Sistema de tiempo real con Raspberry Pi.

A. Blesa*, E. Alcalá*, A. Azuara†, C. Catalán†, F. Serna†
*Dpt. de Ingeniería Electrónica y Comunicaciones
†Dpt. de Lenguajes y Sistemas Informáticos
Universidad de Zaragoza, E. U. Politécnica de Teruel, Email: ablesa@unizar.es

***Abstract*—Las plataformas basadas en *Systems on Chip* (SoC) son capaces de ejecutar Linux a costes muy bajos, resultando una opción a tener en cuenta para utilizarlas en sistemas empotrados. Este trabajo presenta una plataforma Raspberry Pi, con el *kernel* de Linux modificado para cumplir restricciones de tiempo real. Se mide el tiempo de latencia bajo diversas condiciones, y se acotan los tiempos de ciclo de ejecución requeridos para atender tareas de tiempo real. Esta plataforma se ha utilizado con éxito en docencia de sistemas empotrados. Como caso de uso se presenta el control de estabilidad de un eje de un cuadricóptero.**

I. Introducción

La evolución de los sistemas empotrados es un factor que ha impulsado la era "Internet of Things". Los productos que consumimos tienen cada vez más funcionalidades, son más complejos y deben estar conectados a otros dispositivos, tanto locales como remotos.

Para dar respuesta a estas necesidades, los fabricantes ofrecen diversas soluciones tecnológicas: Dispositivos programables (del tipo PLD o FPGA), microcontroladores de diferentes características (en cuanto a velocidad de reloj, tamaño del bus, memoria, dispositivos de E/S, buses de comunicaciones, etc.). En la actualidad se dispone de un abanico que contempla sencillos microcontroladores de 4 bits hasta dispositivos con arquitecturas de 32 o 64 bits capaces de soportar sistemas operativos complejos.

Un paso más en esta dirección son los *System on Chip* (SoC) que integran en un único circuito integrado todo un computador. Raspberry Pi [1] es un ejemplo de este tipo de dispositivos. Esta plataforma permite ejecutar un sistema operativo Linux completo, incluyendo posibilidades de entorno gráfico, conectividad Ethernet, atención a dipositivos USB y con disponibilidad de acceso en la tarjeta a GPIO. Esta última característica, junto con su bajo coste de adquisición y una gran disponibilidad de software libre, hace adecuada a Raspberry Pi para aplicaciones de automatización y control sencillas, susceptibles de ser utilizadas en ámbitos docentes, tanto universitarios como no universitarios.

En este trabajo se presenta un procedimiento para modificación del *kernel* de Linux, que viene en la distribución Raspbian estándar [2], para Raspberry Pi con el objeto de dotar características de tiempo real en esta plataforma. En la sección II se hace referencia a las diferentes soluciones propuestas hasta la actualidad, en la sección III se describe el proceso de modificación del *kernel*. En la sección IV se presentan resultados de latencia para una aplicación sencilla en diferentes situaciones de carga. Seguidamente, en la sección V, se explica su uso para el control de estabilidad en vuelo de un cuadricóptero, en una asignatura de sistemas empotrados, incidiendo en las posibilidades docentes de este tipo de sistemas. Por último, se presentan las conclusiones resumiendo aquellas más relevantes.

II. Tiempo real en Linux

A. Fundamentos

Un sistema operativo para tiempo real es aquel capaz de garantizar los requisitos temporales de los procesos que controla. En función de la exigencia temporal podemos distinguir: a) *Tiempo real estricto (Hard Real Time)* en aplicaciones críticas desde el punto de vista de seguridad de las personas: Todas las acciones deben ocurrir dentro del plazo especificado, una sola respuesta tardía de una acción puede tener consecuencias fatales, también se llaman sistemas de tiempo real críticos (ej. sistemas de control de centrales nucleares o de aviones). b) *Tiempo real estricto* para aplicaciones que no son críticas para la seguridad de las personas: Los sistemas que permiten perder sólo algunos plazos de ejecución (típicamente del 90 al 95 %). Se deben cumplir los *deadlines*, pero no hay seguridad de personas en juego (ej. sistemas de control de lavadoras). c) *Tiempo real flexible o blando (Soft Real Time)*: Se pueden perder plazos de vez en cuando, también se conocen como sistemas de tiempo real acrítico, o no crítico (ej. *streaming* de video, videojuegos). d) *Firm real time*: El resultado de la computación se considera obsoleto si no ha llegado en el tiempo establecido (ej. previsión del tiempo atmosférico).

