Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Alejandro Nortes Pollos

Para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Power modeling with performance monitoring counters of a Zynq platform.

FEBRERO 2019

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE ECUACIONES

SIGLAS Y ACRÓNIMOS

- **PMC:** Performance monitoring counters.
- **DVFS:** Dinamic voltage and frequency scaling.
- **RISC:** Set de instrucciones reducido.
- **PMU:** Performance monitoring unit.
- **PLE:** Preload engine.
- **PL:** Programmable logic.
- **PS:** Processing System.
- **I/O:** Entrada salida.
- **TTC:** Triple timer counter.
- **Bsp:** Board support package.
- **ALU**: UNIDAD ARITMÉTICO LÓGICA.
- **SoC (System on a Chip)**

1 INTRODUCCIÓN

1.1 Datos Generales

Título:	Power modeling with performance monitoring counters of a Zynq platform.
Autor:	Alejandro Nortes Pollos
Datos autor:	5357461-Y uo231069@uniovi.es
Empresa:	IKERLAN,S.Cop
Tutor empresa:	Tomaso Poggi
Tutor académico:	José Antonio Cancelas Caso
Fecha inicio	Junio 2018

Tabla 1-1 Datos Generales del proyecto

IKERLAN es un Centro Tecnológico que busca la excelencia en la I+D+I, gracias a la continua adaptación a las necesidades de sus clientes y a la cercanía a la realidad empresarial (es una cooperativa miembro de la Corporación MONDRAGON). Fiel a su misión, trabaja día a día desde 1974 para desarrollar soluciones que permitan a sus clientes ser cada vez más competitivos.

Gracias a un modelo único de colaboración, que combina actividades de transferencia tecnológica, investigación propia y formación de personal altamente especializado, IKERLAN es a día de hoy el *partner* tecnológico de confianza de importantes empresas del país.

Para ello, IKERLAN está estructurado en tres unidades de especialización tecnológicas:

- TECNOLOGÍAS DE ELECTRÓNICA, INFORMACIÓN Y COMUNICACIÓN
- ENERGÍA Y ELECTRÓNICA DE POTENCIA
- FABRICACIÓN AVANZADA

El proyecto se está realizando en la unidad de tecnologías de electrónica, información y comunicación. Esta unidad está formada por áreas y estas a su vez en equipos. El área donde se está realizando este proyecto es la de sistemas embebidos confiables y el equipo es el de sistemas de tiempo real.

1.2 Resumen

El consumo de potencia es una de las principales restricciones en el diseño de procesadores, ya que características cruciales como el tiempo de vida, la fiabilidad y la frecuencia de operación dependen de la potencia consumida. Aparte, regular y reducir la potencia de los procesadores permite aumentar la autonomía de dispositivos remotos y mejorar el rendimiento de los centros de supercomputación.

Como indica [1], existen diversas técnicas para monitorizar y/o controlar el consumo de un procesador, tales como:

- DVFS (Escalado dinámico de voltaje y frecuencia): El DVFS es una técnica que nos permite controlar el voltaje y frecuencia del procesador para regular la potencia consumida por este.
- Power Models: Se construye un modelo, utilizando parámetros del procesador, que nos permita estimar el consumo de potencia.
- Clock gating: Esta técnica consiste en apagar la señal de reloj a ciertos componentes del procesador, cuando no son utilizados.
- Se pueden utilizar distintas técnicas para ahorrar energía en diversos componentes del procesador, como realizar una correcta organización de la memoria.

En este proyecto, se ha centrado la investigación en el diseño, implementación y validación de un modelo de potencia. Los parámetros del modelo serán los Performance Monitoring Counters (PMC), una serie de contadores que registran ciertos eventos hardware del procesador, como número de ciclos de reloj, número de instrucciones ejecutadas, acceso a la cache[2]. Se han escogido dichos parámetros para nuestro modelo, por los siguientes motivos:

- Varios de los eventos hardware que registran los PMC, están altamente relacionados con el consumo de potencia.
- Ya ha habido varios ensayos y proyectos que han usado esta técnica y han dado resultados positivos [3, 4] [5, 6].
- Los PMC pueden ser medidos de forma interna por el procesador, lo que simplifica la medida de estos y el error que puede producirse en la medida de los mismos.

El diseño del modelo se basará en la estrategia usada en [3, 4] [5, 6]. El modelo será basado en una combinación lineal de PMC, ponderados por coeficientes obtenidos mediante el método de mínimos cuadrados. Para la validación, se mirará el error medio del modelo, su sensibilidad, que es la capacidad que tiene el modelo en detectar un cambio significativo en el consumo de potencia.

Una vez se conozca cuales son los PMC y como se va a diseñar el modelo sobre el consumo de potencia, se deben realizar las siguientes actividades:

- Buscar y/o diseñar e implementar benchmarks que estresen la CPU y nos permitan tener distintas medidas sobre el consumo de potencia y los PMC.
- Diseñar e implementar un algoritmo que ejecute los benchmarks de manera secuencial y que registre la potencia consumida y el valor de los PMC del procesador en cada benchmark ejecutado.
- Elegir que PMC se utilizarán en el modelo. Una vez escogidos los PMC, diseñar y validar el modelo que nos permita estimar el consumo de potencia con los datos obtenidos en el apartado anterior.
- Implementar dicho modelo en el controlador utilizado y generar de manera automática informes sobre el consumo de potencia (*profiling*).

En este proyecto, el hardware a utilizar es la placa Zynq ZC702, de Xilinx, que tiene integrado el procesador ARM Cortex-A9. Esta placa se ha elegido porque no existe literatura en la que se diseñe un modelo de potencia basado en PMC, usando este hardware y sin utilizar un sistema

operativo. Además, estas placas comerciales disponen del hardware necesario para medir la potencia consumida y monitorizar los PMC.

1.3 Alcance y objetivo

El objetivo principal de este proyecto es la construcción de un modelo, basado en performance monitoring counters, que permita estimar el consumo del procesador ARM Cortex-A9. Como plataforma de trabajo se usará la placa ZC7002 de Xilinx, que tiene dicho procesador integrado en la placa. En concreto en el System-on-Chip (SoC) Zynq 7000 de Xilinx.

Para conseguir este objetivo se deben realizar los siguientes objetivos:

- Estudio del arte sobre los performance monitoring counters, la arquitectura ARM y sobre la PCB Zynq ZC702.
- Búsqueda y/o diseño e implementación de benchmarks para testear la CPU con la que se trabaja.
- Selección de los PMC a usar en el modelo a implementar.
- Diseño matemático de un conjunto de modelos que nos permitan estimar la potencia de uno o más cores de la CPU.
- Diseñar e implementar un conjunto de algoritmos que realice las siguientes tareas: Ejecutar, de manera secuencial, los benchmarks seleccionados; mientras tanto, simultáneamente realizar, periódicamente, una adquisición de datos de la CPU (Performance monitoring counters, voltaje, temperatura, potencia consumida, tiempo de muestreo) y luego almacenar dichos datos en un fichero de texto.
- Utilizando los datos generados por el algoritmo, implementar los diferentes modelos de potencia y evaluar su sensibilidad y precisión.
- Implementación de dichos modelos de potencia en la placa de trabajo y generación automática de informes sobre el consumo de potencia.

2 ESTADO DEL ARTE

El consumo de potencia es un factor muy importante en el diseño de cualquier elemento con capacidad de procesamiento. Ya que al optimizarlo los equipos se calentarán menos, la vida útil de los mismos se alargará y tendrán un mejor rendimiento.

Al crear un modelo que nos permita estimar el consumo de potencia, se pueden implementar técnicas que permitan una mayor eficiencia energética, esto es muy beneficioso para:

- Los dispositivos inalámbricos que dependen de una batería para funcionar, ya que si se puede regular el consumo de potencia de estos dispositivos, se podrá aumentar la autonomía de los mismos y/o reducir la carga de la batería.
- Los centros de supercomputación o granjas de servidores son instalaciones con una gran capacidad de procesamiento y con mucho consumo de potencia. Lo que propicia que dichos procesadores se calienten mucho. En consecuencia, se necesita de un sistema de refrigeración bastante voluminoso y costoso. Por ende, es casi obligatorio disponer de métodos que nos permitan regular y monitorizar el consumo de potencia, mejorando el rendimiento y reduciendo costes en el sistema de refrigeración y la energía consumida.

Además, debido al avance tecnológico y el proceso de globalización, el número de terminales con un sistema de procesamiento está aumentado de manera exponencial, lo que significa que se va a necesitar más energía eléctrica para alimentar dichos dispositivos. En consecuencia, es inevitable que estos dispositivos tengan sistemas para controlar la potencia gastada, evitando que el consumo de energía eléctrica se dispare, así como los efectos medioambientales que la generación de esta pueda ocasionar.

2.1 Power modeling y PMC

El Power *M*odeling consiste en diseñar un modelo matemático que permita estimar el consumo de potencia. La cantidad de modelos es muy diversa, ya que depende del tipo de datos que se usen para el modelo y las diferentes técnicas o herramientas a utilizar para construir el modelo. Por ejemplo, el modelo de potencia puede ser una combinación lineal, cuyos coeficientes se obtengan mediante el método de mínimos cuadrados, del descenso del gradiente o de manera analítica. Aparte, se puede recurrir a técnicas de machine learning o deep learning para obtener dicho modelo. Las metodologías para generar este tipo de modelos, se pueden dividir en dos grandes grupos: “bottom-up” y “top-down”.

2.1.1 Modelos Bottom-up

Tal y como explica [4] y [7], los modelos “Bottom-Up” tienden hacia un enfoque analítico, ya que estudian a fondo la microarquitectura de la CPU, cogen una de sus especificaciones (número de pipelines, tamaño de la cache, …) y de manera secuencial, se mira la actividad de dicha área y su consumo de potencia. Este tipo de modelos pueden usar como parámetros del modelo los PMC u otro tipo de parámetros. Como cualquier metodología, esta contiene una serie de ventajas e inconvenientes.

Las ventajas son que de esta manera es muy fácil desacoplar el consumo de las distintas áreas de la CPU, ocasionando que el modelo posea una granularidad fina y permitiendo que el modelo se pueda generalizar a más cargas. Además, la "responsivness", es decir la capacidad del modelo de detectar cambios en el consumo de potencia, y la precisión del modelo son bastante altas.

Por otro lado, el tiempo de desarrollo en este tipo de modelos es muy alto, debido al tiempo requerido para analizar la arquitectura del procesador y el diseño de microbenchmarks que testeen cada componente de la CPU. Además, una vez implementado dicho modelo en la CPU, los modelos de "Botttom-Up" tardan más tiempo en estimar la potencia que los modelos "Top-Down".

Debido a que realizar este tipo de modelos, puede llevar un coste y tiempo de desarrollo muy altos, se han creado simuladores que permiten simular el comportamiento de las CPUs, ayudando a que el tiempo de desarrollo se reduzca. Un ejemplo de esto es McPAT, un entorno que permite modelar el comportamiento temporal y energético de un procesador de forma simultánea [8].

2.1.2 Modelos Top-Down

Los modelos top-Down utilizan un enfoque más empírico, ya que dada una CPU, se ejecutan una serie de benchmarks, se escogen una serie de parámetros y se mide, a través de dispositivos internos o externos, la potencia consumida por cada benchmark. Los parámetros recogidos suelen ser PMC y se suelen utilizar para construir el modelo que permita estimar la potencia de dicha CPU [3-7] .

Las ventajas de este tipo de modelos, es que su tiempo de desarrollo es mucho más bajo, debido a que no es necesario conocer en detalle la arquitectura de la CPU y una vez implementado el modelo estiman la potencia consumida mucho más rápido que los modelos "Bottom-Up".

En cambio, los modelos "Top-Down" funcionan como una caja negra. Es decir, permiten estimar el consumo de la CPU, pero no permiten desacoplar el consumo en sus distintas áreas, lo que provoca que la granularidad de estos modelos sea mucho más gruesa que la de los modelos "Bottom-Up". Además, este tipo de modelos suelen tener menor precisión y "responsivness", aunque siguen siendo dando resultados bastante buenos [3-6], que los modelos de "Bottom-Up". Aparte, este tipo de modelos no es tan generalizable y en algunos tipos de tareas no estima tan bien la potencia como un modelo "Bottom-up".

Al final, optar por un modelo "Bottom-up" o "Top-Down", dependerá de las especificaciones que tenga el proyecto a realizar, ya que cada metodología tiene unas ventajas e inconvenientes. En la Tabla 2-1 se muestra un resumen de las capacidades de ambos modelos, basado en la tabla 2 del documento [4].

Metodología a usar	Precisión	Responsiveness	Tiempo de desarrollo	Generalización a otros benchmarks	Desacoplamiento de la potencia	Tiempo en estimar la potencia
Bottom-Up	A	A	A	A	A	A
Top-Down	N	N	B	N	N/A	B

Tabla 2-1 Resumen general de ambas metodologías. A: Alto, N: Normal, B: Bajo, N/A: No aplicable.

2.1.3 Regresión lineal

Como se explica en [5, 6] y [3, 4], para obtener el modelo de potencia se suele utilizar una regresión lineal y la ecuación que define el modelo suele ser de las que se muestra en la Ecuación 2-1:

$$P = \left(\sum_{i=0}^{n} \frac{PMC_i}{T_s} * K \ * a_i\right) + f(Vdd, fclk) + f(T)$$

Ecuación 2-1 Ecuación generalmente usada para estimar la potencia.de un core

Como se puede ver en la Ecuación 2-1, el primer termino suele ser un sumatorio de los PMC elegidos ocurridos en el tiempo de muestreo T_s, elegidos y multiplicados por unas constantes K, que suelen ser unos términos referidos a la tensión de alimentación (Vdd) y/o la frecuencia de la CPU (fclk) o una combinación de ambas. Por último a_i son los coeficientes del modelo a hallar.

Es necesario que los PMC sean divididos por el tiempo de muestreo, ya que los PMC son una medida de energía y el modelo debe estimar la potencia. Por otro lado, el segundo término hace referencia al consumo que tiene la CPU simplemente por estar encendida y el tercero tiene en cuenta la temperatura. La Ecuación 2-1 permite estimar la potencia, sólo para un core de la CPU. Eso quiere decir que la Ecuación 2-1 no puede estimar la potencia de varios núcleos de una CPU. Si se quiere estimar la potencia en dos o más cores de la CPU, se debe identificar primero los coeficientes del modelo para un solo core en la Ecuación 2-1 y luego aplicar la Ecuación 2-2 para estimar la potencia en varios cores, usando los coeficientes obtenidos en la Ecuación 2-1.

$$P = \sum_{j=0}^{m} \left(\sum_{i=0}^{n} \frac{PMC_{ji}}{T_s} * K \ * a_{ji}\right) + \alpha(f(Vdd, flck) + f(T))$$

Ecuación 2-2 Ecuación para estimar la potencia consumida de m cores.

Como se puede apreciar en la Ecuación 2-2, para estimar la potencia de dos o más cores se estima, por una parte, la potencia consumida en cada core a través del valor de sus respectivos PMC y, por otra, el consumo inherente de la CPU por estar encendida y funcionado. En la Ecuación 2-2, el último termino está multiplicado por un coeficiente α que siempre es mayor que cero. Además, a priori no se puede determinar cuánto, ni en qué proporción va a aumentar el consumo de potencia por estar dichos núcleos activos. Por lo tanto, para que este modelo de buenos resultados, se debe optimizar el valor de α para reducir al mínimo el error del modelo. Aparte, ese valor de α variará, si se cambian el número de núcleos a utilizar o el tipo de PMC a utilizar.

3 HERRAMIENTAS UTILIZADAS

En este aparatado, se explicará el hardware y software utilizado para la realización del proyecto.

3.1 Cortex A9-r2p2

ARM define una arquitectura de instrucciones reducidas (RISC) implementadas en procesadores. Alguno de los tipos de procesadores que se fabrican con la arquitectura ARM son:

- Cortex-A: Estos procesadores ofrecen el máximo cociente entre trabajo y energía consumida que priman este tipo de característica sobre todo. Éstos son utilizados en el entorno de la automoción, medicina y almacenamiento.
- Cortex-R: Este tipo de procesadores suelen ser utilizados para aplicaciones que necesitan procesamiento en tiempo real. Éstos son utilizados en el entorno de la automoción, medicina y de cámaras.
- Cortex-M: Son procesadores de bajo coste, que priman la eficiencia energética ante todo lo demás. Los ámbitos donde estos procesadores son usados es en el sector de automoción, medico y aplicaciones de sistemas embebidos.

Para este proyecto, se va a utilizar el procesador Cortex A9-r2p2, que es un procesador de la categoría A.

Tal y como explica [2], el Cortex A9-r2p2 es un procesador de alto rendimiento (o prestaciones) y bajo consumo y con un subsistema L1 de cache, que permite que haya una capacidad de memoria virtual. Este procesador tiene una arquitectura ARMv7-A y corre instrucciones de 32 bits de ARM, instrucciones de 16 bits de Thumb, las cuales son un subconjunto de instrucciones equivalentes a las instrucciones de 32 bits de ARM. También dispone de instrucciones de 8 bits-Java en el estado de Jazelle, el cuál es una extensión que nos permite ejecutar código Java de forma directa en el hardware.

Dicho procesador puede trabajar con un solo core o con varios a la vez. En la Ilustración 3-1, se puede ver un diagrama de alto nivel con los procesos de la CPU.

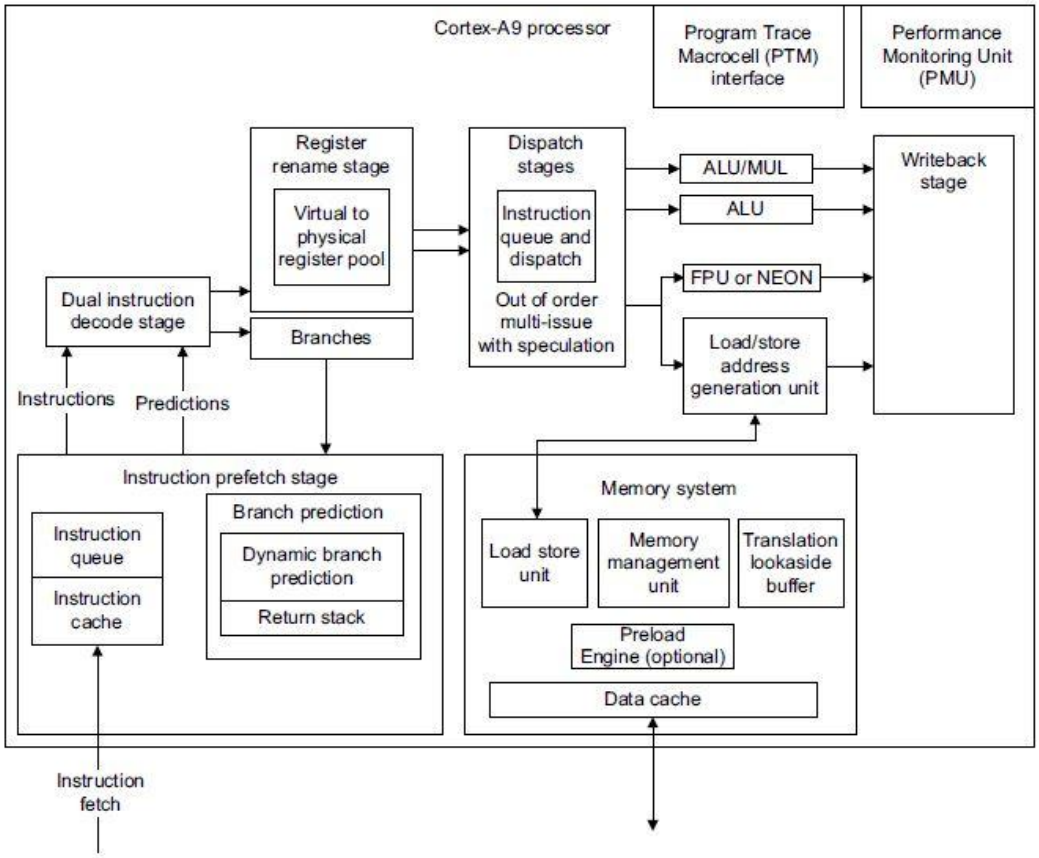


Ilustración 3-1 Diagrama de alto nivel del Cortex-A9 de [2]

Ahora se procederá a explicar algunas de las diferentes áreas de la CPU que son usadas en este proyecto [2]; si requiere más información consulte [2].

3.1.1.1 Performance Monitoring Unit (PMU)

Esta unidad se encarga de monitorizar el rendimiento de la CPU. Para ello, existen los Performance Monitoring Counters. Una serie de contadores que aumentan su valor cuando detectan ciertos eventos hardware (por ejemplo instrucciones ejecutadas, acceso a la cache, nº ciclos de reloj). Este procesador puede detectar 57 eventos hardware. Respecto a los PMC, hay dos observaciones importantes:

- Cada núcleo del procesador tiene su propio set independiente de PMC. Esto significa que no se pueden contar los ciclos de reloj del core 0 con el contador de ciclos de reloj del core 1 y viceversa.
- Cada core solo puede monitorizar 6 de esos contadores de manera simultánea.

Si se mira en detalle la información sobre la PMU en [2], se puede observar que los contadores pueden agruparse en varías categorías, que se explicarán ahora:

- **Ciclos de reloj:** Estos contadores están relacionados con los ciclos de reloj del procesador.
- **Cache:** Estos contadores indican eventos hardware relacionados con la memoria cache (acceso, refill, unaligned acess).

- **Unidad de Load/Store:** Estos contadores registran eventos afectados con la Load/Store Unit, ver Ilustración 3-1. Esta unidad se encarga de manejar todas las operaciones de carga y almacenamiento y se encarga de desacoplar las instrucciones de carga y descarga de la tubería principal y la ALU.
- **Branch prediction:** Todos los contadores que detecten eventos concernientes a la unidad de branch prediction, ver Ilustración 3-1. Este módulo se encarga de predecir la rama, como si fuera un *if* en C, por la cual irá el set de instrucciones. Reduciendo el tiempo que tarda el flujo de instrucciones en ejecutarse.
- **Instrucciones ejecutadas:** Este tipo de eventos hardware permite saber el número de instrucciones ejecutadas por la CPU.
- **Instrucciones ejecutadas en la ALU:** Este tipo de contadores registran las operaciones especiales como las operaciones de tipo entero o flotante.
- **Instrucciones en Java:** Cuenta el número de eventos que se han realizado utilizando las instrucciones de Java del procesador.
- **Instrucciones sobre PLE:** Registra la cantidad de veces que ha ocurrido algo con el PLE (Preload Engine [2]). El cuál es un motor que carga ciertas regiones de la memoria del programa en la cache de la CPU.

3.2 Zynq ZC702

La placa ZC702 es una placa de evaluación y se ha seleccionado por varios motivos. Primero, en la empresa hay stock de estas placas y se ha trabajado anteriormente con ellas. Por lo tanto, es fácil que se reduzca el tiempo de programación y configuración en estas placas. Segundo, la placa dispone de un sistema para monitorizar diversas variables internas (voltaje, temperatura, potencia). Esto nos permite prescindir de un sensor externo para medir el consumo de potencia, reduciendo costes y reduciendo los errores y/o interferencias que puede ocasionar conectar un sensor externo a la placa.

Por otro lado, esta placa posee características muy comunes entre componentes de sistemas embebidos, incluyendo una RAM DD3 de 1Gb, puertos de entrada salida (I/O) de propósito general, varios tipos de comunicación (I2C, CAN, Ethernet, USB), convertidor AD, sistemas para almacenar datos (SD, EEPROM) e interfaces UART y HDMI. Además, dispone de un Zynq-7000 SoC de Xilinx, con dos componentes principales:

- PS (Processing system): Este módulo contiene la unidad de procesamiento de la placa, la cual puede ser programada en lenguajes de alto nivel (C, C++).
- PL (Programmable Logic): Este módulo contiene lógica programable del mismo tipo que se encuentra en las FGPA**s**.

Ambos módulos pueden usarse a la vez o por separado. Para este proyecto, solo va a ser necesario usar la PS, que está constituida por un procesador ARM Cortex A9-r2p2.

A continuación, en la Ilustración 3-2 e Ilustración 3-3 serán mostrados un diagrama de bloques de los diferentes módulos de la PCB y la forma física de esta misma, respectivamente [9]. Por otro lado, en la Ilustración 3-4 se mostrará un diagrama de alto nivel del SoC y en la Ilustración 3-5 se ilustrará en más detalle los diferentes bloques de los que está compuesto el SoC de la placa.

