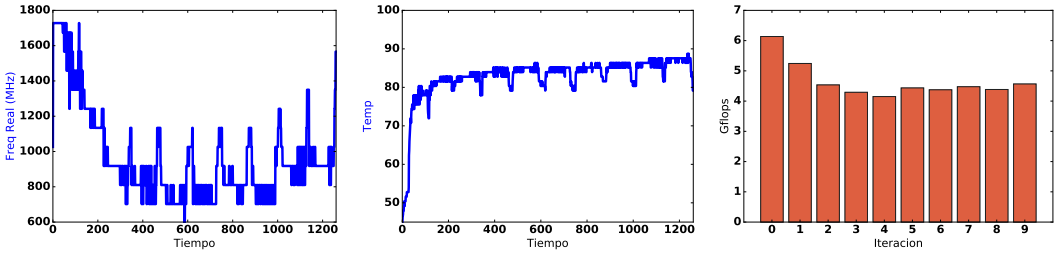
(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 38: Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.5 A=200000 B=-397790 C=197790 E=-2.0 F=1.0



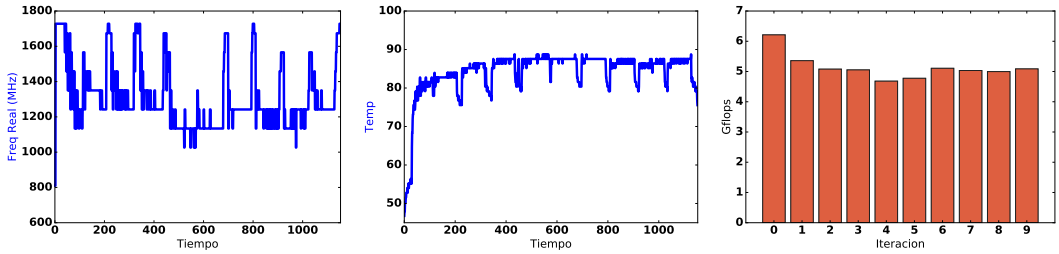
(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 39: Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.75 A=133330 B=-265190 C=131860 E=-2.0 F=1.0



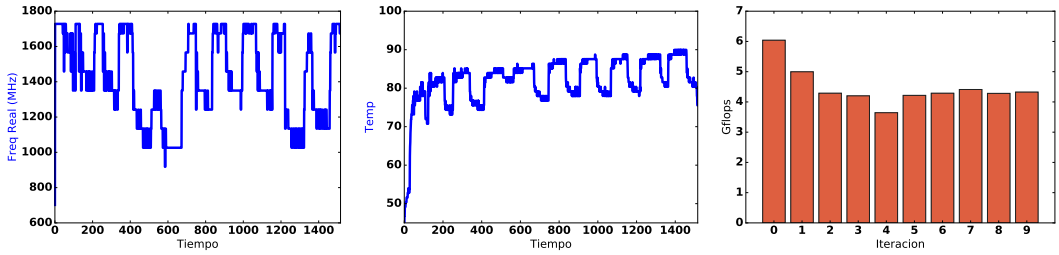
(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 40: Gráficas de prueba con T=85 K=100K Ti=180 Td=180 α=0.9 A=111111 B=-220990 C=109880 E=-2.0 F=1.0



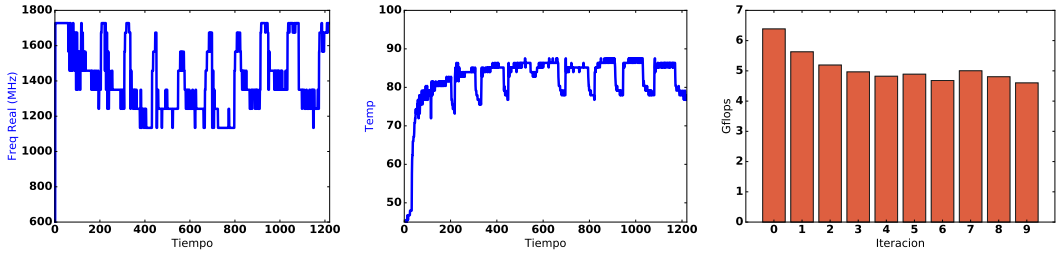
(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 41: Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.5 A=200000 B=-396700 C=196710 E=-2.0 F=1.0



(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 42: Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.75 A=23330 B=-264460 C=131130 E=-2.0 F=1.0



(a) Gráfica de frecuencia (b) Gráfica de Temperatura (c) Resultado benchmark

Fig. 43: Gráficas de prueba con T=85 K=100K Ti=180 Td=90 α=0.9 A=111110 B=-220380 C=109270 E=-2.0 F=1.0

C

COMO USAR PROGRAMA PRUEBAS

A la hora de probar el governor en *kernel* se ha creado un script en bash que automatiza la tarea [10]. Este se encarga de desactivar los cores, modifica si es necesario el governor y lo configura y además, se encarga de ejecutar el programa de pruebas y registrar tanto la temperatura como la frecuencia. Este programa se puede usar para probar tanto el governor por defecto como el diseñado aquí.

Para el governor por defecto basta con ejecutar el script sin parámetros.

Si queremos probar el *PID_governor* se deben pasar 6 parámetros. A, B, C, E, y F que son los valores que se usan para el PID discreto. Además de pasar la temperatura objetivo en miligrados.

A lo largo de la ejecución el programaste nos mostrara los pasos que toma y si sucede algún error. Cuando termine tendremos 3 ficheros:

- *temps_log.txt* el cual almacena las temperaturas y las frecuencias
- *bench_result.txt* los resultados de la ejecución
- *log.txt* la salida estándar del benchmark