El criterio fundamental de evaluación para conocer el rendimiento de un sistema operativo de tiempo real es la latencia y el *jitter* ante eventos (ej. interrupciones). La latencia es el tiempo desde que se produce el evento hasta que se ejecuta la primera instrucción de la rutina de servicio asociada al mismo. El *jitter* se refiere a las variaciones en el periodo que experimenta una tarea cuando se ejecuta de manera repetitiva.

El objetivo de un sistema operativo de tiempo real es reducir la latencia y el *jitter*. Linux (por su propia naturaleza de sistema operativo multiusuario y multitarea) no permite "a priori" disponer de un acceso directo desde una aplicación a los recursos hardware. Además, no garantiza características de tiempo real estricto (*hard realtime*). Existen diversas estrategias para proveer al *kernel* de Linux de capacidades de tiempo real, con el objeto de poder utilizarlo como sistema operativo en aplicaciones de automatización y control:

a) Atención prioritaria a determinados *threads* en el *kernel* estándar (*patch* `CONFIG_PREEMPT_RT`) [3]: Esta estrategia

modifica el *kernel* de forma que determinados procesos de *kernel* se ejecuten con máxima prioridad interrumpiendo a procesos de menor prioridad. Se basa en el uso de *kernel threads*.

b) Micro-*kernel*: Implica añadir un segundo *kernel* (micro-*kernel*) para manejar las tareas asociadas al tiempo real, teniendo en cuenta interrupciones, planificación y contadores (*timers*) de alta resolución. El micro-*kernel* se puede entender como una nueva capa entre el hardware y Linux, que es atendido como una tarea de segundo plano (*background*). De los proyectos activos podemos destacar los *patch* Xenomai [4] y RTAI [5].

B. Plataforma Raspberry Pi

Raspberri Pi [1] es un claro ejemplo de uso de SoC: Pertenece una generación de ordenadores denominados SBC (*Single Board Computer*), que se caracterizan por integrar en una sola placa todos los componentes de un ordenador convencional. En concreto esta placa utiliza el chip SoC Broadcom 2835, que consta de un procesador (ARM1176JZF-S a 700 MHZ), una tarjeta gráfica o GPU (Broadcom VideoCore IV) y memoria SDRAM (512Mb).

Raspberry Pi permite ejecutar varias distribuciones Linux. Raspbian [2] se considera su distribución oficial y, en el momento de la presentación de este trabajo utiliza el *kernel* 3.12.32 y el entorno gráfico LXDE.

Existe una activa comunidad de desarrolladores de aplicaciones para esta plataforma. En concreto, es posible encontrar ejemplos y aplicaciones para automatización codificados en diversos lenguajes (C, C++, Python, etc.) y librerías que facilitan el acceso a hardware [6]. Con todos estos recursos es muy sencillo visualizar (Fig. 1) en un laboratorio de electrónica básico cómo una señal periódica se ve fuertemente afectada cuando la CPU debe atender a otras tareas (refresco de pantalla en entorno gráfico, transferencia de ficheros vía ftp, etc.).