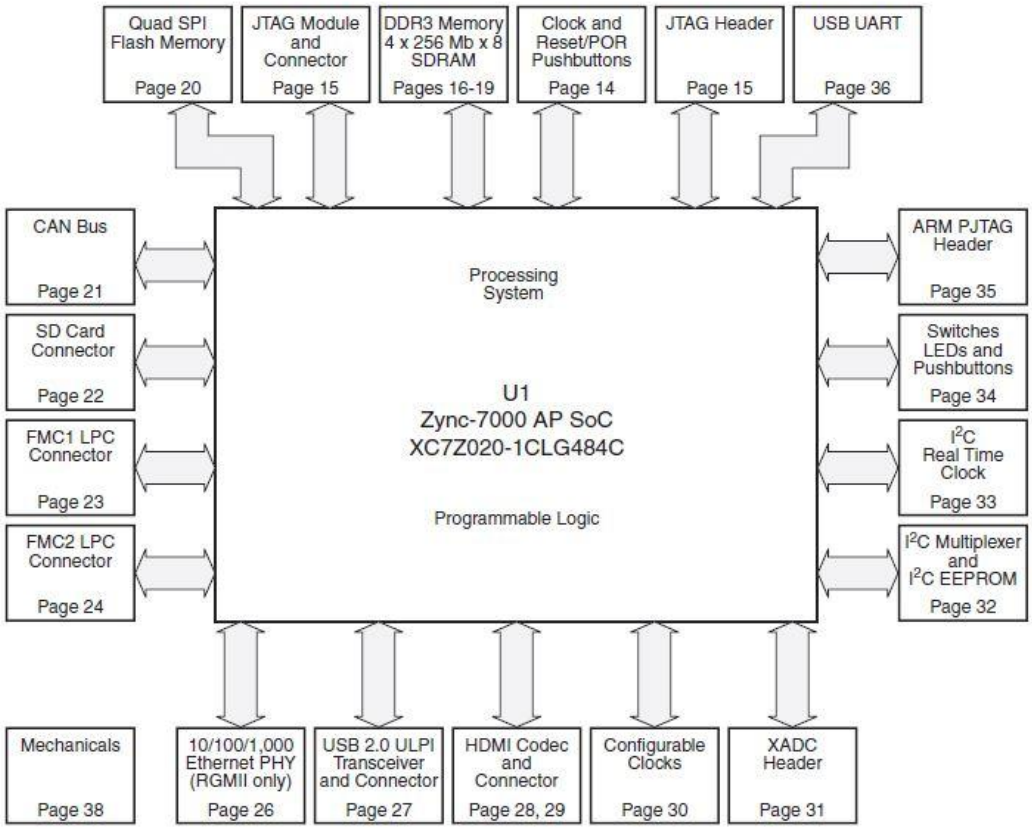


Ilustración 3-2 ZC70 Diagrama de Bloques, sacada de [9]

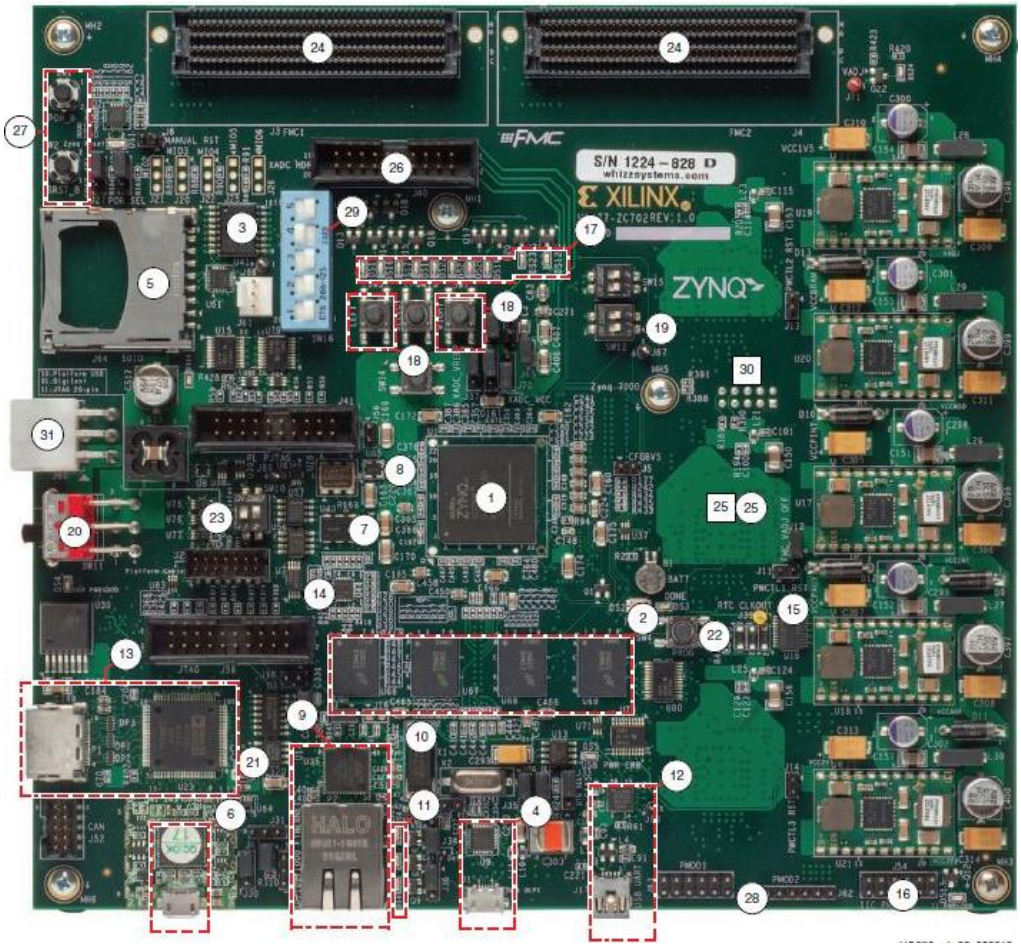


Ilustración 3-3 Aspecto físico de la ZC702, obtenida de [9]

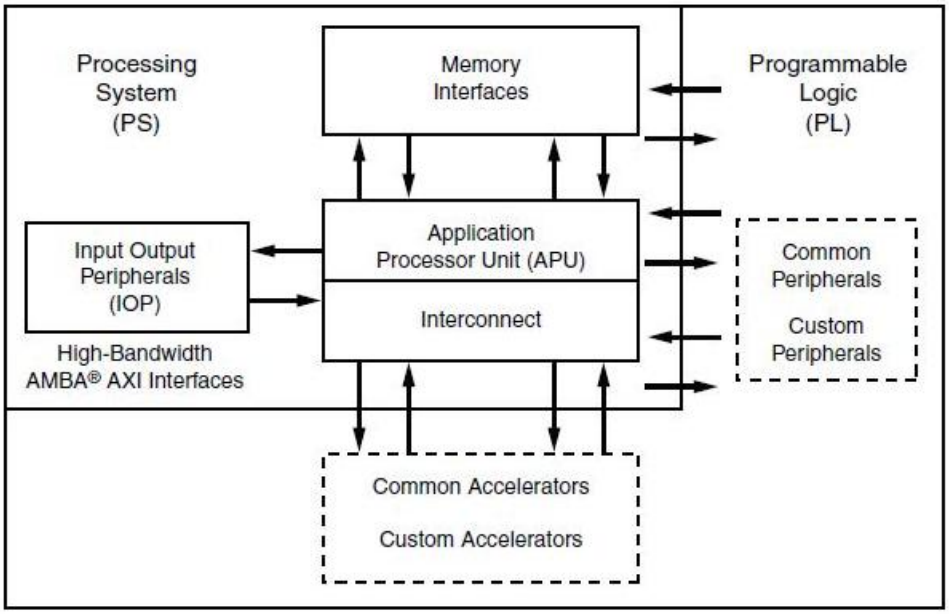


Ilustración 3-4 Diagrama de alto nivel del SoC de la ZC702, sacado de [9]

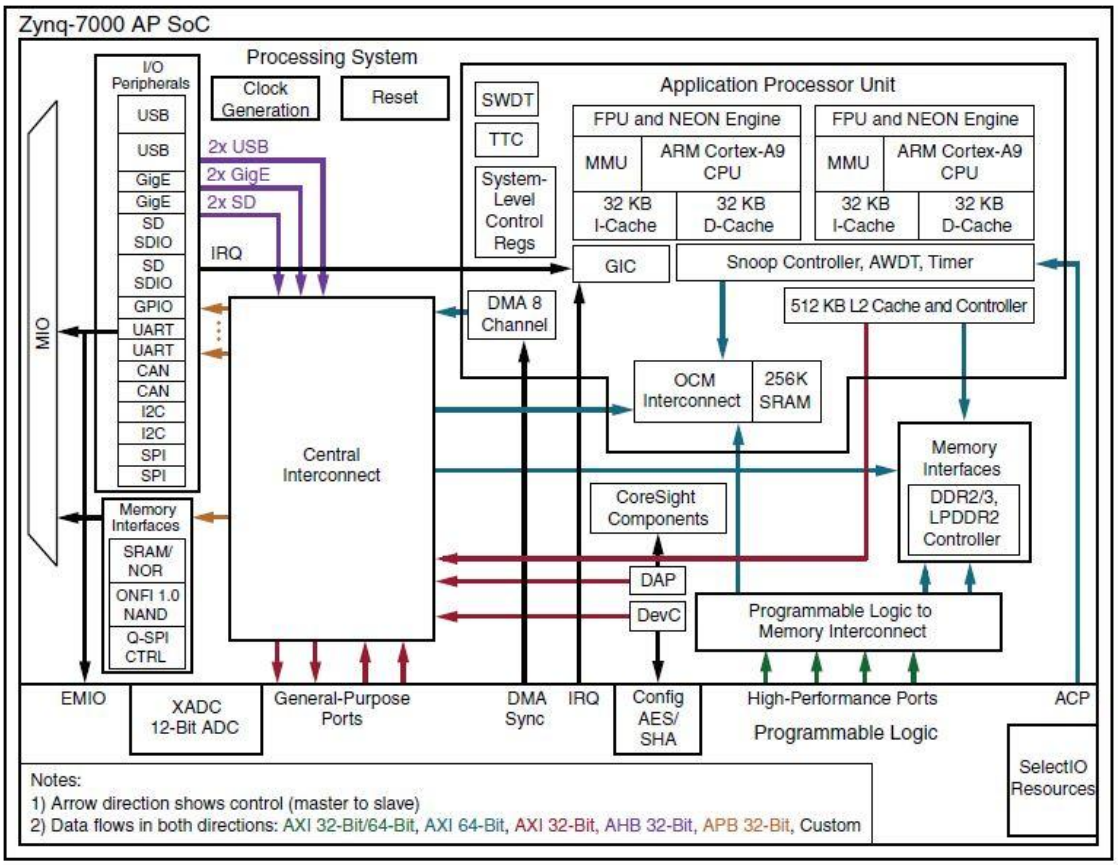


Ilustración 3-5 Módulos del Zynq 7000, sacada de [9]

3.2.1 Módulos del Zynq 7000:

En este apartado, sólo se explicarán en detalle los módulos más usados del SoC en este proyecto. Si desea más información sobre que o cómo funcionan el resto de módulos de la placa, consulte [9-11] y [12].

3.2.1.1 CPU:

La CPU está alojada en la parte de la PS y el procesador de la placa es el Cortex A9, explicado anteriormente. Este procesador puede funcionar con un solo núcleo o con dos. Esta CPU puede programarse sin sistema operativo (bare-metal) o con Linux. Si se quiere volver a ver su arquitectura y/u obtener más información, consulte la Ilustración 3-1 y/o [2].

3.2.1.2 Interrupciones:

Las interrupciones ocurren cuando se produce un evento especial. Entonces, en ese momento el procesador para lo que está haciendo y ejecuta una rutina de interrupción. En este caso, cada CPU puede recibir una serie de interrupciones, que pueden observarse de manera gráfica en la Ilustración 3-6. Para más información, consulte [12].

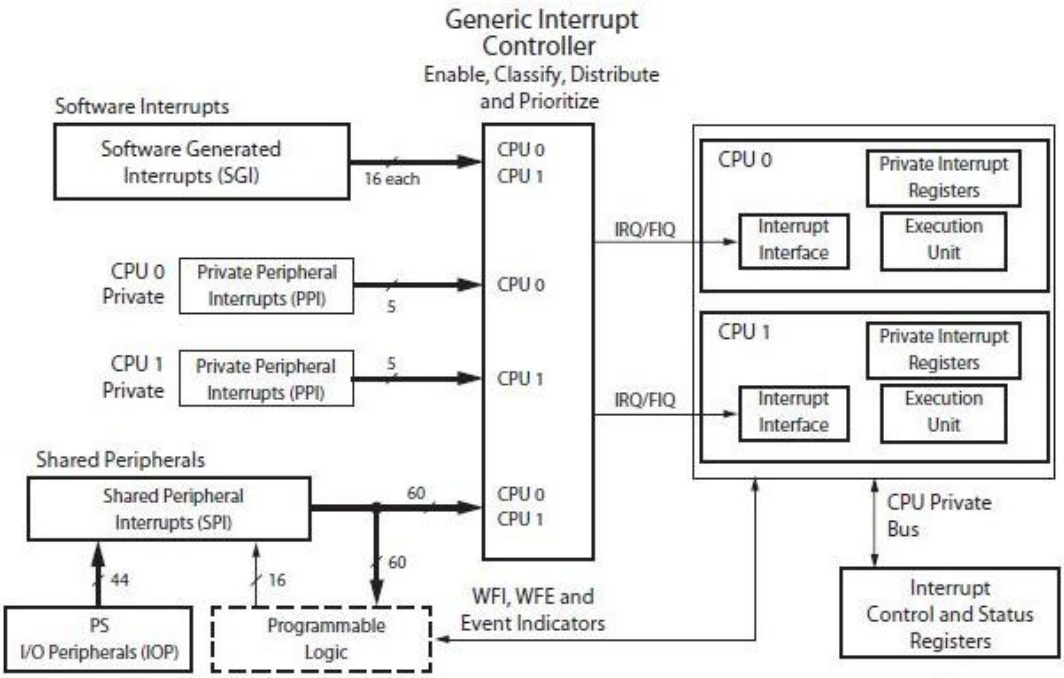


Ilustración 3-6 Diagrama sobre el funcionamiento de las interrupciones, sacada de [12]

Como se ve en la Ilustración 3-6, se puede generar una interrupción mediante:

- Software.
- Periféricos privados de cada CPU.
- Módulos compartidos por ambas CPU (timers, buses de comunicación, etc.).
- Dispositivos I/O.
- PL.

Además, se puede observar que existe un driver para gestionar las interrupciones. Este simplemente se encarga de que cuando se genere una interrupción, se avise a las respectivas CPU para que ejecute su rutina de interrupción. Además, en caso de que ocurran varías interrupciones a la vez, el driver se encarga de gestionar la prioridad de estas y ver quien debe ejecutarse primero.

3.2.1.3 Triple Timer Counter (TTC)

Este módulo contiene tres timers independientes entre sí. Según [12], existen dos TTC en la PS. Las características principales de este módulo [12] son:

- Tres prescalers de 16 bits y contadores up/down de 16 bits.
- Se pueden escoger tres fuentes de reloj (reloj de la PS, reloj de la PL o un reloj externo).
- Cada contador puede generar tres tipos de interrupciones (Overflow, cada un intervalo regular de tiempo o porque el valor de dos contadores coincide).
- A través del reloj de la PL y el externo, se pueden generar ondas, como un PWM.

Dentro del TTC, el timer a utilizar será el 0. Se ha escogido este debido a que usa el reloj de la CPU, cuya máxima frecuencia es válida para este proyecto y nos evita tener que poner un reloj externo o crear uno en la PL.

En la Ilustración 3-7, se mostrará su diagrama de bloques. Si necesita más información sobre este módulo y otros de la ZC702, consulte [12].

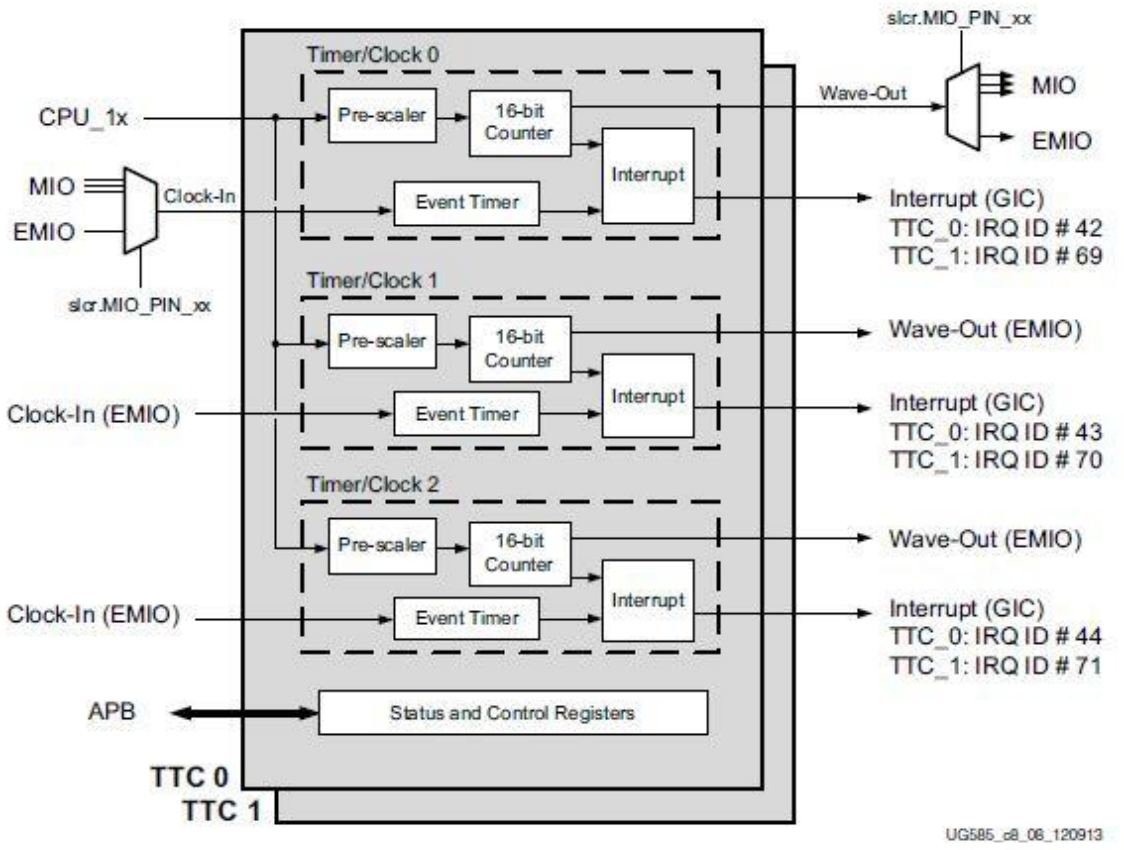


Ilustración 3-7 Diagrama de bloques del TTC, sacada de [12]

3.2.1.4 SD/SDIO Driver

Este controlador nos permite gestionar tarjetas de memoria SD y MMC, permitiendo que se puedan manejar archivos de formato FATFS, el cual es una estructura que nos indica la localización de los diferentes archivos almacenados en el dispositivo. Si requiere más información, consulte [12].

3.3 Herramientas Software

A continuación se explicarán las diferentes herramientas software utilizadas en este proyecto. Por un lado, se necesitará un software que nos permita implementar una serie de algoritmos en la PCB con los que se generará una serie de datos. Aparte, se requerirá de un software matemático que nos permita procesar dichos datos para entrenar, validar y testear los diferentes modelos.

3.3.1 SDK XILINX (XSDK)

XSDK es una extensión del entorno Eclipse que permite el diseño e implementación de aplicaciones para sistemas embebidos en diferentes productos de XILINX, como todas las placas de ZYNQ-7000 SoCs. Se ha escogido este entorno para programar la placa, ya que esta aplicación nos la proporciona el fabricante con una serie de librerías y drivers que simplifican mucho la programación y configuración de los diferentes elementos de la PCB a trabajar.

A continuación, se hará un breve resumen sobre las funcionalidades de este entorno (en caso de precisar más información, consulte [13]):

- Se puede utilizar para diseñar aplicaciones para muchos modelos de Xilinx (Zynq UltraScale+ MPSoC, Zynq-7000 SoCs, y MicroBlaze).
- Se dispone de un editor, compilador para C y C++, linker. Herramientas para depurado en sistemas de un solo procesador o multiprocesador y depuración software y hardware.
- Disponibilidad de librerías y drivers para los distintos módulos de la placa.
- Posibilidad de programar en bare-metal, en FreeRTOS o PetaLinux (una distribución de Linux).

3.3.2 Python

Como se ha comentado en el apartado 1, una vez se han obtenido el valor de los PMC elegidos y el consumo de potencia por cada benchmark ejecutado, se debe construir una serie de modelos matemáticos que nos permitan estimar la potencia de uno o más núcleos de la CPU. Para ello, se necesitará de un software matemático que nos permita identificar los coeficientes de dicho modelo. Se han barajado las opciones de usar herramientas como Matlab, Python u Octave. En este caso, se ha optado por escoger Python debido a que es un software gratuito y open source, que dispone de unas bibliotecas muy potentes para manejo y visualización de datos.

En este proyecto, se ha utilizado como entorno de programación la distribución de Jupyter Python [14], el cuál es open-source y sin ánimo de lucro. Esta distribución, se puede lanzar en un navegador y es muy flexible ya que permite generar una serie de scripts en el editor de dicha plataforma y luego mostrar los resultados, con el código, como si fuera un cuaderno.

Por otro lado, para la implementación del modelo de potencia y la visualización de los resultados de dicho modelo, se han utilizado dos bibliotecas:

- **Numpy:** Esta biblioteca nos proporciona clases y funciones que nos permiten manejar y crear matrices y realizar operaciones de algebra lineal con ellas. Para más información, consulte [15] y[16].
- **Matplotlib:** Esta biblioteca nos permite representar datos de manera gráfica (gráficos de barras, circulares y otros tipos). Si se quiere profundizar en cómo es la librería, consulte [17].

4 DEFINICIÓN DEL MODELO

En esta sección, se indicará que estrategia se usará para diseñar los diferentes modelos, los benchmarks a utilizar en este mismo, cuáles y cómo se van a elegir los PMC para cada modelo y el diseño matemático de dichos modelos.

4.1 Selección del tipo de estrategia: Top-Down

Antes de empezar a diseñar el modelo que estime la potencia, se debe escoger que tipo de modelo de potencia se va a realizar. En este caso, se ha optado por la estrategia "Top-Down" por los siguientes motivos:

- Debido a que el tiempo de desarrollo de este proyecto no es muy largo. El conocer en detalle la arquitectura del procesador y diseñar un modelo "Bottom-Up" consumiría más tiempo del que se dispone. En consecuencia, se ha optado por la estrategia de "Top-Down", cuyos tiempos de desarrollo son mucho más cortos.
- El nivel de error y la "responsivness" que ofrece este modelo son aceptables para las especificaciones de este proyecto.

Después, se debe elegir si se va a trabajar con un sistema operativo (Linux) o ninguno (bare-metal). En el siguiente párrafo, se explicarán las ventajas y desventajas de cada uno.

Programar la placa con un sistema operativo Linux proporciona muchos benchmarks de fácil implementación. Además, Linux posee instrucciones y directivas que permiten sincronizar tareas de forma fácil y sencilla. Además, al utilizar un modelo "Top-Down" no se puede desacoplar el efecto que produce el sistema operativo sobre los PMC y el consumo de potencia.

En cambio, si se programa la placa en bare-metal, el valor de los PMC y del consumo de potencia no será alterado por ningún sistema operativo. Además, la empresa ha desarrollado librerías para facilitar la obtención de dichos datos A cambio de esto, el número de benchmarks disponibles en bare-metal es más limitado. Aparte, muchas veces será necesario que estos sean implementados en lenguaje C. Igualmente, en bare-metal existen mecanismos para sincronizar tareas, pero suelen ser más difíciles de implementar que los mecanismos ofrecidos por Linux.

Para este proyecto, las ventajas que nos ofrece trabajar en bare-metal son más beneficiosas que sus desventajas. Adicionalmente, para este proyecto se dispone de suficientes benchmarks para estimar bien la potencia del modelo y no es un aspecto muy crítico que haya un cierto indeterminismo en la sincronización de tareas. Además, todavía no existe literatura sobre modelos de potencia implementados en una PCB sin sistema operativo por lo que este estudio resulta más provechoso.

Para finalizar, se diseñará y validará un modelo que permite estimar la potencia de un core de la CPU. Después, a partir del anterior modelo, se diseñará y validará un modelo que permita estimar la potencia de los 2 cores de la CPU.

4.2 Diseño y/o selección de benchmarks

Un benchmark es una secuencia de instrucciones, ejecutadas por la CPU, para medir sus características. En este caso, es importante qué benchmarks se van a usar y que cada uno afecte a la CPU de manera diferente. Dotando así de mayor robustez al modelo de potencia a implementar.

Por lo general, existen aplicaciones o programas con varios benchmarks para testear la CPU de ordenadores y/o sistemas embebidos como LMbench [18] o Mibench [19]. El problema es que estos benchmarks requieren de un sistema operativo para funcionar. En consecuencia, no se pueden utilizar dichas herramientas.

Al final, se ha optado por escoger una serie de benchmarks bastante conocidos en el ámbito de la informática, que pueden ser implementados en lenguaje C, más otros benchmarks diseñados de forma interna. Por último, los benchmarks se agruparán en dos categorías: benchmarks para entrenar el modelo de potencia y benchmarks que se usarán como prueba de test. A continuación, se explicará que benchmarks se han usado para cada categoría.

- Entrenamiento :
    - Whetstone: Es un benchmark utilizado para evaluar computadores, desarrollado por Microsoft.
    - Strings: Es un benchmark, cuyo código se ha sacado de [20], utiliza varias funciones de strings para testear la CPU.
    - Tarai: Es una implementación de la función de tak que es una función recursiva. El código de dicho benchmark se ha obtenido de [21].
    - Matmult: Benchmark desarrollado por la empresa que multiplica matrices aleatorias de flotantes.
    - Ficheros: Simplemente coge una lista de enteros y les asigna un valor, el cuál ha sido desarrollado para este proyecto.
    - Sleep: Pone la CPU a dormir, será el benchmark de idle.
    - Pi: Es un benchmark que calcula Pi, cuyo código se ha cogido de [22].
    - Linpack: Es un benchmark que mide la capacidad de computación de flotantes, cuyo código se ha adquirido de [23].

- Test:
    - Básico: Es un benchmark, cuyo código fuente se ha sacado de [20], que implementa funciones básicas de c.
    - Regulador: Este benchmark consiste en implementar un lazo de control digital en lenguaje c y ha sido desarrollado para este proyecto.

4.3 Selección de los PMC

En los apartados anteriores se ha hablado de seleccionar una serie de PMC, pero no se ha especificado cuales. Se sabe que este procesador tiene un set de 57 PMC y la CPU solo puede monitorizar 6 de esos contadores de forma simultánea. Así que en este apartado, se definirán los 6 PMC a usar en dichos algoritmos y el porqué de su elección.

El primer paso a realizar es descartar aquellos PMC que siempre van a ser nulos. En este proyecto no se trabajará con Java ni con el PLE. En consecuencia, los contadores asociados a dichos elementos pueden descartarse. Por otro lado, para seleccionar que PMC se van a utilizar se pueden usar dos enfoques:

1. Probar de manera empírica distintas combinaciones de PMC y ver qué tipo de combinaciones dan mejores resultados. Esta opción es fácil de implementar si el número de PMC disponibles es pequeño. En caso contrario, al existir tantas combinaciones de PMC, el proceso se hace muy largo y tedioso.
2. Buscar en la literatura existente una metodología o una estrategia para escoger los PMC óptimos para diseñar e implementar el modelo de potencia.

Para este proyecto, se aplicará algo similar a lo que se efectuó en [6], pero más simplificado. En [6] para seleccionar el set óptimos de PMC, se llevan a cabo dos tareas: agrupar a los PMC similares en clusters, basándose en la correlación estadística que tienen los PMC entre ellos, e indicar qué PMC están más correlacionados con el consumo de potencia. Lo explicado se puede ver de manera gráfica en la Ilustración 4-1. Aunque en [6], el procesador utilizado es le Cortex-A15 y el set de PMC es diferente al del procesador de este proyecto (Cortex-A9). Esta gráfica nos puede servir como punto de partida para escoger los PMC. Tal y como se narra en [6], una buena selección de PMC se hace escogiendo aquellos PMC que estén muy correlacionados con el consumo de potencia, pero sin seleccionar todos del mismo cluster, dotando así de mayor robustez al modelo. Además en la Ilustración 4-2, se pueden observar que PMC han sido escogidos para el modelo de potencia en [6]. La PMU del ARM Cortex-A15 permite monitorizar 7 PMC de manera simultánea, aunque uno de los contadores siempre es el de ciclos de reloj.

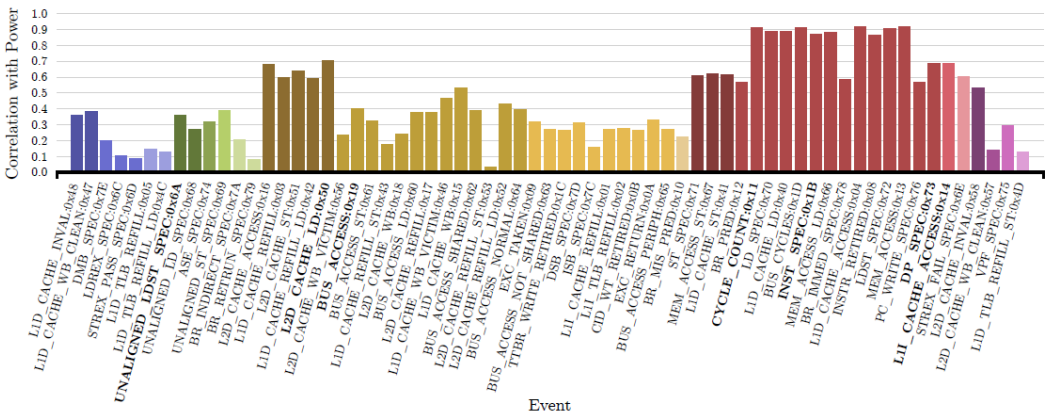


Ilustración 4-1 Correlación de los PMC del procesador Cortex-A15 con el consumo de potencia, agrupados en distintos clusters de colores. Imagen sacada de [6]

	Event Hex	Event Name
1	0x11	CYCLE_COUNT
2	0x1B	INST_SPEC
3	0x50	L2D_CACHE_LD
4	0x6A	UNALIGNED_LDST_SPEC
5	0x73	DP_SPEC
6	0x14	L1I_CACHE_ACCESS
7	0x19	BUS_ACCESS

Ilustración 4-2 Los 7 PMC seleccionados para el modelo de potencia de [6]

En este proyecto no se realizará un análisis de correlación de los PMC con el consumo de potencia, ni estos se agruparán en clusters con otros PMC similares por los siguientes motivos: Primero, aprender, diseñar e implementar este tipo de técnicas consumiría más tiempo del disponible. Segundo, basándonos en la Ilustración 4-1, la arquitectura de la CPU escogida para este trabajo (Ilustración 3-1), los PMC escogidos en [6] y comparando los PMC de ambos procesadores, es posible elegir un set de PMC que permita estimar correctamente la potencia consumida por la CPU.

La estrategia para seleccionar los PMC es la siguiente. Primero, se harán unas pruebas preliminares con una serie de benchmarks y se descartarán aquellos PMC cuyo valor siempre sea nulo. Luego, se agruparán los PMC que son similares entre sí. En este proyecto, no se utilizarán técnicas de correlación como en [6], sino que utilizando como apoyo la Ilustración 4-1, los PMC seleccionados en [6] y la arquitectura del procesador, los PMC serán clasificados en distintos grupos. Por último, una vez agrupados los PMC en diferentes clusters, se elegirán los 6 PMC a utilizar en el modelo a diseñar. En este proyecto no se ha realizado ninguna correlación estadística entre el valor de PMC y el consumo de potencia. En este caso, la relación entre el valor de los PMC y del consumo de potencia se realizará de manera manual y se escogerán PMC similares a los usados en [6].

CICLOS DE RELOJ:

Este cluster agrupa todos los PMC relevantes sobre los ciclos de reloj que cuenta la CPU. Los PMC seleccionados son:

- CLOCK_CYCLES
- DE_CLKEN
- INT_CLKEN

CACHE:

Aquí se agruparán todos los PMC que tienen información sobre el consumo de potencia y la memoria cache del procesador, los PMC elegidos son:

- CACHE_ACESS
- UNALIGNED_ACCESS
- CACHE_REFILL

INSTRUCIONES EJECUTADAS:

Este cluster aglomera los PMC que influyen el consumo de potencia y tienen información sobre el número de instrucciones ejecutadas en la pipeline principal. Los PMC escogidos son:

- MAIN_EXEC
- SW_CHANGEPC

INSTRUCCIONES DE LA ALU (UNIDAD ARITMÉTICO LÓGICA):

Aquí se agrupan todos los PMC que están correlacionados con el consumo de potencia y las instrucciones ejecutadas en la ALU y cuyos PMC son:

- SECEXEC
- FLOATRENAME

BRANCH PREDICTION:

El branch prediction es una técnica utilizada para ahorrar tiempo en los procesadores. Debido a que en las instrucciones de branch no se sabe el camino a seguir hasta el momento final de la ejecución, para ahorrar tiempo la CPU predice que camino se va a coger en la instrucción branch y en caso de equivocarse lo rectifica al final. Los PMC seleccionados son:

- INMEDBRANCH
- BRANCHMISS
- BRANCHPREDICT

UNIDAD DE LOAD/STORE:

Esta unidad se encarga de cargar o almacenar instrucciones en la memoria del programa. Los PMC escogidos son:

- DATA_WRITE
- DATA_READS
- LDSTR

En la Tabla 4-1, se explica qué PMC se han escogido para cada cluster y el porqué de su elección.

GRUPO	PMC elegido	Explicación
CICLOS DE RELOJ	CLOCK_CYCLES	Los 3 PMC suelen tener siempre el mismo valor. Así que por comodidad y debido a que es el PMC usado en el modelo de [6], siempre se usará el contador de "CLOCK_CYCLES".
CACHE	DATA_CACHEACESS	El contador de "UNALIGNED_ACCESS" y "CACHE_REFILL" tienen en algunos benchmarks valor nulo muy bajo. Pudiendo provocar que si se usa como parámetro en el modelo de potencia, este no estime bien la potencia en algunas cargas de trabajo. Aparte el PMC de "CACHE_ACESS" es uno de los PMC seleccionados en [6]. Así que se utilizará dicho contador.
INSTRUCIONES EJECUTADAS	MAIN_EXEC	El contador "MAIN_EXEC" cuenta el número de instrucciones ejecutadas en la tubería principal. Así que su valor estará muy correlacionado con el consumo de potencia. Además, en [6] el contador "DP_SPEC" cuenta el número de instrucciones ejecutadas.
INSTRUCCIONES DE LA ALU (UNIDAD ARITMÉTICO LÓGICA)	SEXEXEC	Se ha probado a usar el contador de "FLOATRENAME" y el modelo de potencia generado con dicho contador da perores resultados que usando el contador "SECEXEX". Esto puede deberse a que en algunos benchmarks no se realicen operaciones en coma flotante, provocando que en algunos benchmarks dicho contador
BRANCH PREDICTION	IMMDEBRANCH	Tras hacer una serie de pruebas se ha visto que los contadores "BRANCHMISS" y "BRANCHPREDICT" en algunas cargas de trabajo son nulos. Mientras que el PMC "INMEDBRANCH" nunca es nulo y siempre tiene un valor diferente en todas las benchmarks ejecutados.
UNIDAD DE LOAD/STORE	DATA_WRITE	Todos los contadores de este cluster dan valores similares y cualquiera de ellos es válido como PMC para estimar la potencia del modelo. Por lo tanto, se ha seleccionado el contador "DATA_WRITE".

Tabla 4-1 Selección y justificación de los PMC para su correspondiente grupo

4.4 Diseño del modelo de potencia

En el apartado 2.1.3 se explicó cómo iba a ser la ecuación del modelo de potencia a diseñar. En este apartado se explicará en profundidad cada término de la ecuación y como a partir del valor de los PMC escogidos, la temperatura y el consumo de potencia se obtendrán los coeficientes del modelo.

En la Ecuación 4-1, se muestra la ecuación para estimar la potencia de un solo core de la CPU. Se indican con a_i, i=0,…, 5, los coeficientes de cada término. Además, se puede apreciar que la

potencia estimada del modelo depende de la temperatura, el valor de los PMC y un término estático. La Ecuación 4-1 está basada en el modelo de potencia que propone [6].

$$Pe = \left(\sum_{i=0}^{5} \frac{PMC_i}{T_s} * Vdd^2 * a_i \right) + Vdd^2 * fclk * a_6 + a_7 * T + a_8 * T^2$$

Ecuación 4-1 Modelo de potencia a implementar para estimar la potencia en un core

Donde *Pe* es la potencia consumida estimada. Esta ecuación puede reescribirse en la notación que se muestra en la Ecuación 4-2. En la cual *a* es una matriz columna que contiene los coeficientes a_i *de* la Ecuación 4-1. *H* es una matriz en la que cada columna, se corresponde con uno de los términos independientes de la Ecuación 4-1 y cada fila se corresponde con el número de veces que se estima la potencia. Por último, *P* es una matriz columna donde se almacenan todos los valores sobre la potencia estimada.

$$P = Ha$$

Ecuación 4-2 Notación alternativa en vectores para la ecuación del modelo

Como se ve en la Ecuación 4-2, el modelo es un sistema linear de ecuaciones, pero en este caso la cantidad de filas en P y H va ser mayor que el número coeficientes *a* existentes. Por lo tanto, al haber más ecuaciones que variables, el sistema no tiene solución. En cambio, se puede encontrar una solución aproximada buscando el mínimo del cuadrado de la norma de (P-H*a), tal y como se ve en la Ecuación 4-3.

$$min_a \|P - Ha\|^2$$

Ecuación 4-3 Minimización del cuadrado de la norma del modelo de potencia

La Ecuación 4-3 es un problema de optimización cuadrática que puede ser resuelta de manera analítica, imponiendo que el gradiente de la norma al cuadrado equivale a cero. La solución es la que se muestra en la Ecuación 4-4.

$$a = (H'H)^{-1}H'P = H^*P$$

Ecuación 4-4 Solución para obtener los coeficientes del modelo

Aplicar la Ecuación 4-4, implica la inversión de matrices, un proceso que puede generar problemas numéricos y oneroso desde un punto de vista computacional. Así que en vez de calcular explícitamente una inversa, se resolverá el sistema de ecuaciones lineales de la Ecuación 4-5, obteniendo así los coeficientes a_i del modelo de potencia.

$$(H'H)a = H'P$$

Ecuación 4-5 Sistema de ecuaciones lineales a resolver para obtener los coeficientes del modelo de potencia

Respecto al modelo para estimar la potencia en *m* cores, en la Ecuación 4-6 se muestra la ecuación que nos permite estimar la potencia cuando la CPU tiene m cores activos.

$$Pe = \sum_{j=0}^{m} \left(\sum_{i=0}^{5} \frac{PMC_i}{T_s} * Vdd^2 * a_i \right) + \alpha(Vdd^2 * fclk * a_6 + a_7 * T + a_8 * T^2)$$

Ecuación 4-6 Modelo matemático para estimar la potencia de una CPU con 2 o más cores activos

Un aspecto importante, es que los coeficientes a_i del modelo de potencia, se obtendrán con el método para estimar la potencia con un solo core de la CPU activo. Además, en todos los cores se utiliza el mismo set de PMC, para estimar el consumo de potencia. Por lo tanto, los coeficientes a_i serán siempre los mismos para todos los núcleos activos en el modelo a implementar. Una vez obtenidos los coeficientes a_i, solo será necesario encontrar el valor óptimo de α. Para obtener el valor de α, se recurrirá al método de las aproximaciones sucesivas. El α óptimo para el algoritmo del apartado 5.2 y aplicando la Ecuación 4-6, es un α de 1,25.

Por último se puede ver que la Ecuación 4-1 y la Ecuación 4-6 serían muy fáciles de implementar en una CPU. Ya que ambas ecuaciones son una suma de ecuaciones lineales, cuya implementación es tan fácil como realizar un conjunto de producto escalares y luego sumar el resultado de dichas operaciones. Algo que muchas CPU y lenguajes de programación pueden realizar de manera sencilla y rápida. Además, dichas operaciones requieren muy poco tiempo de procesamiento, provocando que la estimación de la potencia sea rápida.

5 METODOLOGIA

En este apartado, se explicará el diseño de los algoritmos, los cuáles generarán una serie de datos con los que se podrá construir y evaluar las propiedades de los diferentes modelos a implementar. En la Ilustración 5-1, puede observarse un esbozo de que tareas deben de realizar dichos algoritmos.

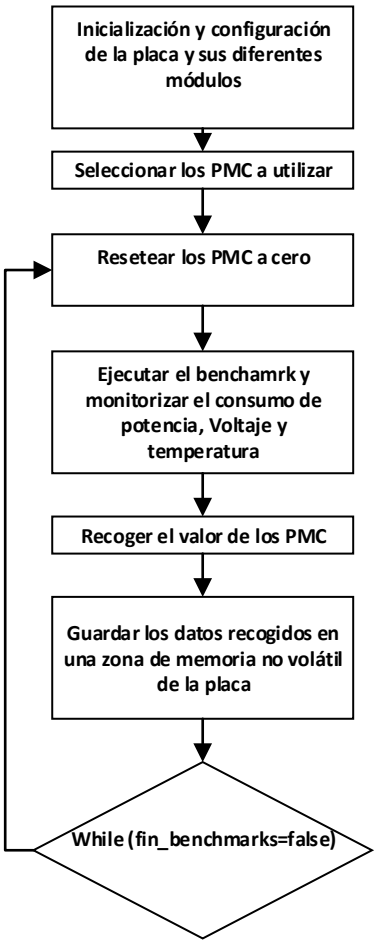


Ilustración 5-1 Esbozo del algoritmo a diseñar

Respecto al almacenamiento de los datos, se ha decido guardarlos en una tarjeta SD, puesto que es un dispositivo de memoria no volátil y puede conectarse a la ZC702 para guardar y/o cargar archivos. Además, los datos de los PMC y del consumo de potencia, se almacenarán en un fichero de texto con las siguientes columnas:

- Índice benchmark: Cada benchmark está numerado, empezando desde 0. Este número sirve para identificar el benchmark usado. El orden de los benchmarks a usar será el mostrado
- Valor numérico de los PMC: Serán 6 columnas con el correspondiente valor de los PMC seleccionados. Dichos números serán enteros y no tendrán unidades.
- Nombre de los PMC: Nombre de los PMC utilizados, estos aparecen de izquierda a derecha.

- Potencia Benchmark: Indica el valor de la potencia consumida de ese benchmark en mW.
- Voltaje: Indica el voltaje de la placa en mV, que es constante y mide 1000mV.
- Temperatura: Muestra la temperatura de la placa en milésimas de grado centígrado.
- Potencia estimada del modelo: Muestra el valor estimado de la potencia, en mW, por el modelo de potencia.

Antes de entrar en más profundidad sobre el algoritmo a diseñar. Se deben tener en cuenta las siguientes especificaciones:

- Los benchmarks se ejecutarán de manera secuencial.
- Cada benchmark se ejecutará varias veces. Teniendo así, varias muestras de los PMC y del consumo de potencia del mismo benchmark. Consiguiendo que el modelo tenga mayor fiabilidad.
- El consumo de potencia debe de medirse cuando se esté ejecutando el benchmark. Esto significa que ambas tareas deben de ejecutarse de manera simultánea.

En este proyecto, se han desarrollado 3 algoritmos para cumplir lo descrito en la Ilustración 5-1. Además, algunos algoritmos servirán para estimar la potencia en un solo core de la CPU y otros para estimar la potencia de los dos núcleos de la CPU. A continuación, se listará cada algoritmo utilizado.

- **Algoritmo 1 core e interrupción**: Este algoritmo utiliza un solo núcleo de la CPU y nos permite generar datos para estimar la potencia consumida por un core del procesador.
- **Algoritmo 2 cores e interrupción**: Este algoritmo utiliza dos núcleos de la CPU y nos permite generar datos para estimar la potencia consumida por un core del procesador.
- **Algoritmo 2 cores e interrupciones**: Este algoritmo utiliza dos núcleos de la CPU y nos permite generar datos para estimar la potencia consumida por dos cores del procesador.

5.1 Algoritmos para estimar la potencia de un core

5.1.1 Algoritmo: 1 core e interrupción (Potencia 1 Core)

A continuación, se explicará el diseño y funcionamiento de este algoritmo, el cual produce una serie de datos con los que construir el modelo para estimar la potencia de 1 core. En la Ilustración 5-2, se puede ver un esquema de las tareas que ejecuta este algoritmo.

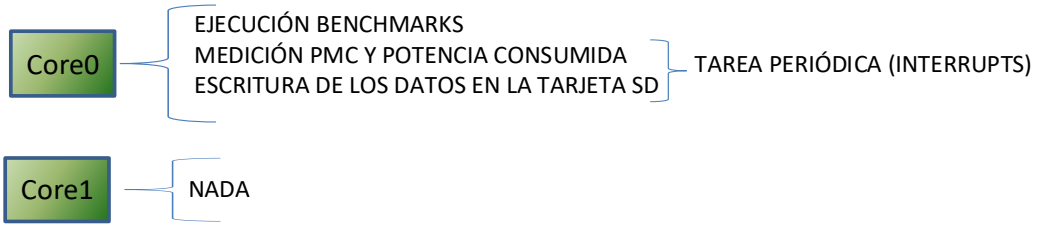


Ilustración 5-2 Esquema sobre las actividades que realiza el algoritmo de 1 core e interrupción

Este algoritmo usa un timer del TTC y una interrupción para obtener periódicamente muestras de los PMC y del consumo de potencia, para cada benchmark ejecutado. Por un lado, el programa principal estará ejecutando el benchmark de manera indefinida. Por otro lado, se ha configurado el TTC para que tras un intervalo regular de tiempo, elegido por el usuario, lance una interrupción. Cuando salte la interrupción, se medirá la potencia, voltaje, temperatura y el valor de los PMC y se contará cuantas muestras de dichos parámetros para ese benchmark se han tomado. Si el número de muestras tomadas supera un umbral, escogido por el usuario, se dará la orden para que se cambie el benchmark a ejecutar al salir de la interrupción. Posteriormente, se escribirán el valor de los PMC, potencia, voltaje, temperatura y tiempo de ejecución en la tarjeta SD, con el formato explicado al principio del apartado anterior. Por último, se saldrá de la rutina de interrupción volviendo al programa principal.

Es importante notar que al iniciarse la rutina de interrupción el benchmark deja de ejecutarse. Esto nos lleva a preguntarnos, cómo se puede asegurar que la medida de potencia tomada en la rutina de interrupción, se corresponda con la potencia consumida por el benchmark. Esto puede asegurarse por el siguiente motivo: el Hardware usado para medir la potencia, utiliza un filtro RC para evitar el antialiasing. Este filtro provoca que cuando se producen grandes cambios de potencia, la medida no cambie de manera instantánea, sino que empiece a descender lentamente y con un cierto retraso. Por lo tanto, si se mide la potencia consumida justo al entrar en la interrupción, la medida será válida.

Otro requisito muy importante en este algoritmo es que se cumpla lo estipulado en la Ecuación 5-1.

$$t_{TTC} \gg t_{INT}$$

Ecuación 5-1 Ecuación que se debe de cumplir para que el algoritmo funcione

Siendo t_{TTC}, el periodo que hay entre una interrupción y otra del timer del TTC y t_{INT} el tiempo que tarda en procesarse la rutina de interrupción. Se ha comprobado de manera empírica que t_{INT} no es un valor constante, esto puede provocar que entre dos medidas del valor de los PMC, de un mismo benchmark con el mismo tiempo de ejecución, sean muy diferentes entre sí. Esto se debe a que el tiempo que se está ejecutando un benchmark no es t_{TTC}, sino lo que se muestra en la Ecuación 5-2.

$$t_{BENCH} = t_{TTC} - t_{INT}$$

Ecuación 5-2 Ecuación que muestra el tiempo real que se está ejecutando dicho benchmark

Como se puede observar solo si cumple la Ecuación 5-1, se puede asumir que el tiempo que tarda el TTC en generar una interrupción y el tiempo real que se está ejecutando el benchmark son casi iguales. Provocando que las diferencias de los PMC entre distintas pruebas del mismo benchmark sean insignificantes. Aparte, la Ecuación 5-1 nos impone una restricción temporal sobre la frecuencia máxima a la que puede saltar la interrupción.

Para ver como es el algoritmo de este apartado, consulte la Ilustración 5-3 e Ilustración 5-4. Como se puede apreciar en la Ilustración 5-4, de cada benchmark solo se cogerán un número finito de muestras sobre la medida de los PMC y del consumo de potencia. Cada vez que se

recoge una muestra, el valor de “Contador” se incrementa en una unidad y se pone a cero al cambiar de benchmark. Cuando el valor de contador alcance un valor máximo, determinado por el usuario, desde la interrupción se dará la orden para cambiar de benchmark, aumentado una unidad el valor de “n”. Esta orden no es inmediata, ya que puede suceder que cuando la interrupción ordene cambiar de benchmark, este todavía no haya finalizado. En ese caso, se esperará a que el benchmark termine de ejecutarse e inmediatamente se saltará al siguiente benchmark. Mientras tanto, cuando salte la interrupción, no se recogerán más muestras, hasta que empiece a ejecutarse el nuevo benchmark.