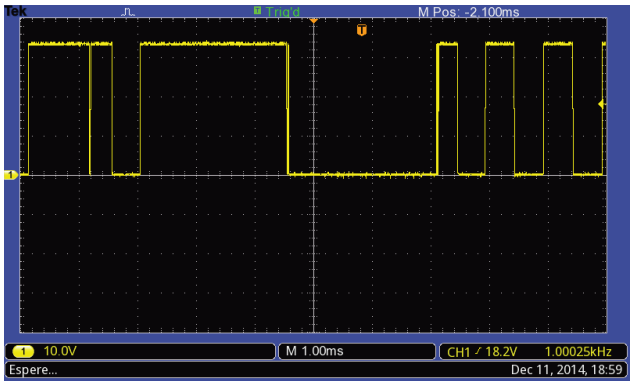

Fig. 1. Generación de eventos periódicos en la Raspberry bajo condiciones de no tiempo real. El tiempo de ciclo (0.5 ms.) se ha elegido para evidenciar de forma gráfica los problemas de tiempo que aparecen con sistemas operativos de propósito general.

El código que genera esta función utiliza los servicios de sistema para activar periódicamente (y con un tiempo de ciclo determinado) el *thread* asociado a esta tarea. En concreto, la función `clock_nanosleep` bloquea el *thread*, una vez que se ha ejecutado el acceso a GPIO, hasta que el reloj del sistema alcanza el valor `next`, dejando la CPU a otros *threads*. El planificador lo atenderá cuando pase el tiempo indicado. De esta forma se puede implementar un *thread* periódico. El ejemplo mostrado en la Fig, 1 utiliza la línea 4 del puerto GPIO de la Raspberry (que ha sido configurado como salida) y el bucle que genera la señal periódica es el siguiente:

```
//Referencia inicial de tiempo
clock_gettime(CLOCK_REALTIME, &next);

//Bucle del thread
while(1) {
  // genera onda cuadrada
  if (i++ % 2) {
    GPIO_SET = 1<<4;
  }
  else {
    GPIO_CLR = 1<<4;
  }
  // espera cumplimiento tiempo de ciclo
  timespec_add_us(&next, ps->period_us);
  clock_nanosleep(CLOCK_REALTIME,
   TIMER_ABSTIME, &next, NULL);
}
```

III. Modificación del *kernel*

El procedimiento de modificación del *kernel* es muy similar para todas las opciones descritas en la sección II, y está suficientemente descrito en los repositorios correspondientes [3, 4, 5]. En este trabajo se ha optado por la opción `CONFIG_PREEMPT_RT` [3]. También se han han probado distribuciones basadas en Xenomai con resultados equivalentes en cuanto a tiempos de latencia.

Para modificar el *kernel* se deben seguir los siguientes pasos: *a)* Descargar los ficheros "fuente" del *kernel* que se facilita en la distribución Raspbian, así como los del parche de tiempo real elegido. *b)* ejecutar el comando `patch` para modificar los ficheros fuente del *kernel* con los *patches* del repositorio rt [3], eligiendo la versión adecuada al *kernel*. *c)* configurar el *kernel*, seleccionando el modo de prioridad `full preemptible kernel` (una vez ejecutado el comando `make menuconfig`). *d)* compilar el *kernel* y los módulos asociados al mismo. *e)* instalar el *kernel* y los módulos en la tarjeta SD de almacenamiento permanente para la Raspberri arrancar con la nueva configuración. *f)* verificar su funcionamiento con herramientas de test.

Para Raspberry Pi es aconsejable hacer el proceso de patch y compilado en un ordenador con mayor potencia de cálculo, utilizando compiladores cruzados. La versión de los paquetes de tiempo real deben ser concordantes con la versión de *kernel* que se quiere utilizar.

Todo este proceso, basado en comandos del sistema operativo, se guarda en un fichero script (`compile.sh`), que automatiza el proceso. Uno de los problemas que se han encontrado para el adecuado mantenimiento y explotación de este *kernel* modificado, es la constante evolución de las

distribuciones Raspbian, y con ello las de sus versiones de *kernel* (y en ocasiones, la evolución del hardware). Otro fichero de comandos script, denominado `update.sh`, resuelve este problema.

Tanto los ficheros script `compile.sh` y `update.sh`, junto con una guía de instalación detallada, como los ficheros fuente del *kernel* y parches utilizados, así como una imagen de la distribución Raspbian con el *kernel* modificado se pueden encontrar en el repositorio [7].