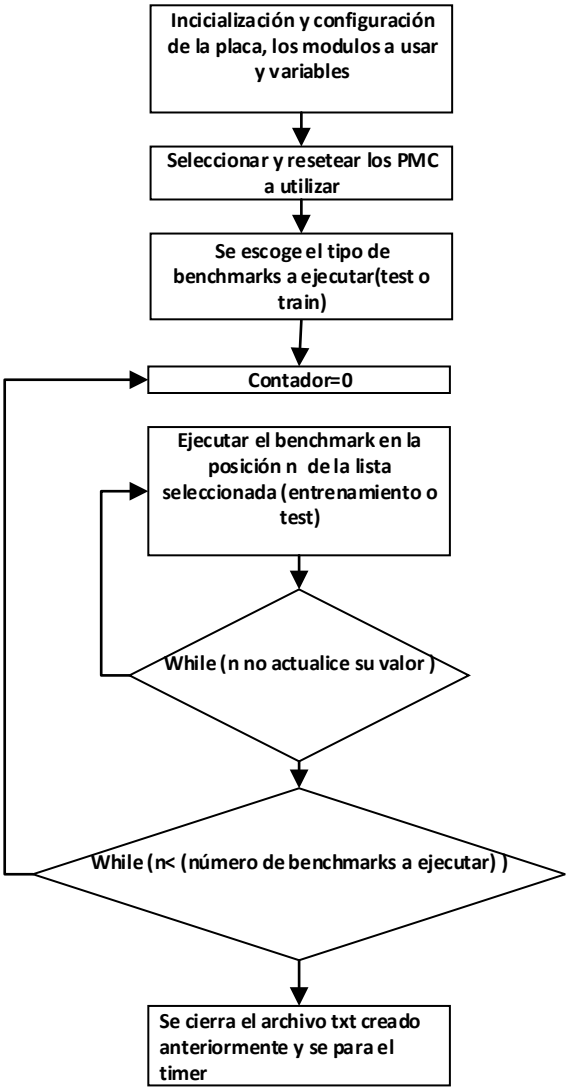


Ilustración 5-3 Algoritmo en pseudocódigo para algoritmo de interrupción con 1 core (programa principal)

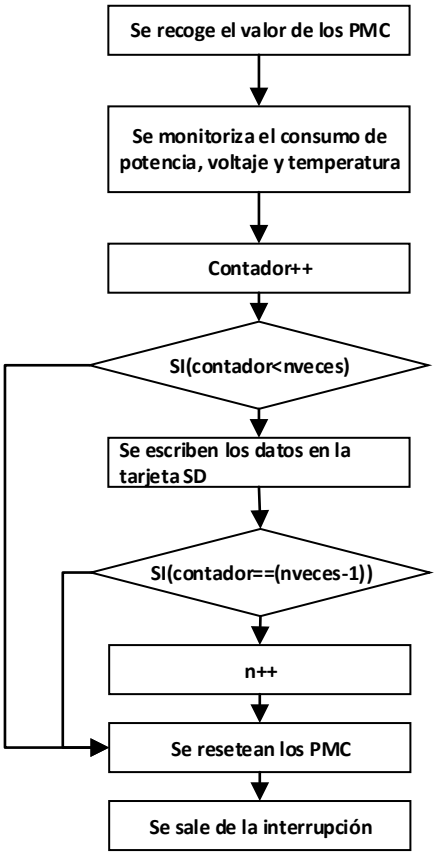


Ilustración 5-4 Algoritmo en pseudocódigo para algoritmo de interrupción con 1 core (interrupción)

5.1.2 Algoritmo: 2 cores e interrupción (Potencia 1 Core)

Este algoritmo es muy similar al del apartado 5.1.1, pero con ciertas diferencias. Por un lado, en el core 0 se hace todo lo explicado en el apartado 5.1.1, excepto la monitorización del consumo de potencia, voltaje y temperatura, la cual se realizará en el core 1 y mediante una serie de registros se enviará dichos datos al core 0. En la Ilustración 5-2, se resume todo lo explicado de manera gráfica.

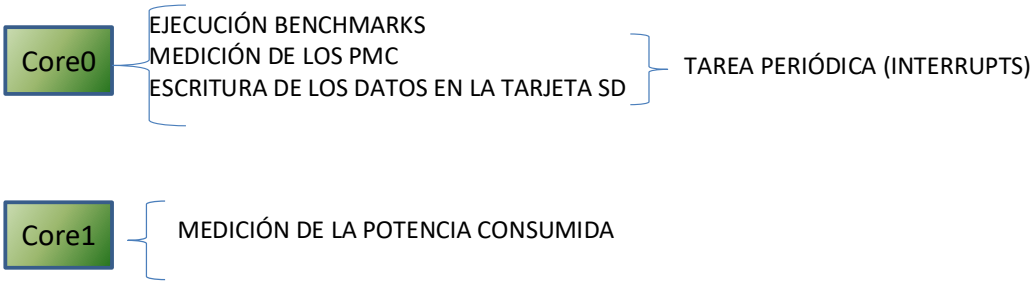


Ilustración 5-5 Esquema sobre el funcionamiento del algoritmo de 2 cores e interrupción

Para que este algoritmo funcione deben cumplirse las restricciones temporales explicadas en el apartado 5.1.1. Además, debe cumplirse lo estipulado en la

$$t_{mt} < t_{bench}$$

Ecuación 5-3 Restricción temporal para el algoritmo de 2 cores e interrupción

Siendo t_{mt}, el tiempo que se tarda en monitorizar la placa (potencia, voltaje, temperatura) y t_{bench} es el tiempo que tarda el benchmark en ejecutarse. La Ecuación 5-3 debe cumplirse para que este algoritmo funcione. Si t_{mt} es más pequeño que t_{bench} , se puede asegurar que la potencia, voltaje y temperatura medidas en ese instante temporal, corresponden con el benchmark ejecutado. En caso de no cumplirse la Ecuación 5-3, no se puede asegurar que la medida de potencia obtenida, se corresponde con la del benchmark ejecutado.

El algoritmo a diseñar en el core 0 es muy similar al del apartado anterior. La única diferencia radica en que durante la rutina de interrupción, el valor de la potencia consumida se obtendrá a través de un registro de la memoria compartida y no del sensor. Por otro lado, el algoritmo del core 1 puede verse en la Ilustración 5-6.

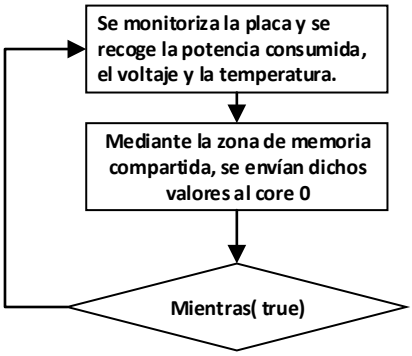


Ilustración 5-6 Algoritmo en pseudocódigo para el core 1 de la CPU y el algoritmo de 2 cores e interrupción

5.2 Algoritmos para estimar la potencia de dos cores

5.2.1 Algoritmo 2 cores e interrupciones: (Potencia 2 cores)

Este algoritmo nos permitirá generar los datos para el modelo que estima la potencia de 2 cores de la CPU. Para ello, se utilizará el algoritmo del apartado 5.1.1, pero con algunas diferencias y se ejecutará de manera simultánea en ambos cores. Un esquema de las actividades ejecutadas en los diferentes cores puede verse en la Ilustración 5-7.

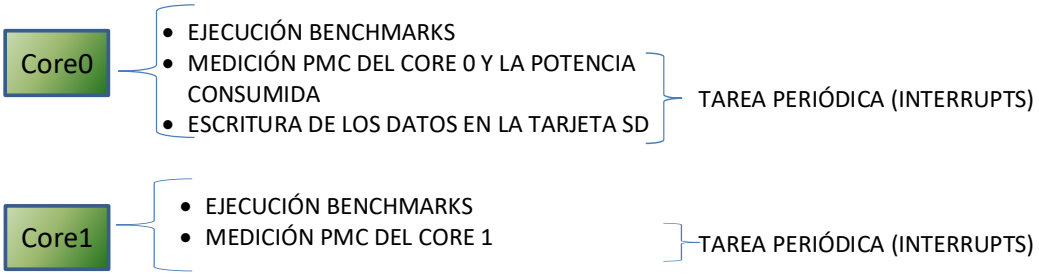


Ilustración 5-7 Esquema del funcionamiento del algoritmo de 2 cores e interrupciones

Es importante que antes de poner en funcionamiento este algoritmo, se haya implementado el algoritmo del apartado 5.1.1 y haber obtenido los coeficientes del modelo de potencia para un solo core. Debido a que en el propio algoritmo se estima la potencia del modelo, es necesario haber obtenido los coeficientes del modelo para un solo core. De ese modo, se puede aplicar la Ecuación 4-6, que estima la potencia con dos o más cores. En el siguiente párrafo, se explicarán las diferencias de cada core, con respecto al algoritmo del apartado 4.2

Ahora cada core estará ejecutando un conjunto de benchmarks, estos pueden estar ejecutando el mismo benchmark a la vez o no, provocando que el número de combinaciones de benchmarks a ejecutar en ambos cores aumente de manera exponencial. En este proyecto, se han desarrollado tres secuencias de benchmarks a ejecutar en ambos cores, las cuáles son:

- Que en ambos cores se ejecuten los mismos benchmarks de manera simultánea y que ejecute la serie de benchmarks de entrenamiento del apartado
- La secuencia mostrada en la Tabla 5-1, que no es más que se ejecuten, en cada core, distintos benchmarks de entrenamiento del apartado 4.2, sin ningún orden en especial.
- En la Tabla 5-2, se puede ver la otra secuencia a ejecutar, que es como la secuencia de benchmarks de la Tabla 5-1, pero incluyendo algún benchmark de test del apartado 4.2.

Índice de benchmark	Benchmark a ejecutar en core 0	Benchmark a ejecutar en core 1
0	calcular_pi	benchmark_strings
1	whetstone	prueba_ficheros
2	matmult	prueba_ficheros
3	benchmark_strings	sleep
4	tarai	calcular_pi
5	whetstone	matmult
6	sleep	tarai
7	prueba_ficheros	benchmark_strings

8	linpack	linpack

Tabla 5-1 Secuencia de ejecución de los benchmarks de entrenamiento con este algoritmo

Índice de benchmark	Benchmark a ejecutar en core 0	Benchmark a ejecutar en core 1
0	regulador_planta	benchmark_strings
1	whetstone	prueba_ficheros
2	matmult	prueba_ficheros
3	benchmark_strings	sleep
4	tarai	calcular_pi
5	whetstone	matmult
6	sleep	tarai
7	prueba_ficheros	regulador_planta
8	linpack	linpack

Tabla 5-2 Secuencia de ejecución de los benchmarks de entrenamiento y test con este algoritmo

En el core 1 se usará el timer 1 del TTC, ya que el timer 0 está ocupado por el core 0. Además, para estimar bien la potencia, los timers de ambos cores deben de sincronizarse y empezar a la vez. Por último, la escritura de los datos en la tarjeta se hará en el core 0. Así que en el core 1 no se trabajará con la tarjeta SD. Por lo tanto, en la rutina de interrupción de este core solo se recogerá el valor de los PMC del core 1 y se estimará la potencia estimada por los PMC de dicho core. Luego, dicho valor se enviará al core 0, a través de la memoria compartida. Asimismo, el índice del benchmark ejecutándose o las veces que se han entrado en la interrupción ejecutando dicho benchmark, se recibirán del core 0, mediante una serie de registros en la zona de memoria compartida.

Por otro lado, una de las diferencias a destacar en el core 0, entre el algoritmo de este apartado y el del apartado 5.1.1, se encuentra en la rutina de interrupción. Ya que primero se estima la potencia del core 0, calculando solo la potencia estimada por los PMC. Después, se espera en bucle hasta que se actualice la potencia del core 1 y se coja el valor de la zona de memoria compartida.

Otro aspecto importante, es que la rutina de interrupción de ambos cores, debe cumplir las siguientes restricciones:

- El índice del benchmark a ejecutar y las veces que con un mismo benchmark se ha entrado a la interrupción, son recursos compartidos. Por lo tanto, cuando el valor de dichas variables se utilice en un core, el otro núcleo no puede leer ni escribir su valor en la zona de memoria compartida. Además, una vez se haya actualizado dicha variable, el otro núcleo debe de actualizar el valor de dicha variable de manera inmediata.
- Se debe asegurar que primero se calcula la potencia estimada por los PMC del core 1 y se envía a la zona de memoria compartida, antes de estimar la potencia total consumida por la CPU. Si no se hiciera en este orden, el valor que se recoge de memoria compartida no es la potencia consumida por los PMC del core 1 en ese instante temporal y el modelo no calcularía la potencia correctamente.
- El tiempo de muestro de los PMC en ambos cores debe ser el mismo. De esta forma se puede asegurar que el tiempo que se está ejecutando ambos benchmarks en cada cores es el mismo. Por lo tanto, ambos cores deben entrar y salir de la interrupción de manera simultánea. Si no se cumpliera dicha condición, los parámetros para construir el modelo serían erróneos y este no realizará una correcta estimación de la potencia.

Para cumplir las siguientes restricciones, se necesita de un mecanismo de exclusión mutua multi-core. El problema es que la arquitectura de ARM no provee de ninguna instrucción o directiva que permita la sincronización entre dos cores. Por lo tanto, se tendrá que diseñar e implementar un mecanismo de sincronización para cumplir con las restricciones descritas.

El mecanismo diseñado es como un semáforo, solo que se puede utilizar en varios cores y sin sistema operativo. Un esquema de cómo funciona, se puede ver en la Ilustración 5-8. Una observación importante para que este algoritmo funcione es que exista una zona de memoria compartida en la que ambos cores puedan hacer operaciones de lectura y escritura. Si no se cumple este requisito, el mecanismo de sincronización no funcionará.

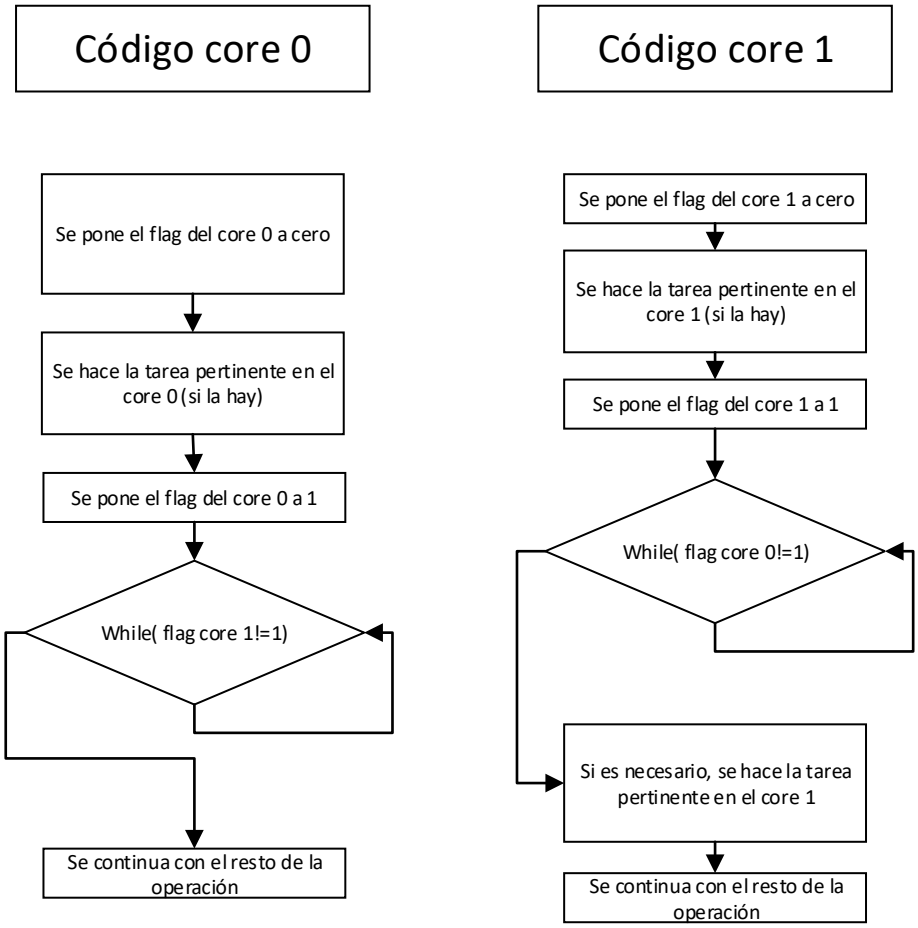


Ilustración 5-8 Algoritmo de sincronización a implementar en la rutina de interrupción

A partir de la Ilustración 5-8, se explicará en mayor detalle como es el algoritmo diseñado. Como se puede apreciar, el algoritmo tiene dos variables booleanas, que se llaman flags y tienen una serie de peculiaridades:

- Cada flag está asociado a un core, la escritura y lectura de cada flag se realizarán en núcleos diferentes.
- El flag solo puede tomar 2 valores: si este vale 0, cuando se alcance cierto punto del código, el core al que está asociado se mantendrá en espera ocupada hasta que su valor cambie. Si el flag vale 1, significa que ya se ha realizado la sincronización o se ha salido de la sección crítica y el núcleo correspondiente puede continuar su actividad.

Este mecanismo nos permite dos cosas: primero, que un core pueda acceder de manera exclusiva a un recurso compartido, sin que el otro core lo pueda modificar; segundo, sincronizar los dos cores para que uno de ellos no avance, hasta que el otro core no termine una determinada tarea.

Un aspecto relevante es como seleccionar el α óptimo, en el proceso para obtener dicho α se ejecutará el método de las aproximaciones sucesivas. Es decir, se elegirá un α y se ejecutará el algoritmo, como el algoritmo estima la potencia durante el tiempo de ejecución, luego, se comparará el valor entre la potencia estimada y la real para dicho α. En función del error entre

ambas magnitudes, se aumentará o disminuirá el valor de α. Este proceso se repetirá hasta alcanzar el α óptimo.

A continuación, se mostrará el algoritmo del core 0 (programa principal e interrupción), en la Ilustración 5-9 y la Ilustración 5-10. Por otro lado, en la Ilustración 5-11 y la Ilustración 5-12, se mostrará el algoritmo del core 1 (Programa principal e interrupción).

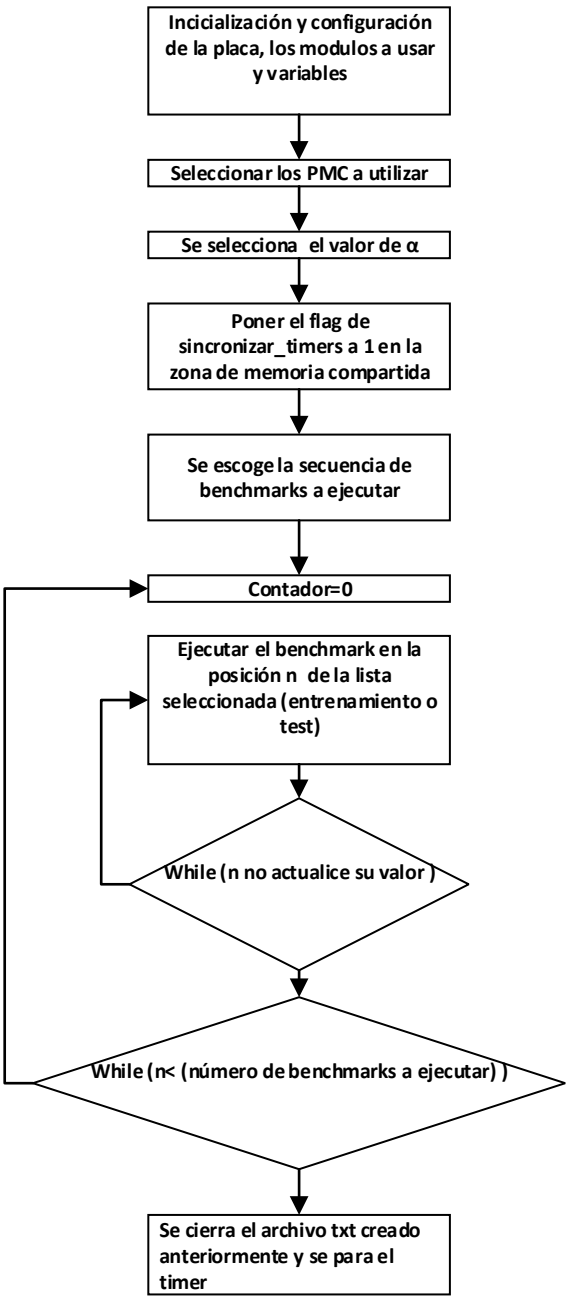
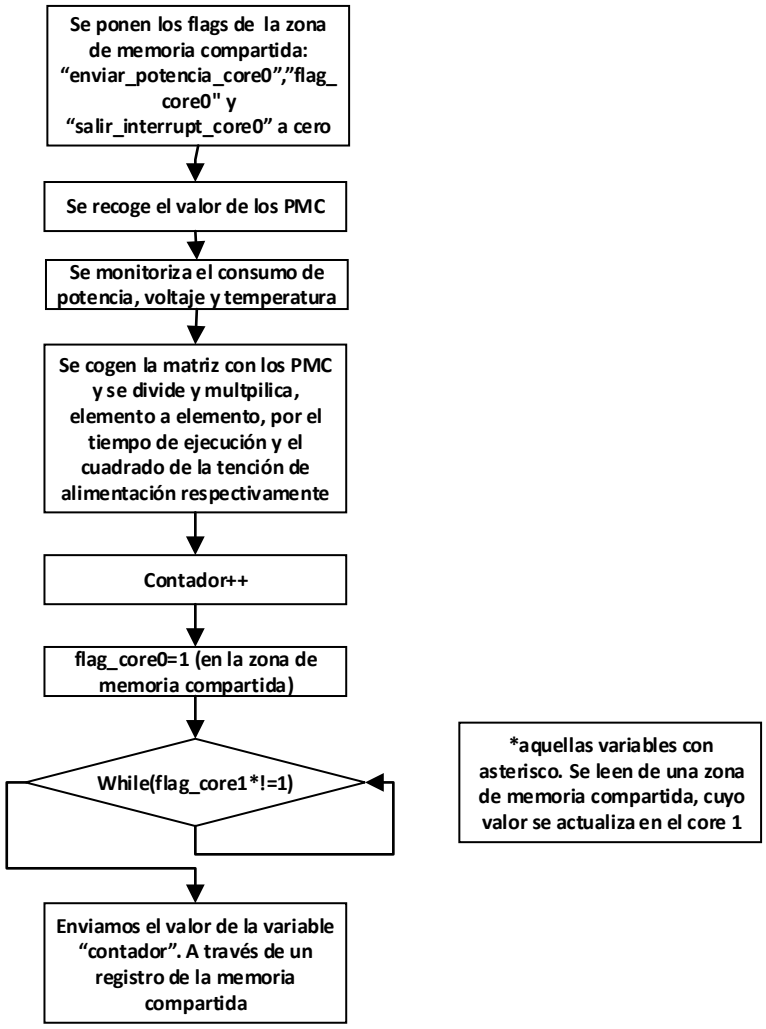


Ilustración 5-9 Algoritmo 2 cores e interrupciones (potencia 2 cores): core 0-programa principal


Se ponen los flags de la zona de memoria compartida: “enviar_potencia_core0”,”flag_core0" y “salir_interrupt_core0” a cero
Se recoge el valor de los PMC
Se monitoriza el consumo de potencia, voltaje y temperatura
Se cogen la matriz con los PMC y se divide y multpilica, elemento a elemento, por el tiempo de ejecución y el cuadrado de la tención de alimentación respectivamente
Contador++
flag_core0=1 (en la zona de memoria compartida)
While(flag_core1*!=1)
Enviamos el valor de la variable “contador”. A través de un registro de la memoria compartida
*aquellas variables con asterisco. Se leen de una zona de memoria compartida, cuyo valor se actualiza en el core 1

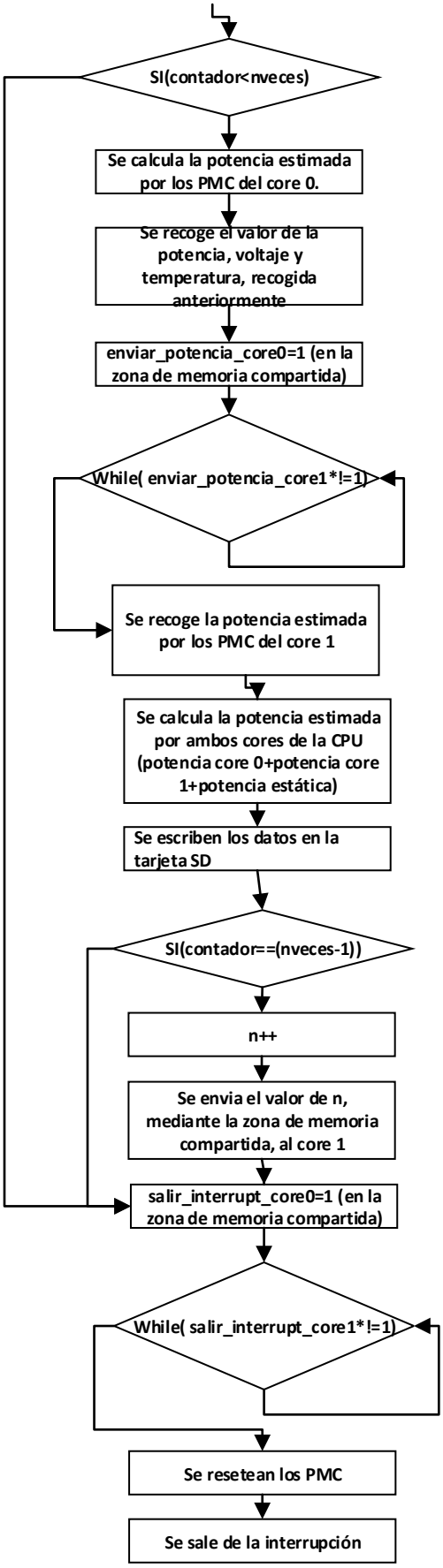


Ilustración 5-10 Algoritmo 2 cores e interrupciones (potencia 2 cores): core 0 interrupción

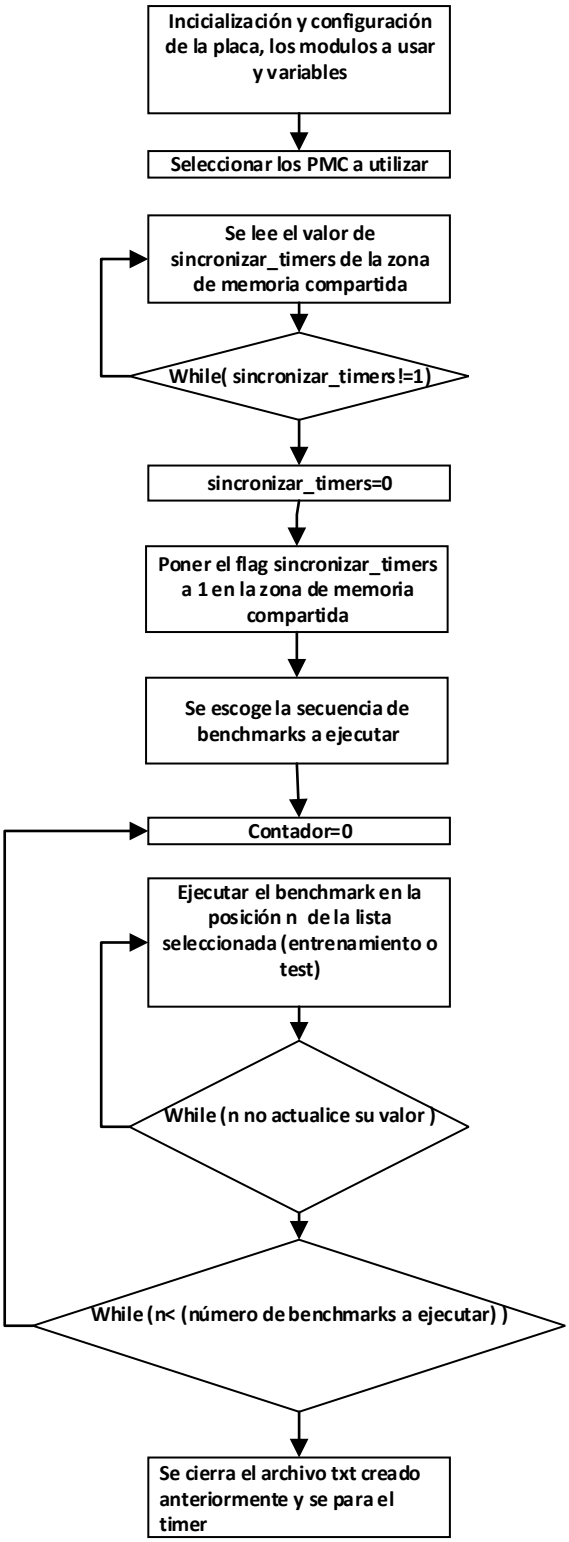


Ilustración 5-11 Algoritmo 2 cores e interrupciones (potencia 2 cores): core 1-programa principal

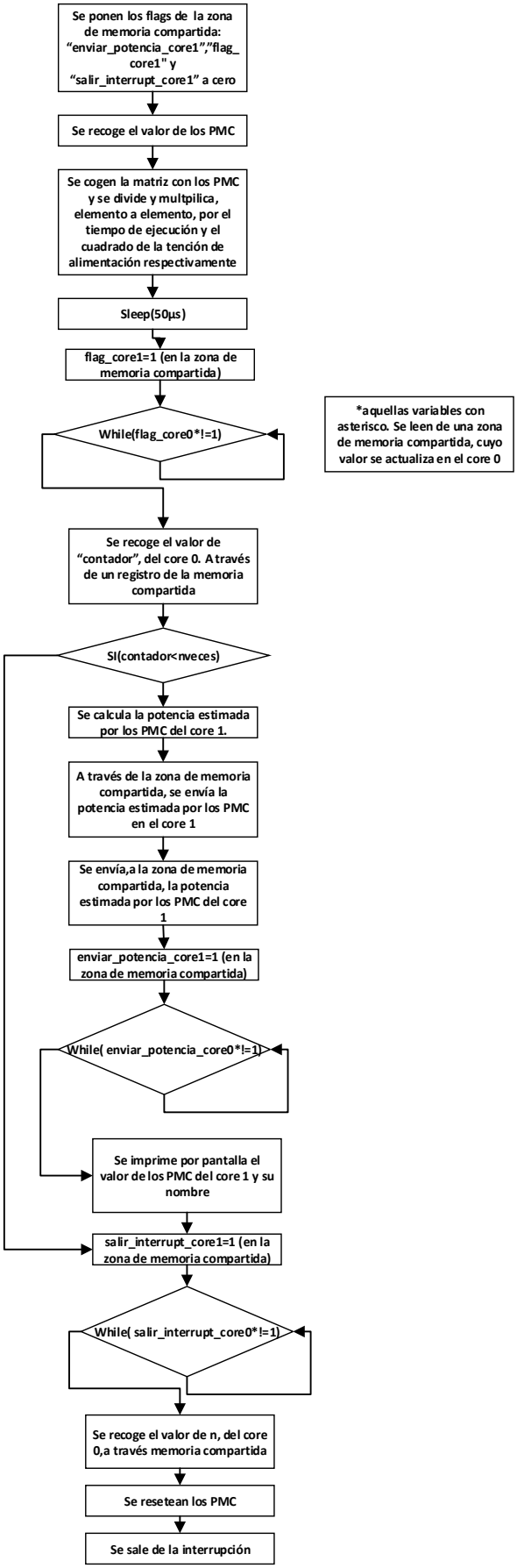


Ilustración 5-12 Algoritmo 2 cores e interrupciones (potencia 2 cores): core 1 interrupción

5.3 Implementación de los algoritmos en la placa Zynq-7000

Ahora que ya se explicó el diseño de los algoritmos propuestos, se mostrará como dichos algoritmos serán implementados en la Zynq-7000 para que puedan funcionar. En este apartado se hablará de que bibliotecas y módulos se van a usar para configurar los diferentes partes del algoritmo (2 cores, PMC, SD, monitorización, etc.).

5.3.1 Xilinx SDK:

Para implementar los algoritmos explicados en los apartados 5.1 y 5.2, se usará el entorno "Xilinx SDK", que nos permite programar diferentes placas de circuito impreso. El programa funciona por proyectos que dejan configurar diferentes opciones:

- Sistema operativo a usar (bare-metal, Linux o FreeRtos). En este proyecto, siempre se usará bare-metal.
- Plataforma Hardware a usar, es un archivo con la configuración hardware de la placa a usar. En esta opción siempre se escogerá el archivo de la placa ZC702.
- Núcleo del procesador a utilizar.
- Lenguaje de programación del proyecto (C, C++). En este caso, debido a que ciertas librerías han sido implementadas en lenguaje C, se usará dicho lenguaje.
- Board support package (bsp) a manejar. Este paquete funciona como una capa de abstracción del hardware, dotándonos de una serie de librerías para manejar los diferentes módulos de la placa de forma sencilla, sin necesitar conocer de forma extensa el hardware de la placa.

Una vez generado el proyecto, se podrán crear los ficheros .c, .h y las bibliotecas que se consideren necesarias, para implementar los algoritmos anteriormente descritos.

Como se puede apreciar, la placa ZC702 es una placa comercial suministrada por el fabricante (Xilinx). Por lo tanto, el archivo con la configuración hardware vendrá integrada en SDK y no será necesario crear dicho archivo. Por otro lado, para utilizar los dos cores del procesador, deben de crearse dos proyectos, cada uno asignado a un core la CPU. Además, para que los dos cores funcionen a la vez, la dirección de memoria donde se guardarán ambos programas deben ser distintas entre sí. Por último, para que la aplicación corra con los dos cores, es necesario crear una "Run Configuration" específica que ejecute ambos cores a la vez.

A continuación, se explicará en detalle las bibliotecas y módulos utilizados para configurar las diferentes funciones de los algoritmos explicados.

5.3.2 Memoria Compartida

La implementación de la memoria compartida es muy fácil. A cada core se le asignará una zona de memoria, en hexadecimal, que será la misma para ambos cores. Además, el entorno de trabajo tiene una librería instalada por defecto que nos permite escribir o leer en el área de memoria compartida seleccionada mediante una serie de funciones.

5.3.3 Librería PMC

Para poder obtener el valor de los PMC elegidos se usará la biblioteca, desarrollada internamente por IKERLAN, “pm_mon”.

Esta biblioteca tiene varias funciones: una función “set” que con una matriz de 6 enteros de 32 bits, que son los identificadores de los PMC, pone a cero el valor de dichos PMC. Para después activarlos y que se pongan a contar los eventos hardware asociados a cada PMC. Para recoger el valor de dichos PMC, se usará una función “get”, que pasándole una array de 6 elementos de 32 bits nos dará el valor de los PMC escogidos anteriormente.

5.3.4 Librería de Monitorización

Para poder monitorizar el consumo de potencia, voltaje y temperatura, se usará la librería “Monitoring”, que ha sido desarrollada por IKERLAN.

Esta biblioteca nos permite monitorizar muchos parámetros de la placa, entre ellos la potencia consumida por la CPU, su voltaje y la temperatura de la placa.

Para poder monitorizar la placa se seguirán los siguientes pasos:

- Crear una instancia del tipo de variable que nos proporciona la biblioteca y que guardará los datos de monitorización.
- Llamar a la función “getChanelInfo”, cuyo parámetro es la variable creada en el paso anterior. Esta función es la que monitoriza la placa y obtiene los datos sobre consumo de potencia, voltaje y temperatura.
- Luego se recogerán cada uno de esos datos en variables separadas.

Una cosa importante de esta librería es que para obtener la lectura de la potencia de los sensores externos al Zynq colocados en la placa ZC702, dicha información tiene que pasar por un bus de comunicación (PMBus). Este fenómeno genera una latencia en la monitorización, provocando que se tarde 1,2 ms en monitorizar la placa.

5.3.5 Tarjeta SD

Para poder abrir ficheros en la tarjeta SD y escribir en ellos con la placa ZC702. SDK nos ofrece la biblioteca “xilffs”, la cual nos permite el manejo de archivos “FAT”, un formato de archivos que utilizan las tarjetas SD. Para saber cómo funciona dicha librería puede consultar [24].

La librería nos proporciona una serie de funciones de alto nivel que nos permiten trabajar con la SD, sin tener que programar en bajo nivel el hardware de la tarjeta. Para configurar la SD y poder crear ficheros de texto en los que escribir datos, se harán los siguientes pasos:

- Se llamará a la función que monte la tarjeta SD en la placa para que funcione.
- Se invocará a la función que permite crear un sistema de archivos, donde se pueda crear y/o abrir archivos.
- Se creará un nuevo archivo de texto en el que se pueda escribir datos.
- Cuando sea pertinente, se llamará a una función que escriba los datos que se desean guardar en la tarjeta SD.

- Una vez se acaben de realizar las operaciones pertinentes, se cerrará el archivo de texto.

Para escribir los datos se usará una función que permite escribir datos en ficheros almacenados en la tarjeta. Lo único, es que está función por defecto está deshabilitada, para habilitarla se seguirán los siguientes pasos:

- Dentro de la Bsp del proyecto a ejecutar y mirando en la carpeta que nos indica el núcleo de la CPU a usar.
- En ese directorio se buscará la carpeta nombrada “include”.
- Dentro de esa carpeta, se buscará el archivo “xparameters.h”.
- En ese archivo se buscará la constante “FILE_SYSTEM_USE_STRFUNC” y se cambiará su valor a 2.

Si no se realizan los pasos mencionados el compilador no encontrará dicha función y el programa no funcionará.

5.3.6 TTC e interrupciones

Para el TTC y las interrupciones, el entorno de trabajo proporciona una serie de drivers y funciones que permite configurar dichos módulos en alto nivel, sin tener que conocer en profundidad el hardware de la placa. En los siguientes párrafos se explicará como configurar las interrupciones y el timer para hacer lo especificado en los algoritmos.

5.3.6.1 Configuración interrupciones

En SDK todos los módulos de la placa (tarjeta SD, Ethernet, TTC, interrupciones) tienen una serie de bibliotecas y ejemplos que nos permiten manejar y configurar dicho modulo de manera fácil y rápida. Además, en el archivo “xparameters.h” están definidos una serie de parámetros que nos indican cómo se puede configurar dicho módulo. Para este proyecto, los pasos a realizar son:

1. Hacer una instancia del driver de interrupciones.
2. Inicializar el driver que controla las interrupciones.
3. El driver debe de conectarse al hardware de ARM que maneja las interrupciones.
4. Habilitar las interrupciones en ARM.

Todas estas acciones van a ser realizadas mediante funciones que nos proporciona sdk.

5.3.6.2 Configuración del TTC

El timer a usar dentro de este módulo se puede configurar de tres maneras: el timer llega a un intervalo temporal, el timer se sobrecarga y 2 timers del TTC alcanzan el mismo valor. Para los algoritmos del apartado 5.1.1, 5.1.2 y 5.2.1, se requiere que el timer genere una interrupción tras un intervalo regular de tiempo. Por ende, el timer ha sido configurado en modo intervalo, ya que es la opción que más se ajusta a lo explicado en los apartados 5.1.1, 5.1.2 y 5.2.1.

El TTC, al igual que las interrupciones, dispone de un driver y una batería de funciones que simplifican la configuración y arranque de dicho módulo. Para configurar el timer del TTC se llevarán a cabo estas acciones:

1. Hacer una instancia del driver.
2. Con su correspondiente función, indicar que timer se va a utilizar el y configurarlo en modo intervalo.
3. El TTC dispone de una función que dada la frecuencia del intervalo calcula automáticamente el prescaler del timer y el valor al que el timer debe de activar la interrupción.
4. Después, mediante una llamada a función, se asocia el driver de interrupción con la interrupción por intervalo del timer y se le pasa la rutina de interrupción a ejecutar.
5. Activar el timer para que se ponga a contar.

Otro aspecto positivo de esta configuración. Es que cuando salte la interrupción del timer, este se reiniciará automáticamente y volverá a contar desde cero, permitiendo que la interrupción salte de manera cíclica, tras un intervalo regular de tiempo.

Por último, y teniendo en cuenta las restricciones temporales explicadas en el apartado 5.1.1, la frecuencia a la que salta la interrupción del timer, se corresponde con el tiempo de muestro de los PMC. En la Tabla 5-3, se indicarán las frecuencias y tiempos de muestreo seleccionados.

Tiempo de muestreo de los PMC (ms)	Frecuencia de muestreo de los PMC(Hz)
1000	1
500	2
333	3
250	4
100	10

Tabla 5-3 Selección de los periodos y frecuencias de muestreo.

Como se puede apreciar en la Tabla 5-3, el mínimo tiempo de muestreo para los PMC es de 100ms. Si se eligiera un tiempo de muestreo menor, no se cumplirían las restricciones temporales del apartado 5.1.1 y el modelo no sería capaz de estimar correctamente la potencia.

5.3.7 Organización de los datos

Como se mencionó en el apartado 5. Todos los algoritmos al ejecutarse, generan un fichero de texto, donde se encuentran una serie de datos. Estos ficheros son muy importantes, ya que con sus datos se generarán los modelos de potencia para estimar la potencia de 1 y/o dos cores de la CPU. Asimismo, cada fichero de datos puede tener unos de estos tres propósitos, que se explicarán inmediatamente:

- **Entrenamiento**: Estos ficheros de datos serán los encargados de entrenar el modelo que nos permite estimar la potencia y con el que se obtendrán los coeficientes del mismo.

- **Validación**: En este grupo están los ficheros de datos que ejecutan la misma secuencia de benchmarks que la de los ficheros de entrenamiento y nos permite conocer el error y sensibilidad del modelo ante cargas de trabajo similares.
- **Test**: Este tipo de ficheros ejecuta una secuencia de benchmarks diferentes a la de los ficheros de entrenamiento y nos permite saber cómo se comporta el modelo ante diferentes cargas de trabajo a las que no ha sido entrenado.

Aparte, los ficheros tendrán una codificación u otra, en función del algoritmo, la frecuencia de muestreo y la secuencia de benchmarks utilizada. Dicha codificación, puede observarse en los ficheros de datos anexos a este proyecto.

Finalmente, puede suceder que en los ficheros de texto haya que corregir la potencia estimada por alguno de los algoritmos, debido a los siguientes motivos:

- En el algoritmo del apartado 5.1.2, se estima la potencia de 1 core usando 2 cores. Al tener encendido los dos cores, el consumo medido no se corresponde con el consumo de 1 solo núcleo de la CPU. Así que habrá que corregir la potencia estimada del fichero de texto para que mida la potencia de 1 solo core. Para corregirlo, se medirá el consumo de un core en vacío y luego lo mismo para dos cores. Entre ambas medidas, habrá una diferencia en el consumo de potencia, que son aproximadamente unos 70mW. Lo que se hará es extrapolar esa diferencia a todos los benchmarks ejecutados. Por lo tanto, en los ficheros generados con el algoritmo del apartado 5.1.2, habrá que restar esa diferencia de 70mW a la columna de la potencia estimada.
- Por otro lado, en el algoritmo del apartado 5.1.1. se ha comprobado que no ejecutando nada en la CPU existen diferencias en la potencia consumida, si esta se mide con o sin interrupción. En este caso la diferencia es de 30mW y aplicando la estrategia del párrafo anterior, se restará dicha diferencia de potencia a los datos generados con dicho algoritmo.

5.3.8 Impresión por pantalla

Como se puede suponer, cuando se implemente y se prueben cada uno de los algoritmos diseñados, el usuario necesita conocer cuál es el estado del proceso. Por ello, se enviarán mensajes en forma de impresiones de pantalla, a través de la UART.

Los mensajes que pueden escribirse en la pantalla, en caso de que todo vaya bien, son:

- La sd ha sido montada, es decir se crea la unidad lógica y el sistema de ficheros.
- La configuración del TTC y de las interrupciones se ha realizado sin problemas.
- Cada vez que se entre en la interrupción.
- Cuando el algoritmo haya terminado de ejecutarse.

Por otro lado, los mensajes de error que pueden aparecer son los siguientes:

- Errores con la SD (No se puede montar la SD y/o sistema de archivos, problemas en abrir y/o escribir el fichero).
- La interrupción y/o TTC han fallado a la hora de ejecutarse.

Además cuando se esté trabajando con los algoritmos que permiten estimar la potencia de 2 cores, pueden aparecer los siguientes mensajes:

- La potencia estimada por los PMC de ambos cores.
- El valor y nombre de los PMC del core 1.

5.4 Implementación del modelo y diseño de los métodos para la validación y testeo

En este apartado, utilizando Python y con los ficheros de texto obtenidos en los diferentes algoritmos implementados, se implementarán, validarán y testearán, los diferentes modelos de potencia para 1 y 2 cores, explicados en apartados anteriores.

Para alcanzar dichos objetivos, se han diseñado una serie de clases en Python con el objetivo de implementar los diferentes modelos de potencia. A continuación en la Ilustración 5-13, se mostrará un diagrama de clases con las relaciones entre todas las clases diseñadas. Subsiguientemente, se explicará cada clase por separado en profundidad.

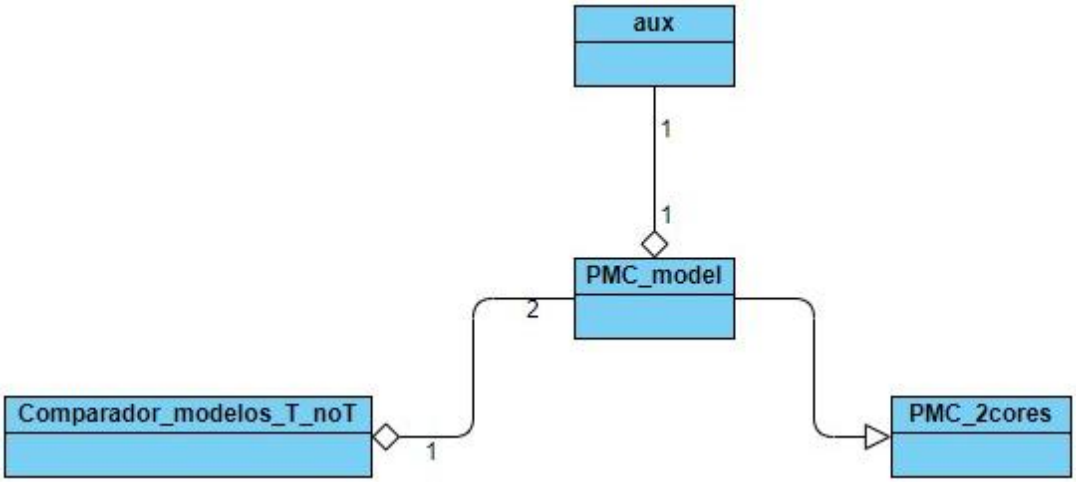


Ilustración 5-13 Diagrama de clases simplificado

Antes de entrar en detalle en cómo son las diferentes clases diseñadas, se hará un pequeño resumen de cada clase y como están relacionadas entre sí:

- Aux: Esta clase engloba todos aquellos métodos que son usados de manera interna en el resto de métodos del resto de clases.
- PMC_model: Esta clase implementará el modelo de potencia para estimar el consumo de 1 core y aquellos métodos que nos permitan evaluar la precisión y sensibilidad del modelo diseñado.
- PMC_2cores: El propósito de esta clase es implementar el modelo que estima la potencia de 2 cores de la CPU.
- Comparador_modelos_T_noT: Esta clase sirve para comparar el error y la sensibilidad, de dos modelos diferentes que estiman la potencia de 1 core.

Como se puede observar en la Ilustración 5-13,la clase “PMC_model” nos permite estimar la potencia de 1 core de la CPU y “PMC_2cores” estima la potencia de 2 cores de la CPU. Es lógico que “PMC_2cores” derive de “PMC_model”. Por otro lado, la clase “Comparador_modelos_T_noT”, compara dos modelos, que estiman la potencia de 1 core, entre sí. Así que parece lógico, que la clase “PMC_model” esté agregada dentro de ella. Por último, se decidió que en los métodos de las clases solo estuviesen incluidos aquellos métodos que permitiesen construir el susodicho modelo y evaluar sus propiedades. Por lo tanto, todos aquellos métodos que son utilizados de forma interna por los métodos principales de ambas clases, se han encapsulado en la clase “aux”. Para que las clases “PMC_model” y “PMC_2cores” puedan utilizar dichos métodos, la clase “aux” ha sido agregada dentro de la clase “PMC_model”.

En los siguientes subapartados, solo se explicarán los aspectos más importantes de cada clase. Si desea conocer en profundidad el funcionamiento de las diferentes clases, consulte el código Python de dichas clases o el anexo “anexo_clases_Python.docx”.

5.4.1 Clase aux

Esta es la clase base, de la que emanan el resto de clases. Su principal objetivo es encapsular los métodos que son utilizados, de manera interna, en el resto de clases o funciones auxiliares. En consecuencia, esta clase no tiene ningún constructor ni parámetros. En la Ilustración 5-14, se pueden ver los métodos de esta clase.

Ilustración 5-14 Métodos de la clase aux

Esta clase tiene tres propósitos principales:

- Obtener la información de los ficheros de texto, como el valor de los PMC, la potencia consumida, el voltaje, la temperatura y el tiempo de muestreo por cada muestra tomada.

- Generar, a partir de los datos obtenidos por los diferentes ficheros de texto, las matrices *H* y *P*, del apartado 4.4. Pudiendo implementar el correspondiente modelo de potencia y obtener así los coeficientes *α* de dicho modelo.
- Métodos auxiliares.

Para el primer propósito, se utilizarán los métodos de "leer_datos" y "leer_potencia_PCB". El primero, extrae el valor de los PMC, índice benchmark, potencia consumida, voltaje, temperatura y tiempo de muestreo, para cada muestra tomada. Luego con dichos datos construye la correspondiente matriz *H* y *P*. Además, también se obtiene una matriz con la potencia media del benchmark ejecutado en cada muestra. El segundo método, simplemente se saca del fichero la potencia estimada por el modelo durante la ejecución del algoritmo. Este campo se recoge aparte, ya que puede haber ficheros donde ese campo no exista y pueden aparecer errores, si se leen datos de una posición inexistente en el fichero de datos.

Por otro lado, el método "leer_datos" ya genera las matrices *H* y *P* del correspondiente modelo de potencia. Aún así, solo genera dichas matrices para los datos de un determinado fichero de texto. Si se deseará generar una matriz *H* y *P*, que contenga los datos de varios ficheros de texto, se debe de llamar al método "return_H_pot". Para finalizar, si se desea obtener los coeficientes *α* del correspondiente modelo de potencia, se debe de invocar a método "entrenar_modelo".

Lo métodos auxiliares sirven para un diverso conjunto de aplicaciones, tales como: Encapsular cierto tipo de operaciones, ayudar a representar gráficamente un conjunto de datos, sacar la media y la desviación típica de un conjunto de datos u obtener el error relativo en porcentaje, para cada muestra, entre la potencia estimada y la potencia real o media.