IV. Medidas de latencia

El acceso a un GPIO, generando una señal periódica, es la primera aproximación que permite comprobar con claridad la diferencia de comportamiento entre el *kernel* de la distribución original y el *kernel* modificado como *preemptive-RT*. La medida de latencia se realiza a partir de tests descritos en [8, 9]. Se distingue entre aplicaciones en espacio de usuario y espacio de *kernel*. Las primeras son más fáciles de implementar y mantener, mientras que las segundas tienen menores tiempos de latencia. La elección entre unas y otras depende de la aplicación, del tiempo de ciclo característico de la misma y de las necesidades de tiempo real estricto que se exigen [10].

Para tiempos de ciclo de 10 ms. y en condiciones de bajo uso de CPU y memoria se observan variaciones del orden de hasta 4 ms para *kernel* Raspbian original. Este escenario cambia completamente al utilizar el *kernel* modificado según la sección III.

`cyclictest` [9] es una batería de test que permite medir los tiempos de latencia frente a respuesta a estímulos para sistemas basados en Linux, y es el que se ha utilizado en este trabajo. Su algoritmo es muy simple y utiliza las librerías `/sys/class/gpio` para acceder a las GPIO [10]:

```
clock_gettime((&now))
next = now + par->interval
while (!shutdown) {
 clock_nanosleep((&next))
 clock_gettime((&now))
 diff = calcdiff(now, next)
 next += interval
```

Este código permite comprobar la latencia bajo diferentes condiciones. En la tabla I se muestran valores obtenidos de latencia bajo las condiciones de operación reales. Se dan seis escenarios: Eventos cíclicos de 0.5, 1 y 10 ms, ejecutados por *threads* sin condiciones de tiempo real (NO-RT) y tiempo real (RT), y se mide la desviación máxima de tiempo de ejecución con respecto del periodo (Error) y, en el caso de tiempo real, el porcentaje de eventos que no cumplen una desviación menor del 10% del tiempo de ciclo. Este umbral es arbitrario y depende de la aplicación a considerar.

Según los datos datos obtenidos, para aplicaciones que exigen restricciones de tiempo real estricto, no se aconsejan tiempos de ciclo por debajo de los 10 ms. para esta plataforma.

En la Fig. 2 se observa que, para el mismo código que para la generada en la Fig. 1 pero ejecutándose bajo un *thread* de tiempo real. El comportamiento cumple claramente con los requisitos, a pesar de que el tiempo de ciclo es muy inferior al recomendado por el análisis estadístico de la tabla I.

Thread	**Tiempo de ciclo: T (μs.)**	**|Error|** (μs).	**Desb.** $\geq 10\%$
NO-RT	500	507	-
NO-RT	1000	1818	-
NO-RT	10000	761	-
RT	500	77	15%
RT	1000	84	8%
RT	10000	81	$\leq 1\%$

TABLA I
MEDIDAS DE LATENCIA SIN CARGA. EVENTO PERIÓDICO QUE GENERA LA SALIDA DE UNA ONDA CUADRADA POR EL GPIO 4. EN EL CASO DE *threads* DE TIEMPO REAL, "DESB." INDICA EN PORCENTAJE DE EVENTOS QUE NO CUMPLEN LA COTA DEL 10% DE ERROR.

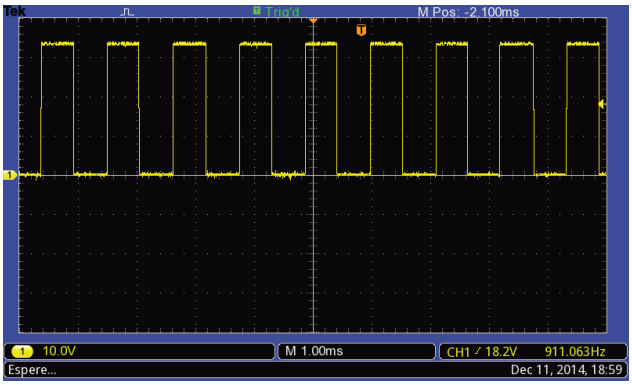

Fig. 2. Generación de una onda cuadrada de 0.5 msg con *thread* de tiempo real. Se observa con claridad la mejora en el comportamiento de la misma