5.4.2 Clase PMC_model

En esta clase encapsularía el modelo que nos permite estimar la potencia de un core de la CPU y los diferentes métodos que nos permiten evaluar la precisión y sensibilidad del modelo. Como se puede observar en la Ilustración 5-13, esta clase es hija de la clase "aux". Permitiendo así que esta clase pueda usar los métodos de la clase "aux" sin problemas. En la Ilustración 5-15, se puede observar los atributos y métodos de esta clase, en la cual todos sus atributos son privados y los métodos son públicos

Como se muestra en la Ilustración 5-15, todos los métodos de esta clase tienen un único atributo, que sirve para seleccionar que conjunto de datos van a ser utilizados en el respectivo método (entrenamiento más validación o test), mediante un 1 o un 0. Además, si se invoca la función sin asignar un valor a dicho argumento, por defecto el método usará los datos de entrenamiento más validación.

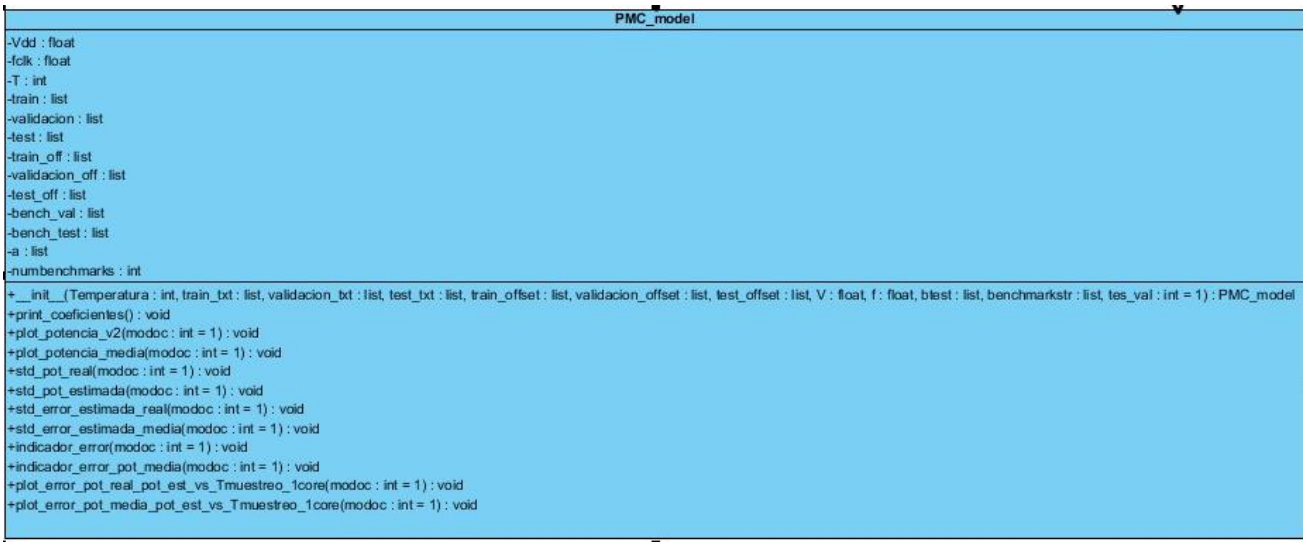


Ilustración 5-15 Métodos y atributos de la clase PMC_model

A continuación se procederá a explicar cada uno de los atributos de la clase "PMC_model":

- Vdd: Nos indica la tensión de alimentación de la CPU
- Fclk: Almacena la frecuencia de reloj de la CPU.
- T: Variable booleana que sirve para indicar si el modelo va a tener corrección por temperatura o no. Esta opción nunca va ser utilizada en este proyecto, pero se ha implementado por si quiere utilizarse posteriormente.
- Train: Es una lista que almacena el nombre de todos los ficheros, con los que se entrenará el modelo de potencia.
- Validación: Es una lista que almacena el nombre de todos los ficheros, con los que se validará el modelo de potencia.
- Test: Es una lista que almacena el nombre de todos los ficheros, con los que se testeará el modelo de potencia.
- Train_off: Es una lista que almacena el factor de corrección potencia, para todos los ficheros con los que se entrenará el modelo de potencia.
- Validación_off: Es una lista que almacena el factor de corrección potencia, para todos los ficheros con los que se validará el modelo de potencia.
- Test_off: Es una lista que almacena el factor de corrección potencia, para todos los ficheros con los que se testeará el modelo de potencia.
- Bench_val: Lista con los nombres de los benchmark usados para el entrenamiento y validación del modelo.
- Bench_test: Lista con los nombres de los benchmark usados para el testeo del modelo.
- a: Lista de flotantes que contiene los coeficientes a_i del modelo de potencia para un core, mostrado en la Ecuación 4-1.
- Numbenchmarks: Variable de tipo entero, que nos muestra el número de benchmarks de entrenamiento usados.

Ahora se explicará de manera resumida que hacen los diferentes métodos de cada clase.

Por un lado, se encuentra el constructor de la clase, el cual se llama "__init__" ya que así lo exige Python. Lo primero que se le hace es pasarle todos los ficheros, sus correspondientes factores de corrección de la potencia, el voltaje, la frecuencia de la CPU y el nombre de las secuencias de benchmarks a utilizar, por parámetro. Luego con los datos de entrenamiento se genera la matriz "H" y "P" del modelo de potencia y con estos se obtienen los coeficientes *a* del modelo

Por otro lado, esta clase dispone de un método que nos devuelve e imprime por pantalla los coeficientes del modelo. El resto de métodos sirven para ver las diferencias que hay entre la potencia estimada y la potencia real medida o la potencia media por benchamrk. Hay métodos como "plot_potencia_v2" y "plot_potencia_media" que nos permiten ver de manera gráfica cómo se comporta la potencia estimada frente a la real o media, para un conjunto de ficheros de datos. Otros nos permiten ver la media y desviación típica de diversos parámetros (error potencia estimada-real, error potencia estimada-media) y estos pueden verse como el error medio por benchmark de cada fichero o de un conjunto de ellos. También existen métodos para obtener el error relativo total, entre la potencia estimada y la real o la potencia estimada y la media, de un determinado conjunto de datos. Este último dato será utilizado como parámetro para evaluar la precisión del modelo. También, se han implementado una serie de métodos que nos permiten ver el error relativo medio, entre la potencia estimada y la real o la potencia estimada y la media, de un conjunto de datos, en función del tiempo de muestreo utilizado.

5.4.3 PMC_2cores

Esta clase encapsula el modelo que nos permite estimar la potencia de 2 cores de la CPU y los métodos para comprobar la sensibilidad y precisión del modelo. Como se puede ver en la Ilustración 5-13, esta clase deriva de la clase "PMC_model". Además, los métodos de esta clase son muy similares a los de dicha clase. La única diferencia reside en cómo se obtiene la potencia estimada, a que esta será extraída de los ficheros de texto, que se pasan por parámetro al objeto.

En la Ilustración 5-16, se puede ver los atributos y métodos de dicha clase, cuyos nombres son muy similares a los de "PMC_model" para mostrar la similitud en su ejecución. Si desea ver en detalle cómo funcionan estos métodos, consulte el código fuete anexo.

Al igual que en el apartado 5.4.2, todos los métodos de esta clase dispondrán de un único argumento que nos permita seleccionar que conjunto de datos mostrar (entrenamiento más validación o test).

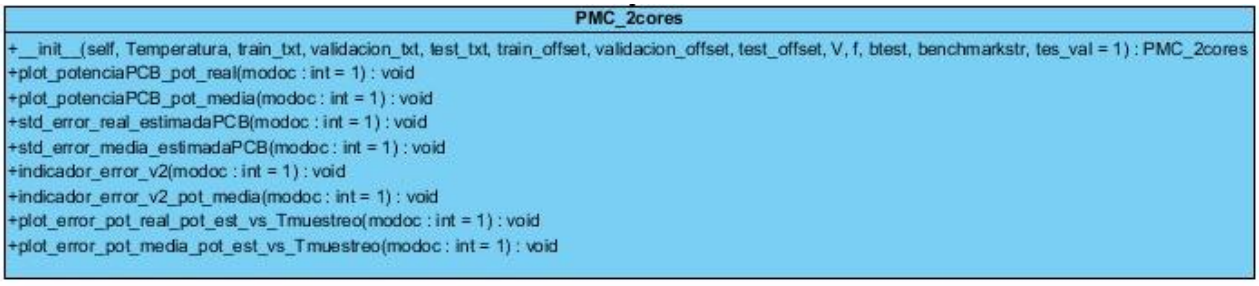


Ilustración 5-16 métodos de la clase "PMC_2cores"

5.4.4 Comparadaor_modelos_T_noT

Esta clase nos permite comparar de manera gráfica dos modelos de potencia, que estiman la potencia de 1 solo core de la CPU. Ambos modelos utilizarán el mismo set de datos para la validación y entrenamiento. La diferencia radica en que cada modelo utilizará una combinación de datos de entrenamiento diferente, generando así un modelo diferente. Al igual que pasa con la clase "PMC_2cores", los métodos implementados en esta clase son muy similares a los de la clase "PMC_model". La diferencia entre ambos métodos es que ahora se debe estimar la potencia de dos modelos, en vez de uno y ambas combinaciones de valores deben de mostrarse gráficamente a la vez. Es decir, cuando se lea un fichero de texto, se sacarán sus correspondientes datos y luego se calculará la potencia estimada para cada modelo. Si desea ver en detalle cómo funcionan estos métodos, consulte el código fuente anexo.

En la Ilustración 5-17 , se puede observar los métodos y atributos de la clase. Al igual que en el apartado 5.4.2, todos los métodos de esta clase dispondrán de un único argumento que nos permita seleccionar que conjunto de datos mostrar (entrenamiento más validación o test).

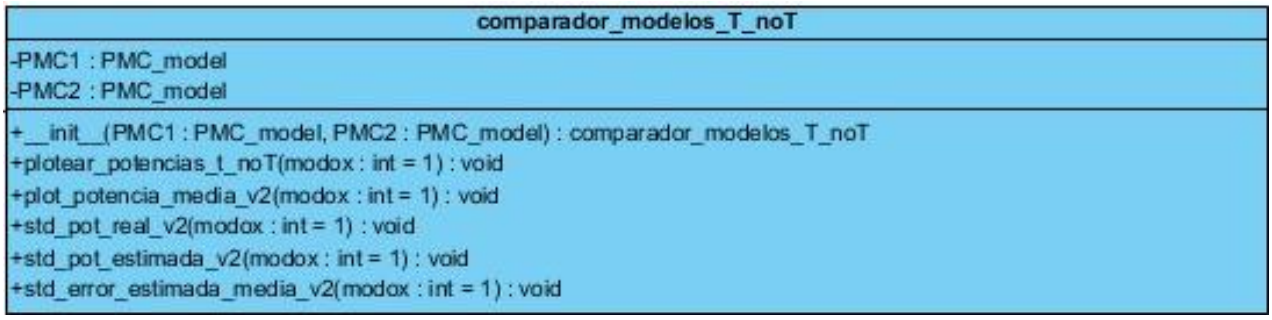


Ilustración 5-17 Métodos y atributos de la clase "comparador_modelos_T_noT"

5.5 Metodología para medir las propiedades del modelo para un determinado set de datos

En el apartado anterior, se ha explicado como implementar cada modelo en las diferentes clases diseñadas y cómo funcionan los métodos que nos permiten saber y comparar las propiedades del modelo. En este apartado se explicará la metodología para poder medir dichas propiedades, para un determinado conjunto de datos, que será la combinación de uno o más ficheros de texto.

Las dos propiedades más importantes del modelo de potencia son la precisión y la sensibilidad. A continuación, se explicará en detalle porqué se han escogido esas propiedades y como se medirán cada una de ellas.

5.5.1 Sensibilidad

La sensibilidad es la capacidad que tiene el modelo de detectar un cambio de magnitud en la potencia consumida. En este caso la sensibilidad del modelo debe detectar la variación de potencia al cambiar de benchmark.

En este proyecto se supone que el consumo de potencia durante la ejecución de un benchmark es más o menos constante. Así que la potencia consumida, en teoría, solo variará al cambiar de benchmark y los modelos a diseñar deben ser capaces de estimar dicha variación en el consumo de potencia y en el mismo sentido. Por otro lado, también se le exige al modelo que durante la ejecución de un benchmark la potencia estimada sea más o menos constante y filtre el posible ruido asociado a la medida de la potencia real. Además, la detección debe realizarse de manera rápida, para evitar retrasos entre la potencia estimada y la real. En consecuencia, la sensibilidad del modelo debe ser lo suficientemente alta para detectar la variación en el consumo de potencia al cambiar de benchmark y lo suficientemente baja para no detectar, en la medida de lo posible, el ruido que pueda tener la medida de la potencia real.

En [3, 4] se utilizan algoritmos de detección de fase para medir y cuantificar la sensibilidad del modelo. Este método nos permite automatizar el proceso para medir y cuantificar la sensibilidad del modelo, permitiéndonos medir y cuantificar la sensibilidad de manera automática, y comparar fácilmente la sensibilidad entre modelos generados con distintos datos de entrenamiento. El problema es que utilizar este tipo de algoritmos, requeriría un tiempo para el aprendizaje, diseño e implementación extenso. Así que se ha descartado esta opción y se optará por una opción cuya implementación y evaluación de la sensibilidad es más fácil, aunque esta sea evaluada de forma cualitativa y no se pueda automatizar. En los siguientes párrafos, se explicará cómo funciona el método diseñado, sus ventajas, desventajas y el porqué de su elección.

La opción a implementar es la siguiente: para cada modelo implementado, da igual si estima la potencia de 1 o 2 cores de la CPU, se llamará a su correspondiente método que permite ver gráficamente, para cada fichero de datos utilizado, la potencia media consumida en comparación con la potencia consumida estimada. Se ha escogido la potencia media en vez de la potencia real, debido a que la medida de potencia real en la placa ZC702tiene una componente de ruido relevante y por tanto no se observa muy bien el cambio en la potencia consumida al alternar de tarea. Ahora el usuario debe comprobar, manualmente y para cada gráfica, que está asociada a un fichero de texto, si al variar la potencia real al cambiar de benchmark, la potencia estimada varía en la misma dirección. En este caso, solo nos importa si la potencia estimada y la real varían en el mismo sentido, el cuanto de dicha variación no es importante para evaluar esta propiedad.

Otro aspecto importante a tener en cuenta al evaluar la sensibilidad del modelo, es la oscilación de la potencia calculada durante la ejecución de un benchmark. Para que el modelo realice un buen seguimiento durante la ejecución de un benchmark, se debe cumplir lo que se plasma de manera gráfica en la Ilustración 5-18 y la Ilustración 5-19.

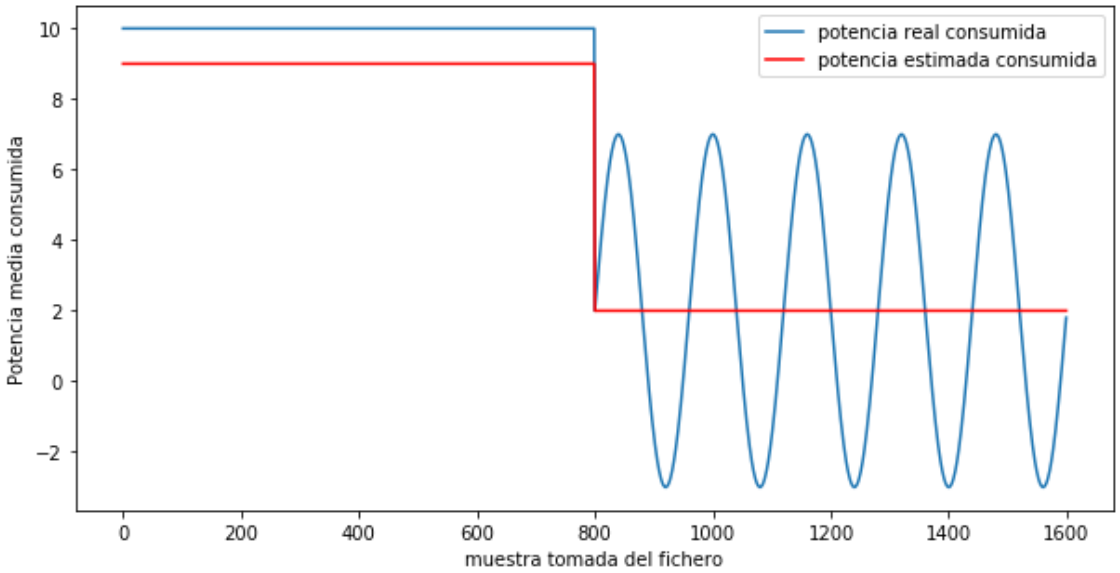


Ilustración 5-18 Ejemplo gráfico del grado de oscilación aceptable de la potencia estimada cuando se está ejecutando un benchmark.

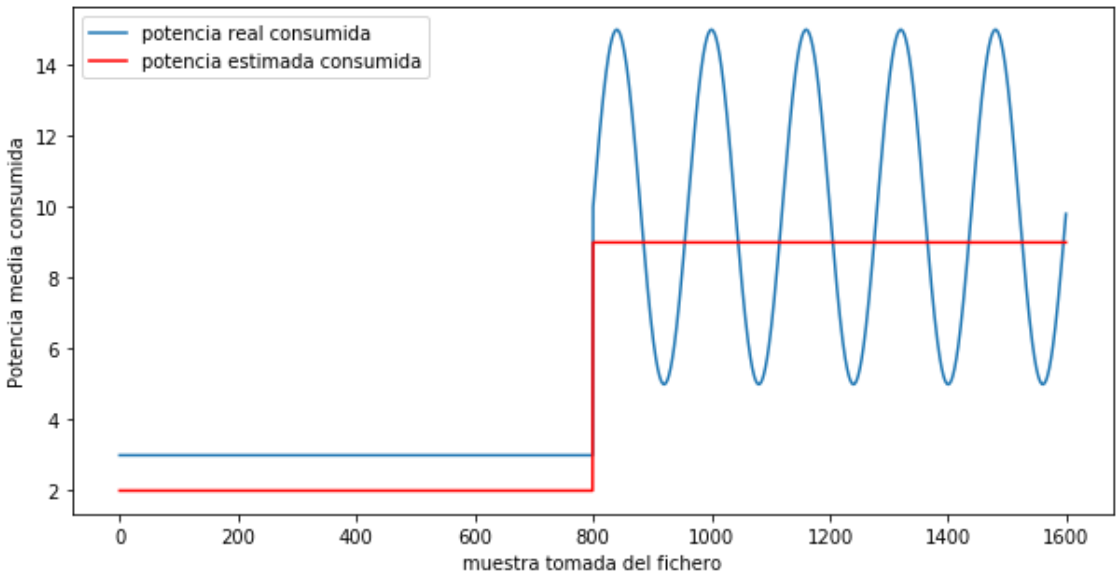


Ilustración 5-19 Otro ejemplo gráfico del grado de oscilación aceptable de la potencia estimada cuando se está ejecutando un benchmark.

Ahora se explicará qué criterios se usan para indicar si la sensibilidad del modelo es buena mala o regular. Por un lado, en cada fichero se debe comprobar si al cambiar de benchmark y variar la potencia real, la potencia estimada varía en el mismo sentido. Por el otro, se debe mirar que cuando se esté ejecutando el benchmark, la potencia estimada por el modelo cumpla lo mostrado en la Ilustración 5-18 y la Ilustración 5-19. Si en algún benchmark de algún fichero no se cumplen dichas condiciones, se considerará como fallo y en función del número de fallos que tenga cada fichero se indicará si ese fichero tiene una sensibilidad buena, mala o regular. Una vez analizada la sensibilidad de todos los ficheros, dependiendo del número de ficheros con sensibilidad mala o regular, se dictará si el modelo tiene una sensibilidad buena,

mala o regular. En la Tabla 5-4 se puede ver los criterios para evaluar la sensibilidad de un fichero y en la Tabla 5-5 lo mismo, pero aplicado al modelo.

Nº de fallos en el fichero, en función del número de benchmarks	Sensibilidad del fichero
[0%,25%]	Buena
(25%,50%)	Regular
[50%,100%]	Mala

Tabla 5-4 Criterios para evaluar el seguimiento de la potencia real vs la potencia estimada para un conjunto de datos almacenados en un fichero

Porcentaje de ficheros malos y/o regulares	Sensibilidad del fichero
[0%,25%] ficheros regulares o [0%,17%] ficheros malos	Buena
(25%,33%) ficheros regulares o (17%,25%) ficheros malos	Regular
[33%,100%] ficheros regulares o [25%,100%] ficheros malos	Mala

Tabla 5-5 Tabla que nos permite evaluar la sensibilidad del modelo en función del tipo de ficheros usados para el entrenamiento, validación y/o test del modelo

Este método nos proporciona las siguientes ventajas: El método es de muy fácil implementación, ya que no requiere aplicar nada más, lo que hace que el método sea muy rápido de implementar.

Por otro lado, el método es rápido, solo si el número de muestras a analizar es pequeño. Si el número de muestras es muy grande, el proceso se vuelve muy largo. Además, como la evaluación se realiza de manera manual, existe el riego de que se comentan errores al evaluar dicha propiedad. Por último, la medida de esta propiedad es cualitativa y no cuantitativa. Es decir, solo se comprobará si el modelo de potencia detecta bien cambios importantes en el consumo de potencia real. No se puede determinar qué modelo es mejor, si ambos modelos tienen una sensibilidad buena.

Debido a que este método no requiere hacer ninguna operación complementaria y nos permite evaluar la sensibilidad del modelo para los objetivos mencionados al inicio de este apartado. Se utilizará esta metodología para comprobar la sensibilidad del modelo.

5.5.2 Precisión

En este apartado se explicará cómo se evaluará la precisión de los diferentes modelos implementados. Como se puede suponer la precisión de un modelo será medida y evaluada mediante su error relativo medio entre la potencia estimada y la potencia real o media consumida para un conjunto de ficheros (entrenamiento más validación o test). El cálculo del error relativo se muestra en la Ecuación 5-4, en la que *err* es el error relativo en %, *P* puede ser

la potencia media o la real y P_e es la potencia estimada por el modelo. Si se quiere calcular el error medio total, solo es necesario hacer la media de todos los errores calculados.

$$err = \left|\frac{P - P_e}{P}\right| * 100$$

Ecuación 5-4 Cálculo del error relativo

Como se puede observar, todas las clases donde se puede implementar un modelo de potencia. Poseen una serie de métodos para obtener el error medio relativo total para un conjunto de datos y el error medio que tiene cada benchmark ejecutado. Dicho error puede ser la diferencia entre la potencia real y la potencia estimada o el entre la potencia media y la potencia estimada. En este caso, se obtendrá el error relativo medio para ambos casos y se comparará el error de distintos modelos entre sí.

6 RESULTADOS

En este apartado se comentarán los diferentes resultados obtenidos en este modelo, se mostrará la precisión, sensibilidad y coeficientes para cada modelo generado. Luego, se explicarán dichos resultados en mayor detalle, dependiendo del tipo de datos usados para validar o testear el modelo.

En esta memoria se resaltarán solo los resultados más importantes. Si se desea obtener más información sobre el comportamiento de cada modelo, se han anexado una serie de scripts en Python, que muestran toda la información.

6.1 Modelo para estimar la potencia de 1 core, usando el algoritmo de 1 core con interrupción

En este apartado se comentará cuán bien es capaz de estimar la potencia el modelo generado con los datos del algoritmo utilizado. Por lo general, la precisión y sensibilidad del modelo es bastante buena. Los coeficientes de dicho modelo se muestran en la Tabla 6-1.

Coeficiente a_i de la Ecuación 4-1	Valor de los coeficientes
a_1	1.35245597e-01
a_2	-1.45992056e-02
a_3	3.72671471e-01
a_4	-3.69418625e-01
a_5	-4.88265417e-01
a_6	5.38485781e-01
a_7	1.23931182
a_8	2.35028977e-03
a_9	5.90711320e-08

Tabla 6-1 Coeficientes para el modelo que estima la potencia de 1 cores usando los datos generados por el algoritmo 1 core e interrupción

A continuación, para cada conjunto de datos usados se resumirá la precisión y sensibilidad obtenida con su correspondiente conjunto de datos.

6.1.1 Datos de entrenamiento y validación

En este apartado, se comentarán los resultados para este modelo cuando se utilizan los datos de entrenamiento y validación.

6.1.1.1 Precisión

En términos generales, la precisión del modelo es bastante buena ya que el error medio total y por benchmark, medio total y medio por fichero de datos, es relativamente no muy alto. El error relativo suele ser menor o igual al 5%., como se puede ver en la Tabla 6-2.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	3.52
Error relativo medio entre la potencia real y la estimada	2.93

Tabla 6-2 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.1

Respecto al error medio total por benchmark, suelen ser bastante parecidas las variaciones de error entre un benchmark y otro, esto se repite para todos los set de datos a analizar en los diferentes modelos. A continuación, se mostrara un ejemplo gráfico de eso en la Ilustración 6-1.