V. Aplicación a la docencia de sistemas empotrados

Los sistemas automáticos son cada vez más complejos, y exigen introducir en el curriculum de los grados relacionados con ingenierías de automatización y electrónica sistemas operativos de 32 bits incluyendo conocimientos de control en tiempo real [11]. Desde nuestra experiencia, los contenidos mínimos que se deben atender son los siguientes: a) hardware: fundamentos de sistemas empotrados, GPIOs y sus técnicas de acceso, comunicaciones (Serie, SPI, IIC, etc.), bloques DAC-ADC y fundamentos de redes de comunicaciones. b) software: fundamentos de sistemas operativos, programación de GPIO (mediante APIs o mediante librerías específicas) y configuración y uso de un S.O. de tiempo real.

A. Control de estabilidad de un cuadricóptero

Para fijar los conocimientos implicados en esta asignatura, se propone un sencillo problema de control de estabilidad en un eje de vuelo de un cuadricóptero.

Los cuadricópteros incorporan una hélice en cada uno de los cuatro motores, los cuales normalmente están situados simétricamente en los extremos de los brazos de la estructura (la cual tiene forma de cruz). Utilizan un sistema de control electrónico y sensores electrónicos, situados sobre el eje del

centro de gravedad, para estabilizar la aeronave. El control de movimiento del vehículo se consigue mediante la alteración de la velocidad de rotación de uno o más motores, cambiando de ese modo su carga de par y las características de empuje y elevación.

La dinámica de los cuadricópteros cuenta con 3 tipos distintos de movimientos básicos, los cuales se pueden ejecutar en conjunto para lograr el tipo de movimiento deseado. Estos tipos de movimientos son los siguientes: *a)* Rotación sobre los ejes X e Y (que definen el plano horizontal), estos ángulos se denominan *roll* y *pitch*. *b)* rotación sobre el eje Z (eje de elevación) y *c)* elevación. En el presente trabajo nos centramos únicamente en el primer tipo.

En el ámbito de una asignatura de sistemas electrónicos empotrados, impartida en el grado de ingeniería de electrónica y automática, un cuadricóptero se puede entender como un problema de control que exige tiempos de respuesta en el rango de unos pocos ms. (también hay que solucionar problemas solucionados con instrumentación, sensado, dinámica de vuelo, etc). Se propone el control de estabilidad de uno de los ángulos de rotación sobre los ejes X e Y, utilizando como plataforma de control la Raspberry Pi con el *kernel* modificado y con características de tiempo real.

1) Hardware: En la Fig. 3 se observa el esquema simplificado del sistema: los sensores, un acelerómetro ADXL345, un magnetómetro HMC5883 y un giróscopo ITG3200 se comunican vía I2C con la plataforma Raspberry Pi, que hace las veces de control evalúando los datos recibidos y envíando las consignas adecuadas a un microcontrolador PCA9685, que genera una señal PWM a cada ESC (*Electronic Speed Controller*) y que, a su vez, controla la potencia suministrada a cada motor. Se puede acceder de forma remota a este sistema, vía wifi, aprovechando los servicios de red que ofrece Linux.

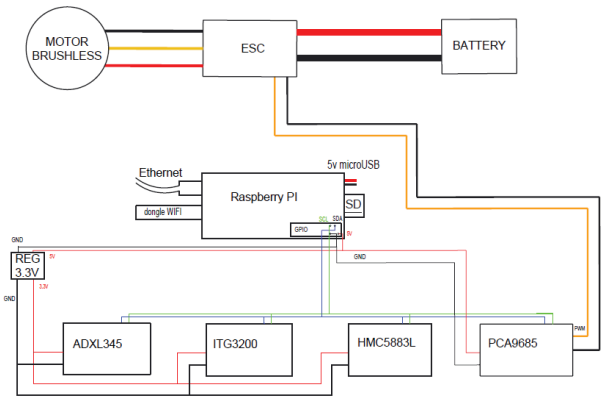

Fig. 3. Esquema de los bloques de entrada y salida del cuadricóptero. Por simplicidad, se presenta sólo un conjunto motor-variador

En la Fig. 5 se observa una instalación de pruebas para ajustar los coeficientes de uno de los ángulos (*pitch* o *roll*). La aplicación y la plataforma COSME permite monitorizar, en tiempo real, variables relevantes del test (consumo, ángulos, etc.)

2) Software: En esencia, el software de control se basa en un clásico lazo PID con tiempo de ciclo de 10 ms. Además será necesario modelar el acceso a los dispositivos de E/S, corregir los errores que puedan introducir los sensores (derivas) y calibrar la señal que se envía a los motores (tener en cuenta el comportamiento dinámico del par motor-hélice).

Este software se ha realizado utilizando en bloques función (FB, de sus siglas en inglés *Function Block*), que son bien conocidos por los ingenieros de control a través del estándar IEC 61131-3 y, más recientemente, por el estándar IEC 61499. La aplicación consiste en una red de FB conectados entre sí. Se dispone de una librería de tipos FB que modelan cada parte de la aplicación (sensores, reguladores PID, actuadores, etc.) y se instancian en la red de FB tantas veces como sea necesario. En la Fig. 4 se observa la red de FB propuesta.

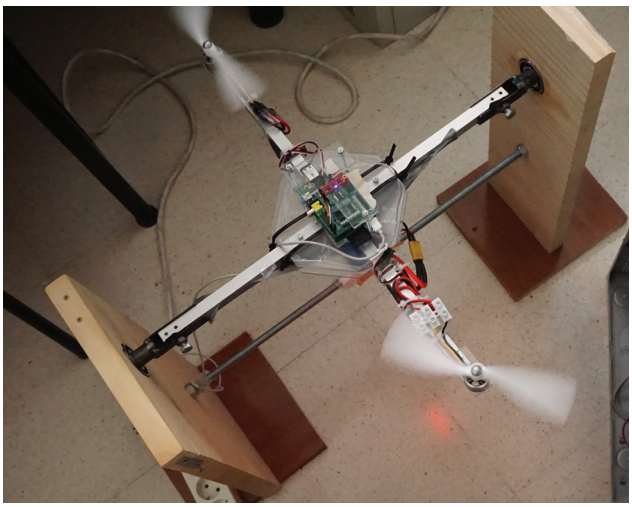

Fig. 5. Prototipo construido montado en un banco de pruebas, que permite ajustar los parámetros de los reguladores para cada eje. En el centro se observan los módulos de sensado y la plataforma Raspberry Pi.

COSME [12] es la plataforma que ejecuta esta red de FB y se ha desarrollado a partir de las especificaciones impuestas por el estándar IEC 61499. En la red (Fig. 4, se distingue con claridad los componentes asociados a sensado, control y actuación del sistema. Además, los FB y la plataforma COSME ocultan al diseñador la complejidad asociada la gestión de recursos que debe hacer el sistema operativo (acceso a hardware, creación de *threads*, gestión de memoria, etc.).

Cada FB se caracteriza por sus entradas y salidas (conectadas a otros componentes) y por variables y funciones que se deben definir para cada instancia del mismo. El diseñador sólo debe atender a la funcionalidad de cada bloque, a la definición y configuración de sus entradas, salidas y variables.

Las plantillas para desarrollar una aplicación COSME se encuentran en [7]. Son un conjunto de ficheros en los que el alumno solo tiene que incluir la funcionalidad de cada bloque en el fichero correspondiente a su tipo de componente y luego definir la red de conexión entre ellas.