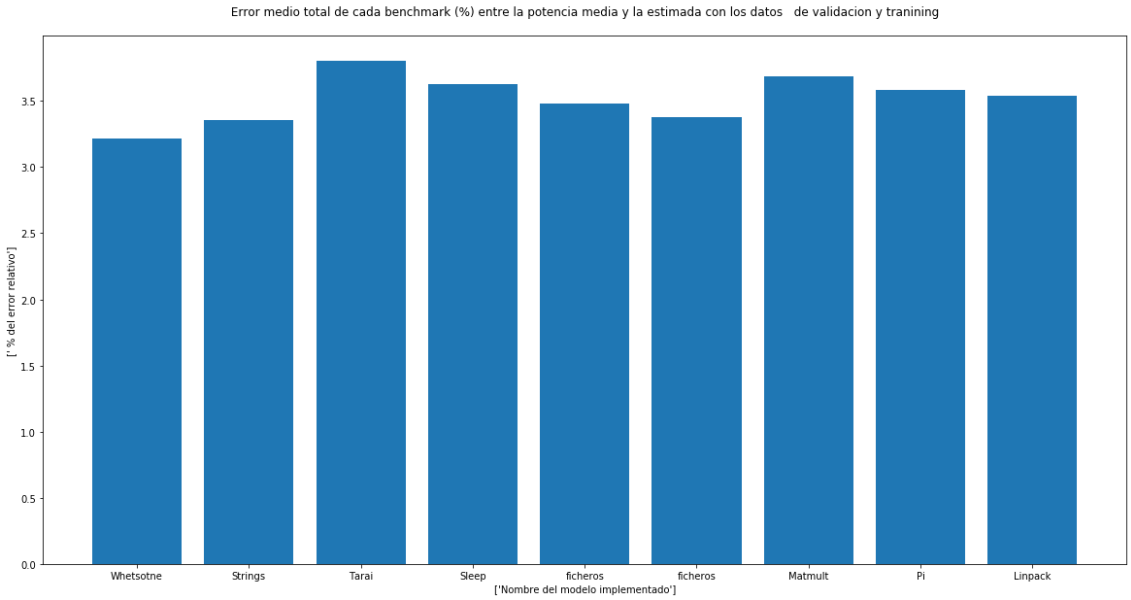


Ilustración 6-1 Error medio de cada benchmark ejecutado para este conjunto de datos dado

Por último, se analizará la evolución del error en función del tiempo de muestreo utilizado. Este se puede ver en la Ilustración 6-2. En dichas gráficas se puede observar que el error es muy parecido en todas, excepto cuando el tiempo de muestreo es de 333ms donde el error es más alto. Esta condición también se cumple para el resto de modelos generados.

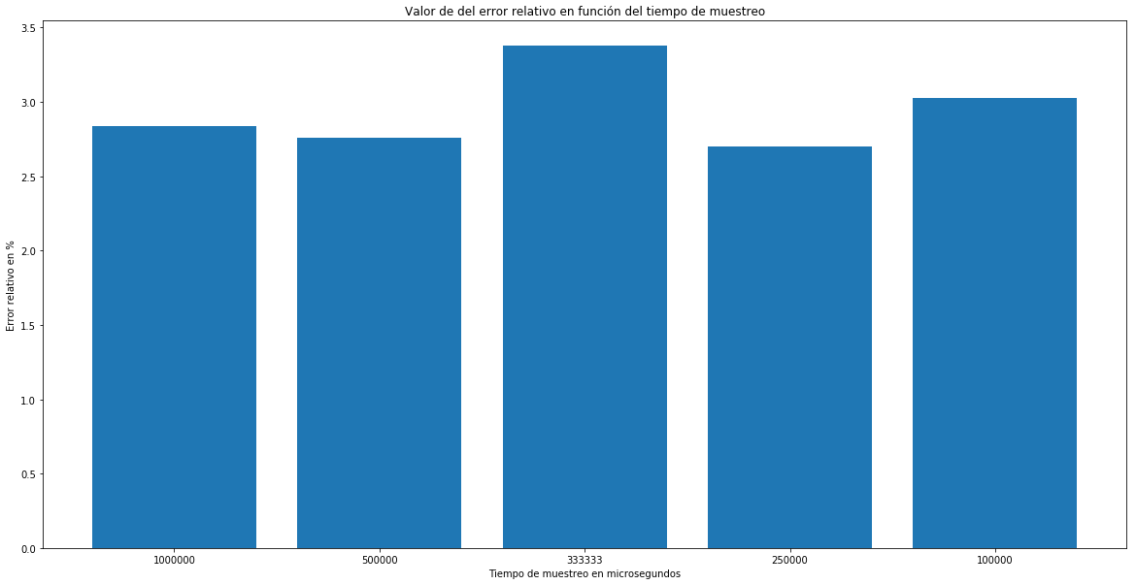


Ilustración 6-2 Error medio en función del tiempo de muestreo para un conjunto de datos dado

6.1.1.2 Sensibilidad

En general la sensibilidad del modelo para este tipo de datos es **BUENA.** El modelo es capaz de detectar una variación grande en el consumo de potencia, al cambiar de benchmark. Además, la potencia estimada filtra bastante bien el ruido de la potencia real. Esto se puede ver de manera gráfica en la Ilustración 6-3. Esto se debe a que durante la ejecución de cada benchmark, el valor de los PMC obtenidos, en cada muestra, son muy similares. En consecuencia, es lógico que la potencia estimada sea un valor más o menos constante durante ese tramo. Un ejemplo de esto se puede ver en la Ilustración 6-4.

Los únicos problemas que se aprecian son los siguientes. Cuando se cambia de benchmark y la variación en el consumo de potencia es muy pequeña, el modelo no es capaz de detectar bien dicho cambio. Por otro lado, según se va reduciendo el tiempo de muestreo, la potencia estimada tendrá cada vez más ruido. Este fenómeno se debe a que como el tiempo de muestreo se va aproximando al intervalo regular de tiempo con el que salta la interrupción, lo estipulado en la Ecuación 5-1 deja de cumplirse al 100%. Provocando que la diferencia de los PMC entre una muestra y otra, habiéndose ejecutado el mismo benchmark, sea cada vez más significativa. Haciendo que la potencia estimada varíe durante la ejecución del mismo benchmark. Este diferencia se puede observarse gráficamente si se comparan la Ilustración 6-4 y la Ilustración 6-5.

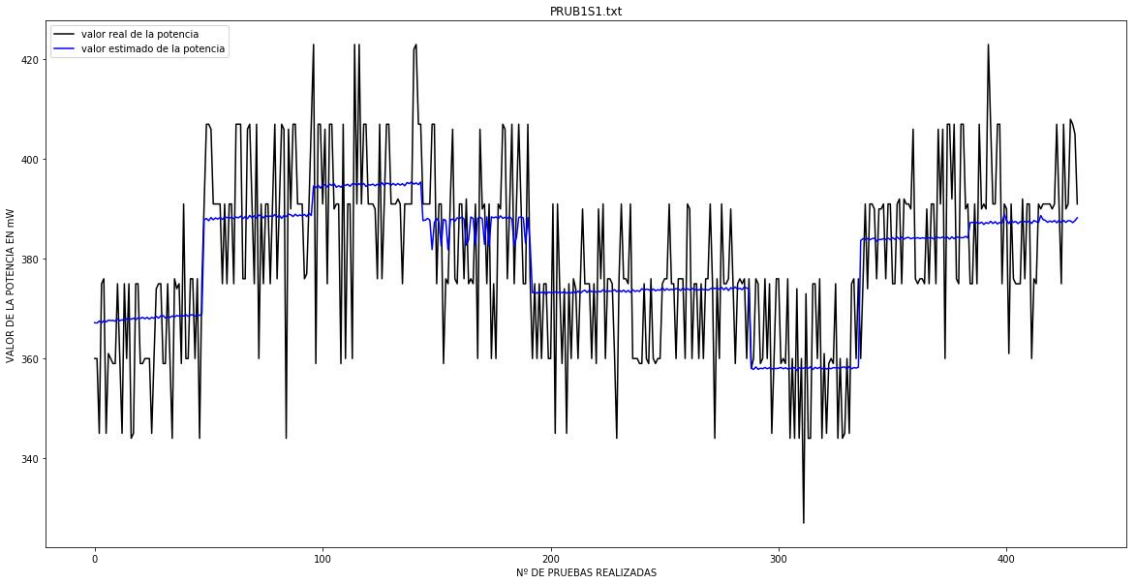


Ilustración 6-3 Evolución de la potencia real frente a la estimada para un cierto fichero de datos utilizado en este modelo

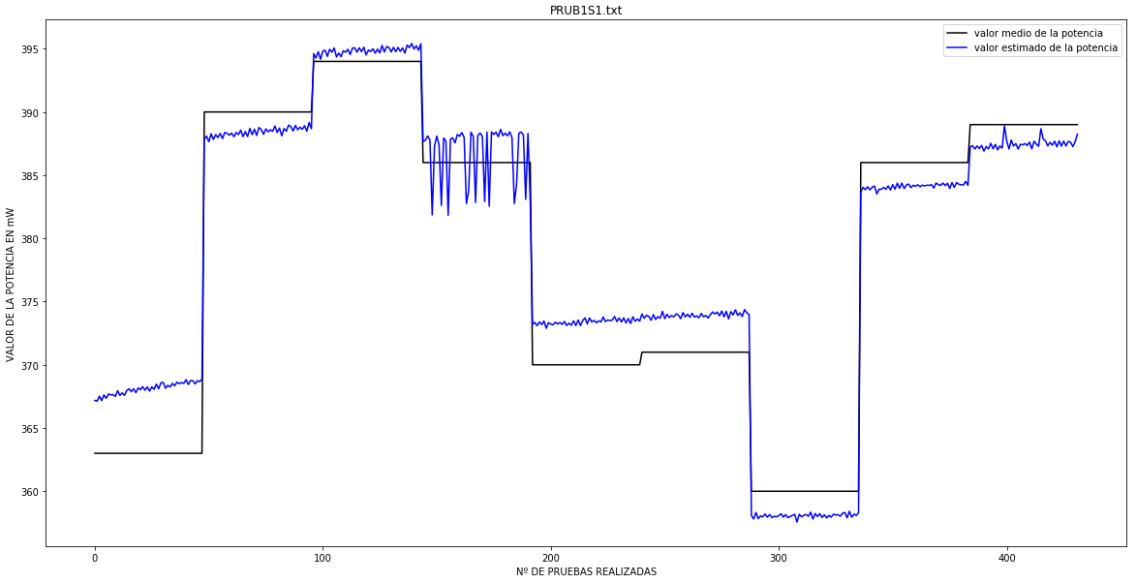


Ilustración 6-4 Evolución de la potencia media frente a la estimada para un cierto fichero de datos utilizado en este modelo, con un tiempo de muestreo de 1s

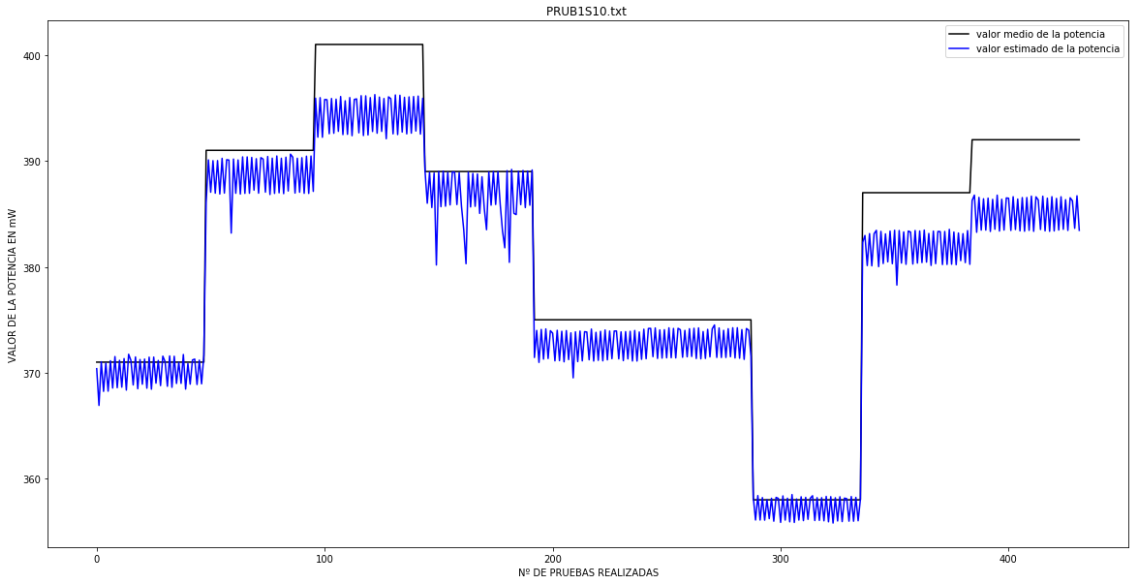


Ilustración 6-5 Evolución de la potencia media frente a la estimada para un cierto fichero de datos utilizado en este modelo, con un tiempo de muestreo de 100ms

6.1.2 Datos de test

En este apartado, se comentarán los resultados para este modelo cuando se utilizan los ficheros de datos de test.

6.1.2.1 Precisión

El error con los datos de test es más alto que el obtenido en los datos de validación, pero la subida es pequeña (en torno al 1,5%) los resultados se pueden ver en la Tabla 6-3.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	4.28
Error relativo medio entre la potencia real y la estimada	5.12

Tabla 6-3 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.1

6.1.2.2 Sensibilidad

Con respecto a la sensibilidad, se puede observar en la Ilustración 6-6 que para estos benchmark la potencia estimada no sigue muy bien a la potencia real. Este se debe a que el consumo de potencia medio entre ambos benchmarks apenas varía. En cambio, el valor de ciertos PMC al cambiar de benchmark si que varía, provocando que la sensibilidad del modelo ante este tipo de cargas no sea muy buena.

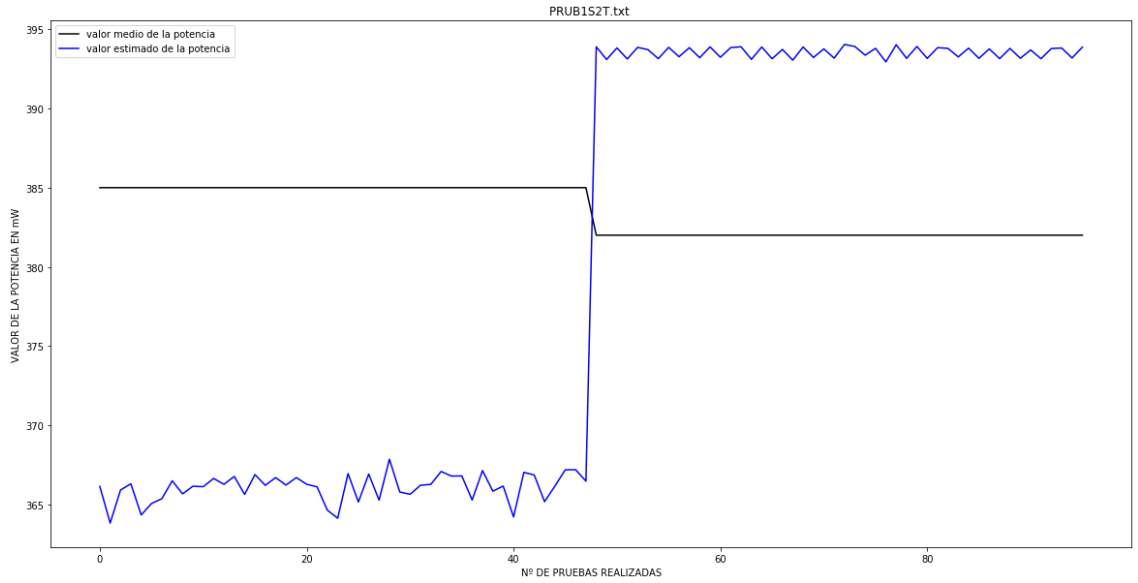


Ilustración 6-6 Evolución de la potencia media frente a la estimada para un cierto fichero de datos de test utilizado en este modelo

6.1.3 Otros datos

En este apartado, se mostrarán los resultados del modelo al estimar la potencia utilizando los datos de entrenamiento y validación de los algoritmos del apartado 5.1.1 y.5.1.2.

6.1.3.1 Precisión

Por lo general la precisión del modelo es muy similar a la mostrada con los datos de entrenamiento y validación, aunque el error es algo mayor, como puede observarse en la Tabla 6-4

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	3.56
Error relativo medio entre la potencia real y la estimada	2.96

Tabla 6-4 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.1

6.1.3.2 Sensibilidad

La sensibilidad para este set de datos es BUENA y le ocurren las mismas cosas que lo explicado en el apartado 6.1.1.2.

6.2 Modelo para estimar la potencia de 1 core, usando el algoritmo de 2 cores con interrupción

En este apartado se comentará cuán bien es capaz de estimar la potencia el modelo generado con los datos del algoritmo utilizado.

La sensibilidad del modelo es buena. Respecto a la precisión, el modelo es bastante preciso, pero es menos preciso que el modelo del apartado 6.1. Los coeficientes del modelo pueden observarse en la Tabla 6-5.

Coeficiente a_i de la Ecuación 4-1	Valor de los coeficientes
a_1	-5.20140279e-02
a_2	-1.79328056e-02]
a_3	4.32407614e-01
a_4	-4.27704858e-01
a_5	-4.73762262e-01
a_6	5.59652496e-01
a_7	4.90422037e+00
a_8	-8.66166272e-02
a_9	3.28693154e-06

Tabla 6-5 Coeficientes para el modelo que estima la potencia de 1 cores usando los datos

La sensibilidad del modelo es la misma y tiene los mismos problemas, que la de los set de datos utilizados en el apartado 6.1. Con respecto a la precisión este modelo es menos preciso que el anterior, debido al que error medio (con la potencia real y con la potencia media) sube ligeramente con los datos de entrenamiento más validación y test y datos extra. Los errores para cada set de datos, pueden verse en la Tabla 6-6,Tabla 6-7 y la Tabla 6-8.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	3.86
Error relativo medio entre la potencia real y la estimada	2.86

Tabla 6-6 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.2

Nombre del error relativo	Valor del error (%)

Error relativo medio entre la potencia media y la estimada	4.86
Error relativo medio entre la potencia real y la estimada	5.80

Tabla 6-7 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.2

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	4.21
Error relativo medio entre la potencia real y la estimada	3.24

Tabla 6-8 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.2

6.3 Modelo para estimar la potencia de 2 cores, usando el algoritmo de 2 cores con interrupciones

En este subapartado, se mostrarán los resultados obtenidos al usar el modelo que estima la potencia de 2 cores. La precisión y la sensibilidad son ligeramente peores que la de los modelos anteriores. El α utilizado en la Ecuación 4-6 es de 1,25 y como coeficientes se han usado los coeficientes de la Tabla 6-1 .Debido a que el modelo del apartado 6.1, es el que mejor precisión tiene al estimar la potencia de 1 core.

6.3.1 Datos de entrenamiento en el que ambos cores ejecutan el mismo benchmark

6.3.1.1 Precisión

Respecto a la precisión en este modelo el error medio total es mayor que con los modelos que estiman la potencia de 1 core, pero aun así el error sigue siendo bajo (en torno al 5%). En la Tabla 6-9 pueden verse dichos errores.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	3.69
Error relativo medio entre la potencia real y la estimada	4.34

Tabla 6-9 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.3

Respecto al error medio total en función del tiempo de muestreo. Para este modelo, los errores siguen siendo bajos y ya no hay un pico de error cuando se muestrea a 333ms.

6.3.1.2 Sensibilidad

Respecto a la sensibilidad esta es BUENA y presenta las mismas ventajas y problemas explicados en los anteriores modelos.

6.3.2 Datos de entrenamiento en el que se ejecuta la secuencia de la Tabla 5-1

6.3.2.1 Precisión

Respecto al error medio total en este caso es superior al error total medio usado para los datos extra y de test de los otros dos modelos, pero inferior respecto a los datos de entrenamiento más validación de otros modelos. Además, ocurre que en este caso el error medio total entre la potencia media y la estimada, es menor que el error medio total entre la potencia real y la estimada. Cuando en el resto de casos es al revés. Estos resultados pueden apreciarse en la Tabla 6-10.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	2.70
Error relativo medio entre la potencia real y la estimada	3.35

Tabla 6-10 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.3

6.3.2.2 Sensibilidad

Por lo general, el modelo es capaz de detectar un cambio grande en el consumo de potencia al cambiar de benchmark, pero no en todos los casos. Como se puede ver en la Ilustración 6-7, la potencia real media al cambiar de benchmark se mantiene constante, pero el modelo estima que la potencia sube. Provocando que el modelo no estime la potencia correcta. Además, se puede observar que cuando se ejecutan ciertas combinaciones de benchmarks la potencia estimada no es tan constate y esta tiene un cierto grado de oscilación. Aun con todo esto, el número de fallos a la hora de estimar un cambio en el consumo de potencia es bajo y se considera que la sensibilidad es **BUENA**.

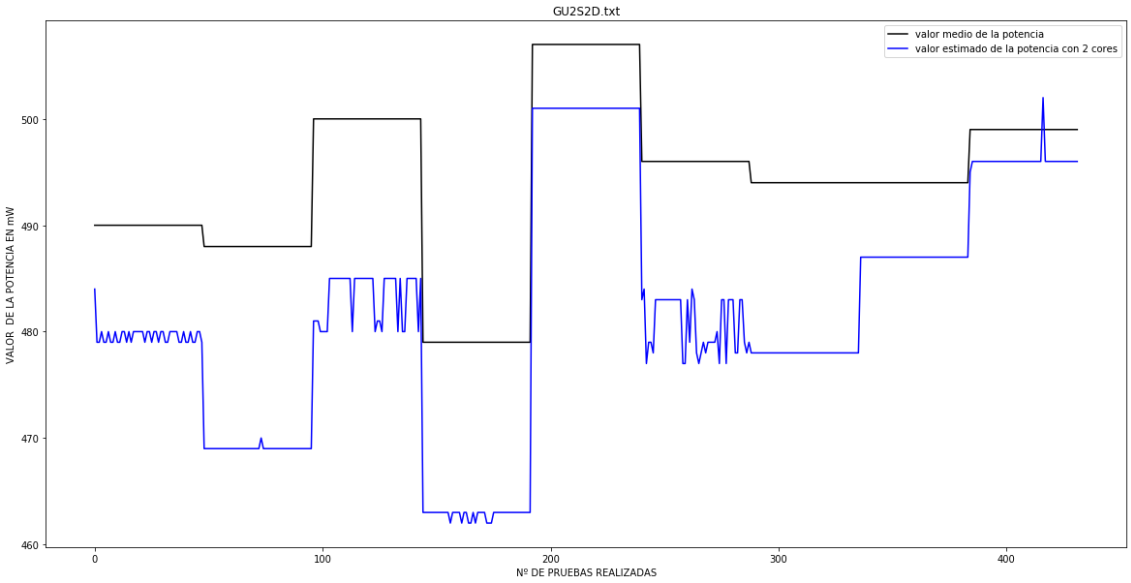


Ilustración 6-7 Evolución de la potencia media frente a la estimada para un cierto fichero de datos utilizado en este modelo

6.3.3 Datos de entrenamiento en el que se ejecuta la secuencia de la Tabla 5-2

6.3.3.1 Precisión

Con estos datos la precisión del modelo es muy parecida a la mostrada en el apartado 6.3.2. En la Tabla 6-11. Para más resultado consulte el script anexo al proyecto.

Nombre del error relativo	Valor del error (%)
Error relativo medio entre la potencia media y la estimada	3.31
Error relativo medio entre la potencia real y la estimada	3.8

Tabla 6-11 Error relativo entre la potencia media y la estimada y el error entre la potencia real y la estimada para los datos de validación y entrenamiento del modelo del apartado 6.3

6.3.3.2 Sensibilidad

Respecto a la sensibilidad, ocurre el mismo problema que el descrito en el apartado 6.3.2.2. Aun así, el número de fallos en el modelo son pequeños y la sensibilidad se considera **BUENA**.

6.4 Resultados finales

Tras explicar los resultados de cada modelo, se puede comprobar que para estimar la potencia de 1 core el modelo del apartado 6.1, es el más preciso de los dos y ambos tienen una sensibilidad similar. Así que dicho modelo es el mejor para estimar la potencia de 1 core. Esto se puede ver en la Ilustración 6-8,Ilustración 6-9,Ilustración 6-10,Ilustración 6-11,Ilustración 6-12 y la Ilustración 6-13.

Por otro lado, se puede ver que excepto para el set de datos de test, el modelo que estima la potencia de 2 cores es menos preciso que los modelos que estiman la potencia de 1 core. Para finalizar, se puede observar que ningún error medio total supera el 6%. Por lo tanto, la precisión del modelo es alta, ya que los errores obtenidos son similares a los obtenidos en [5, 6] y [3, 4].