3) Resultados: COSME dispone de herramientas de monitorización que permiten conocer el valor de variables rele-

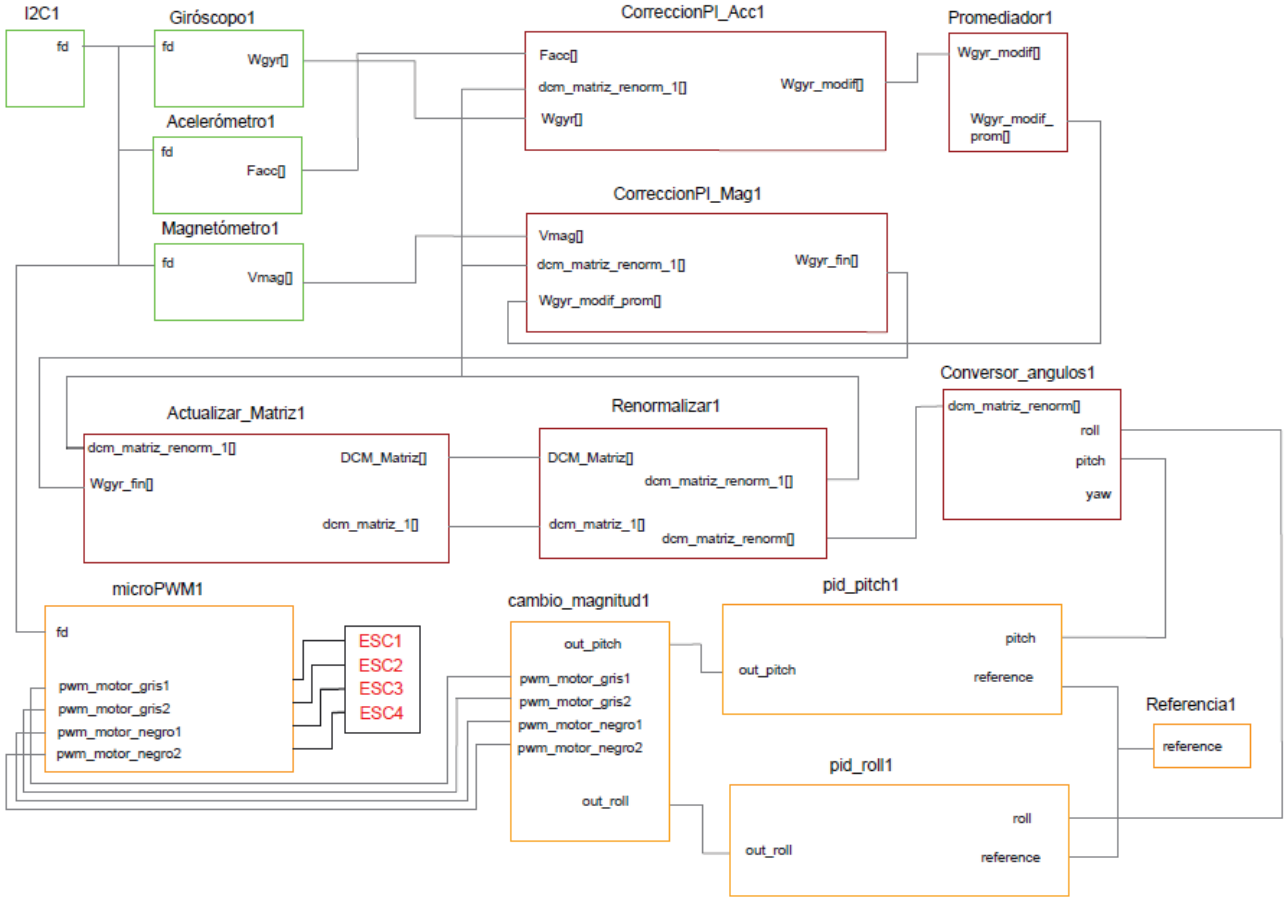

Fig. 4. Red de bloques función del cuadricóptero. Se identifican en diversos tonos los componentes asociados al sensado (verde), a la corrección y renormalización de las coordenadas de referencia (marrón) y a PID y bloques de gestión de los actuadores (naranja) para los ángulos roll y pitch

vantes del diseño en tiempo real, así como la posibilidad de almacenar el histórico de las mismas. En la Fig. 6 se observa cómo varía el ángulo *pitch* al recibir un estímulo. Parte del proceso de puesta en servicio y ajuste de variables se puede ver en [13].

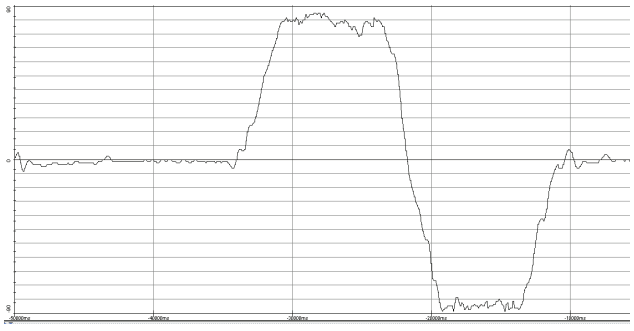

Fig. 6. Variación de ángulo pitch al recibir un estímulo. El comportamiento dinámico se puede en el blog de este proyecto [13]

VI. Conclusiones

El uso de SoC está cada vez más generalizado. Para poder utilizarlos en aplicaciones de automatización y control que exigen tiempos de respuesta del rango de ms. es necesario modificar el *kernel* del sistema opertativo para que tenga características de tiempo real.

Se ha descrito un procedimiento para modificar el *kernel* de una distribución Raspbian, utilizada en la plataforma Raspberry Pi. Se analizan los problemas asociados al cambio de versiones tanto de distribuciones como de hardware.

Una vez modificado, se realizan medidas de tiempo de latencia, que son coherentes con los encontrados en otros trabajos, concluyendo que, para aplicaciones muy exigentes con requerimientos de tiempo real, no son recomendables tiempos de ciclo por debajo de 10 ms.

Esta plataforma, que incluye la Raspberry Pi, el *kernel* modificado y COSME (entorno que permite ejecutar aplicaciones basadas en FB), se utiliza como base en una asignatura de sistemas empotrados del grado de ingeniería electrónica y automática para conseguir el control de estabilidad de un cuadricóptero. Con ello se consigue que el alumno tenga una herramienta práctica para comprender el comportamiento del

S.O. Linux en condiciones de tiempo real, comparándolo con las versiones que no disponen de esta característica.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por la Fundación Universitaria *"Antonio Gargallo"*, el proyecto OTRI num. 0079/2013. También agradecemos a S. Cervantes su colaboración en el mismo.

REFERENCES

[1] Raspberry Pi foundation. URL: www.raspberrypi.org.

[2] *Raspbian distro*. URL: http://www.raspbian.org/.

[3] *preemtive rt kernel repository*. URL: https://www.kernel.org/pub/linux/kernel/projects/rt/.

[4] Xenomai org. *Xenomai project*. URL: http://xenomai.org/.

[5] *RTAI - the RealTime Application Interface for Linux*. URL: https://www.rtai.org/.

[6] *GPIO Interface library for the Raspberry Pi*. URL: http://wiringpi.com/.

[7] *COSME RT repository for Raspber*. URL: https://github.com/COSMEcontrol.

[8] J. Hwan Koh and B. Wook Choi. "Real-time Performance of Real-time Mechanisms for RTAI and Xenomai in Various Running Conditions". In: *Int. J. of Control and Automation* 6.1 (2013), pp. 235–246.

[9] *The cyclictest repository*. URL: https://rt.wiki.kernel.org/index.php/Cyclictest.

[10] *How fast is fast enough? Choosing between Xenomai and Linux for real-time applications*. Proceedings of the Real Time Linux Workshops. 2010. URL: https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf.

[11] J.O. Hamblen. "Using a Low-Cost SoC Computer and a Commercial RTOS in an Embedded Systems Design Course". In: *IEEE J. EDU.* 51.3 (2008), pp. 356–363. DOI: 10.1109/TE.2008.919662. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4589060.

[12] C. Catalan et al. "COSME: A distributed control platform for communicating machine tools in Agile Manufacturing Systems". In: *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*. Sept. 2011, pp. 1–8. DOI: 10.1109/ETFA.2011.6059115.

[13] *The COSME project*. URL: http://cosmecontrol.blogspot.com.es/.