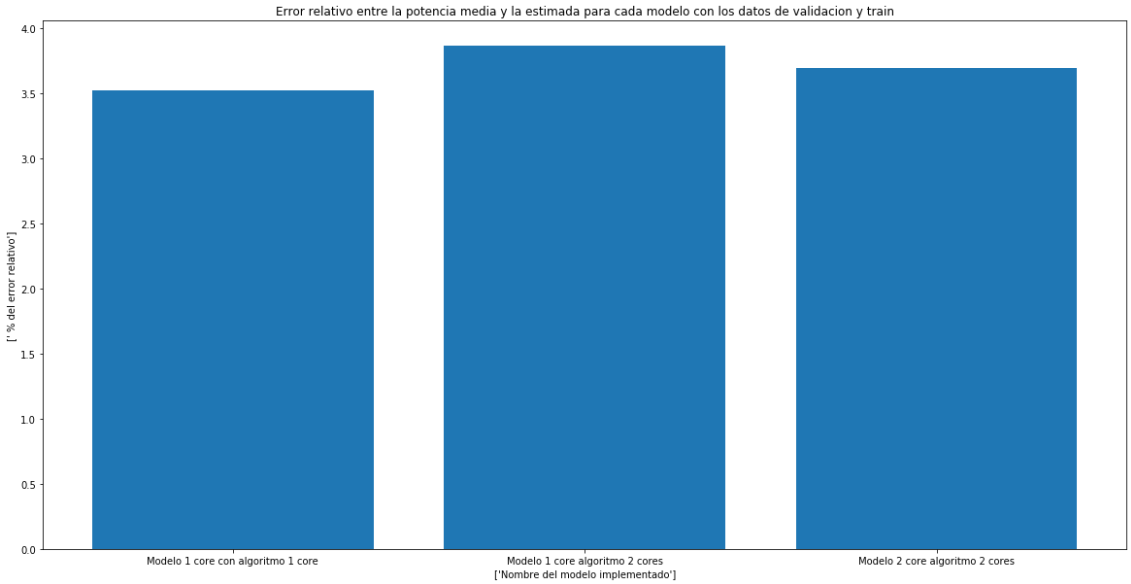


Ilustración 6-8 Error relativo entre la potencia media y la estimada para cada modelo con los datos de validación y train

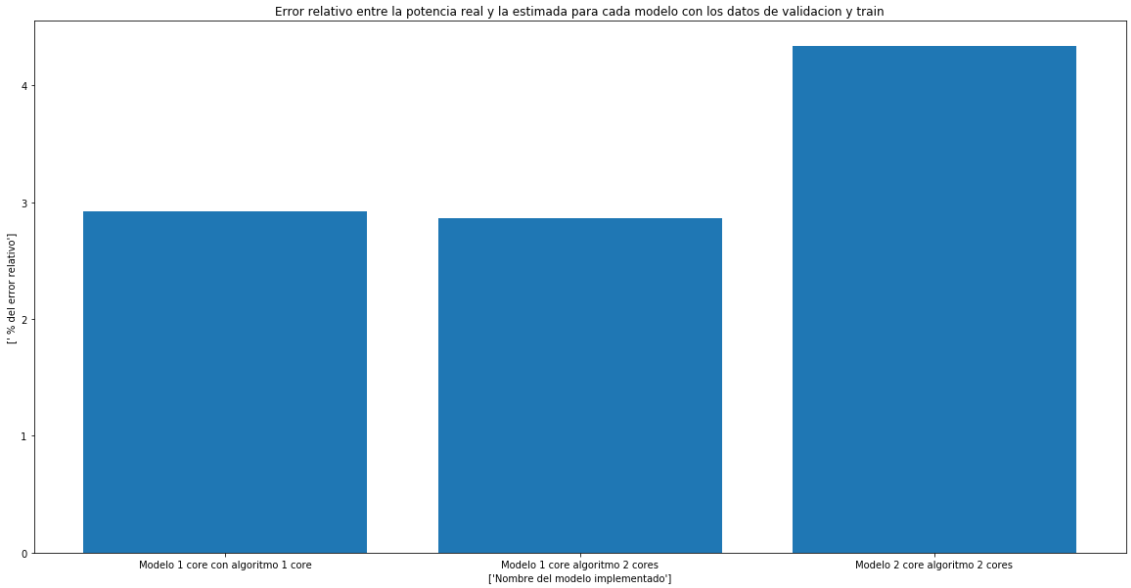


Ilustración 6-9 Error relativo entre la potencia real y la estimada para cada modelo con los datos de validación y train

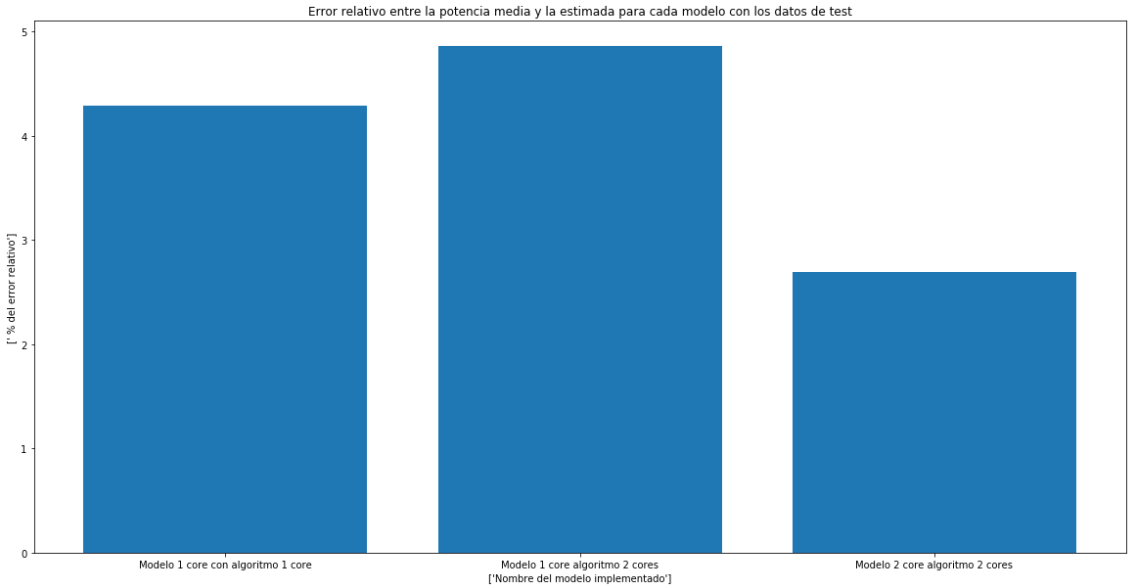


Ilustración 6-10 Error relativo entre la potencia media y la estimada para cada modelo con los datos de test

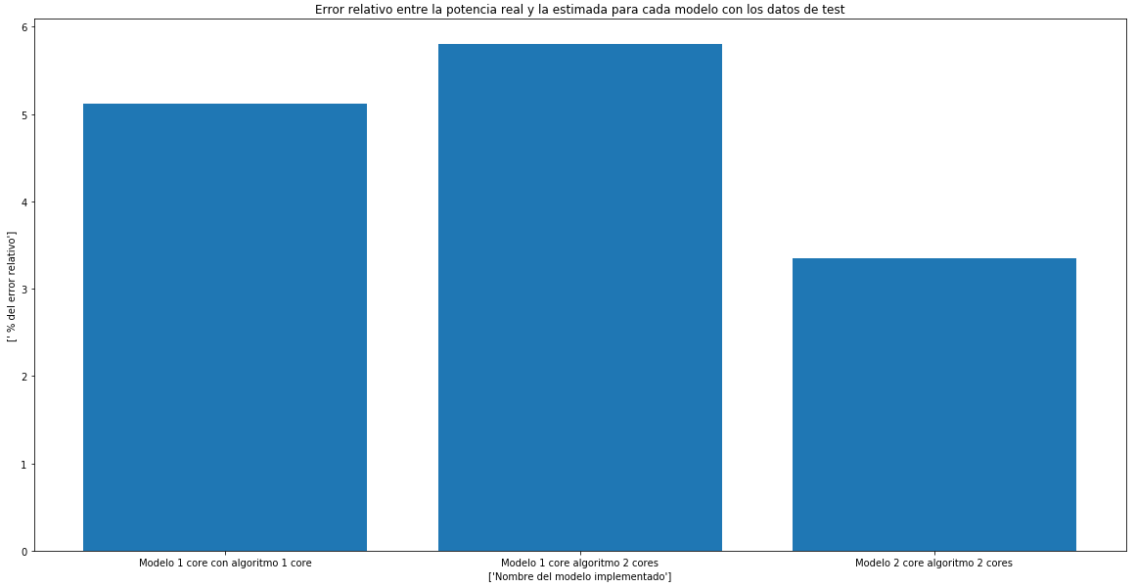


Ilustración 6-11 Error relativo entre la potencia real y la estimada para cada modelo con los datos de test

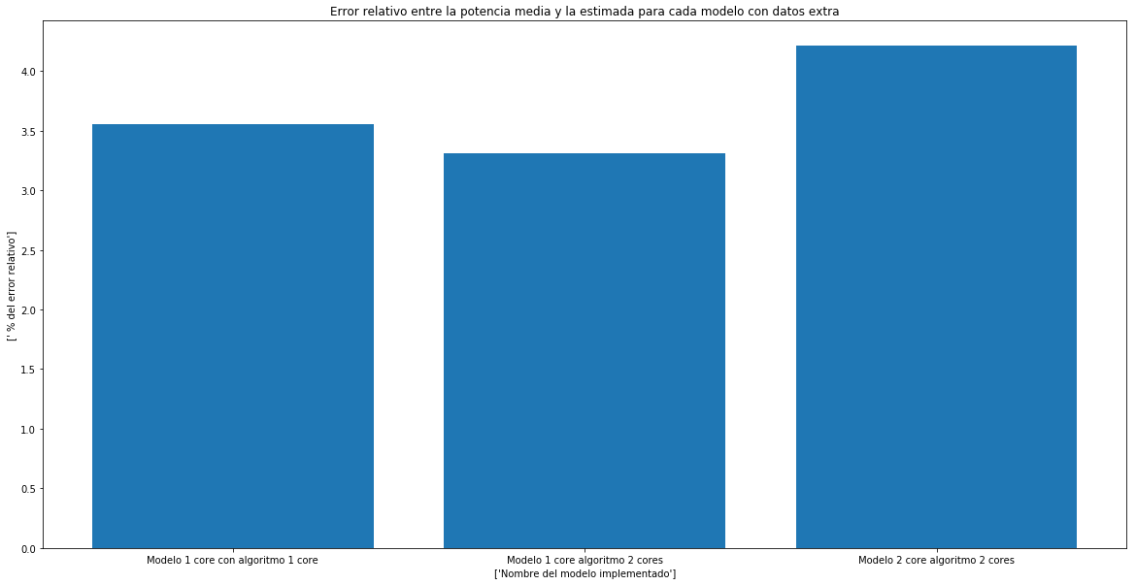


Ilustración 6-12 Error relativo entre la potencia media y la estimada para cada modelo con datos extra de entrenamiento

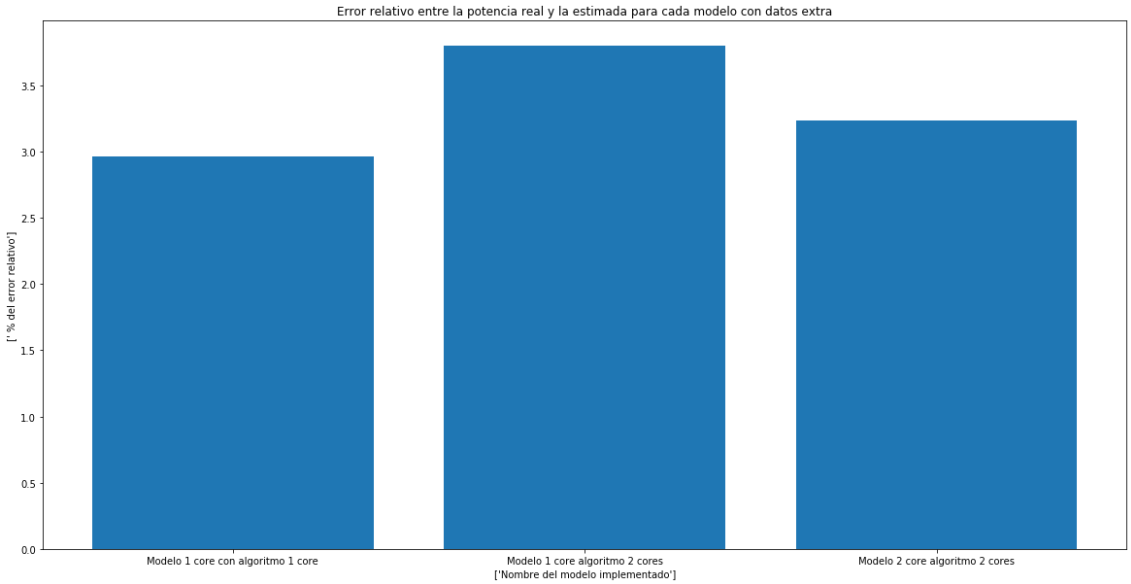


Ilustración 6-13 Error relativo entre la potencia real y la estimada para cada modelo con datos extra de entrenamiento

Respecto a la sensibilidad, los modelos de 1 core perciben mejor cuando el consumo de potencia varía que el modelo de 2 cores. Por otro lado, en todos los modelos filtran bastante bien el ruido producido por la medida de la potencia real haciendo que la potencia medida por benchmark sea casi constante o con muy pocas oscilaciones. Finalmente, a menor tiempo de muestreo más componente oscilatorio tendrá la potencia estimada y el error relativo sube ligeramente.

7 CONCLUSIONES

Respecto al trabajo realizado se pueden sacar las siguientes conclusiones. Se ha llegado a construir una serie de modelos que nos permiten estimar la potencia consumida de uno o más cores de la CPU, cuya precisión es bastante buena y saben detectar los cambios en el consumo de potencia al cambiar de benchmark cuando la diferencia de consumo es significativa. Adicionalmente, dicho modelo estima la potencia consumida por la CPU con menos ruido respecto a los sensores físicos montados en la placa ZC702.

Además, este modelo de potencia puede ser implementado en dispositivos que usen la misma CPU para estimar rápidamente la potencia consumida. Esto puede ser muy útil en los casos en que sea muy difícil integrar sensores para medir la potencia consumida. Asimismo, se ahorraría en costes al prescindir del hardware necesario para monitorizar la potencia consumida.

Respecto a los aspectos a mejorar de este proyecto, hay dos grandes frentes:

- Por un lado, se puede mejorar la parte relativa a la construcción y análisis del modelo de potencia. Para ello, existen mecanismos que permitan analizar la sensibilidad del modelo de manera más analítica y automatizada, como los algoritmos de detección de fase. Además, se puede automatizar la selección de datos de entrenamiento validación y test. De esta forma la generación y el análisis de la sensibilidad de nuevos modelos sería mucho más rápida y fácil.
- Por otro lado, sería interesante implementar este modelo en entornos con un sistema operativo, como Linux. Para ello se tendrían que diseñar nuevos algoritmos que permitan ejecutar una secuencia de benchmarks, mientras de manera periódica se recoge el valor de los PMC y de la potencia consumida.

8 PLANIFICACIÓN

8.1 Recursos humanos

Para los trabajos descritos en el presente documento se dispone de una persona en desarrollo y el tutor del trabajo persona. Las etapas del proyecto se describen a continuación.

8.2 Etapas del proyecto

La planificación para las tareas tiene la siguiente estructura:

1. Labores Previas: Definición del alcance y objetivo del proyecto.
2. Estudio del arte sobre los performance monitoring counters, la arquitectura ARM y sobre la PCB Zynq ZC702.
3. Aprender a manejar Python y Xilinx SDK
4. Búsqueda y/o diseño e implementación de benchmarks para testear la CPU con la que se trabaja.
5. Selección de los PMC a usar en el modelo a implementar.
6. Diseño matemático de un conjunto de modelos que nos permitan estimar la potencia de uno o más cores de la CPU.
7. Diseñar un conjunto de algoritmos que realice las siguientes tareas: Ejecutar, de manera secuencial, los benchmarks seleccionados, Mientras tanto, de forma simultánea realizar, periódicamente, una adquisición de datos de la CPU (Performance monitoring counters, voltaje, temperatura, potencia consumida, tiempo de muestreo) y luego almacenar dichos datos en un fichero de texto.
8. Implementación de dichos algoritmos.
9. Con los ficheros de datos generados, indicar que datos van a ser usados para entrenar los diferentes modelos y cuales para test y validación.
10. Diseñar como se van a entrenar los modelos de potencia con los datos de training, para luego testearlo y validarlo con sus correspondientes set de datos.
11. Implementación de dichos modelos y sus mecanismos para validarlos y testearlos
12. Implementación de dichos modelos de potencia en la placa de trabajo y generación automática de informes sobre el consumo de potencia.
13. Documentación del proyecto.

8.3 Diagrama de Gantt

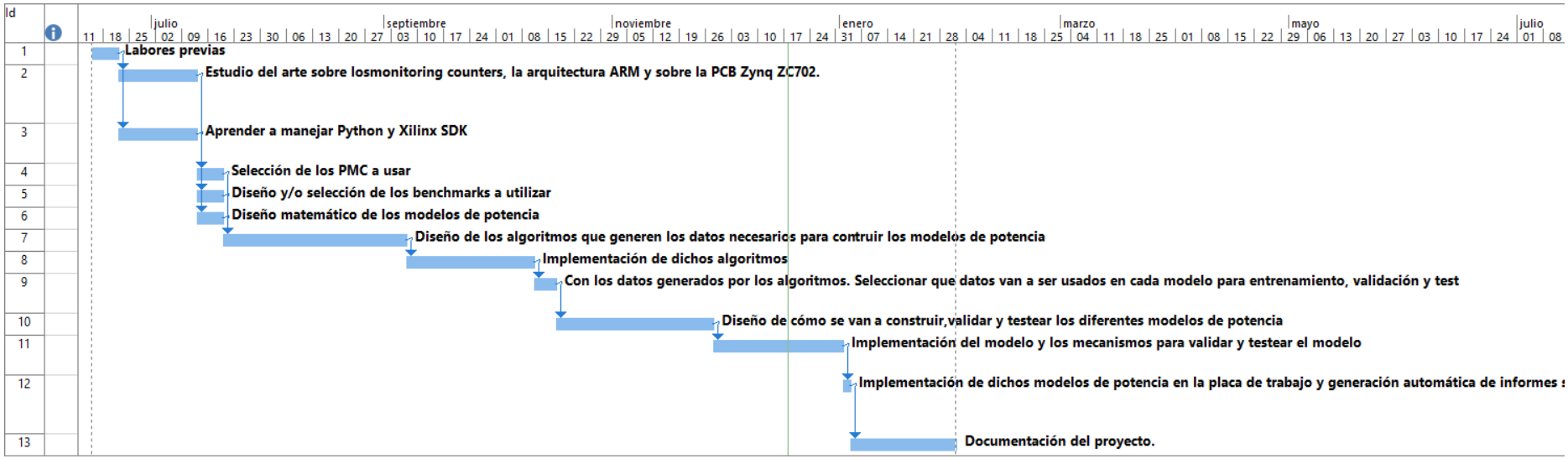


Ilustración 8-1 Diagrama de Gantt con la planificación del proyecto

9 PRESUPUESTO

En este apartado se detalla el presupuesto total del trabajo realizado que incluye materiales y mano de obra.

Respecto a los materiales hardware. Solo se utiliza la placa Zynq-7000 como la empresa la ha adquirido hace tiempo y ha sido utilizada en proyectos anteriores. Se considerará que dicho producto están amortizados y no tienen ningún gasto en el presupuesto.

Por otro lado, en los materiales software pasa lo mismo, licencias como el software Xilinx SDK para programar la placa o el pack de office para la redacción de la memoria. Son licencias que han sido compradas hace mucho tiempo y su coste ya ha sido amortizado. Por último, Python es un lenguaje de código abierto y gratuito y el coste total de esta unidad de obra es nulo.

Respecto a la mano de obra, se necesitará contar el sueldo de un proyectante que será quien realice la mayor parte de las tareas. Por otro lado, se necesitará un ingeniero jefe que se reúna semanalmente una hora con el proyectante para supervisar, ayudar y evaluar al proyectante.

9.1 Mano de obra

Los precios unitarios de mano de obra son los siguientes y el presupuesto de la mano de obra puede verse en la Tabla 9-1.

- 28.8 €/h Trabajo del proyectante.
- 56€/h trabajo del ingeniero jefe.

Nº	Concepto	Horas totales	Coste unitario (€)	Coste Total (€)
1	Hora de trabajo del proyectante	750,00	**5,25 €**	3.937,50 €
2	Hora de trabajo del ingeniero jefe	18,75	**56,00 €**	1.050,00 €
			Subtotal	4.987,50 €

Tabla 9-1 Presupuesto de la mano de obra

9.2 Presupuesto de ejecución material

Nº	Concepto	Coste Total (€)
1	Presupuesto de materiales software	- €
2	Presupuesto de materiales hardware	- €
3	Presupuesto de mano de obra	4.987,50 €
	Subtotal	4.987,50 €

Tabla 9-2 Presupuesto de ejecución material

9.3 Presupuesto de ejecución por contrata

Nº	Concepto	Coste Total(€)
1	Presupuesto Ejecución de Materiales	4.987,50 €
2	Gastos Generales (9%)	448,88 €
3	Beneficio Industrial (6%)	299,25 €
	Subtotal	5.735,63 €

Tabla 9-3 Presupuesto de ejecución pro contrata sin IVA

Nº	Concepto	Coste Total(€)
1	Total Antes de Impuestos	5.735,63 €
2	I.V.A. (21%)	1.204,48 €
	Subtotal	6.940,11 €

Tabla 9-4 Presupuesto de ejecución pro contrata con IVA

10 BIBLIOGRAFÍA

[1] S. Mittal, *A survey of techniques for improving energy efficiency in embedded computing systems*. 2013.

[2] A. Limited, "Cortex™-A9 Revision: r2p2 Technical Reference Manual ", ed, 2010.

[3] M. G. R.Bertran, X.Martorell,N.Navarro,E.Ayguade, "Decomposable and Responsive Power Models for Multicore Processors using Performance Counters," *ICS '10 Proceedings of the 24th ACM International Conference on Supercomputing,* p. 11, 2010.

[4] R.Bertran;M.Gonzalez;X.Martorell;N.Navarro;E.Ayguade, "Counter-Based power modeling methods: Top-Down vs. Bottom-up," *Oxford Academic- Computer Journal,* Paper vol. 56, no. 2, p. 15, 2013.

[5] M. J. Walker, S. Diestelhorst, A. Hansson, D. Balsamo, G. V. Merrett, and B. M. Al-Hashimi, "Thermally-aware composite run-time CPU power models," in *2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2016, pp. 17-24.

[6] M. J. Walker *et al.*, "Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 36, no. 1, pp. 106-119, 2017.

[7] B. K. Reddy, M. J. Walker, D. Balsamo, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, "Empirical CPU power modelling and estimation in the gem5 simulator," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1-8.

[8] L. P. Hewlett-Packard Development Company. (2009, 3/10/2018). *Information about Mcpat*. Available: http://www.hpl.hp.com/research/mcpat/

[9] Xilinx, "ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide," ed, 2015.

[10] Xilinx, "Zynq-7000 All Programmable SoC Data Sheet: Overview," ed, 2017.

[11] Xilinx, "Zynq-7000 All Programmable SoC Software Developers Guide," ed, 2017.

[12] Xilinx, "Zynq-7000 SoC Technical Reference Manual," ed: XILINX, 2018.

[13] Xilinx. (5/10/18). *Xilinx Software Development Kit (XSDK)*. Available: https://www.xilinx.com/products/design-tools/embedded-software/sdk.html

[14] J. Python. (2008-2018, 05/10/2018). *Jupyter Python*. Available: http://jupyter.org/

[15] T. S. community. (2018, 05/10/2018). *SciPy and numpy*. Available: https://docs.scipy.org/doc/numpy/index.html

[16] *NumPyBook*, 1 ed. 2006.

[17] D. D. John Hunter, Eric Firing, Michael Droettboom (2012-2018, 5/10/2018). *Matplotlib*. Available: https://matplotlib.org/tutorials/introductory/pyplot.html

[18] C. S. Larry McVoy. (1996, 09/10/2018). *LMbench*. Available: http://lmbench.sourceforge.net/

[19] J. R. Matthew Guthaus, Todd Austin, Trevor Mudge, Richard Brown. (2002, 09/10/2018). *MiBench Version 1.0*. Available: http://vhosts.eecs.umich.edu/mibench/

[20] (26/06/2018). *Benchmark Whetstone, GNU Development Chain for 68HC11/68HC12*. Available: https://www.gnu.org/software/m68hc11/examples/benchExamples.html

[21] Wikipedia. (28/06/2018). *Tak (function)* Available: https://en.wikipedia.org/wiki/Tak_(function)

[22] N. T.Ooura. (1999, 8/10/2018). *alculation of PI(= 3.14159...) using FFT and AGM* ----. Available: https://github.com/tonyho/ARM_BenchMark/blob/master/pi_css5_src/pi_fftcs.c

[23] (28/06/2018). *Benchmark Programs and Reports*. Available: http://www.netlib.org/benchmark/

[24] (28/06/2018). *FatFs - Generic FAT Filesystem Module*. Available: http://elm-chan.org/fsw/ff/00index_e.html